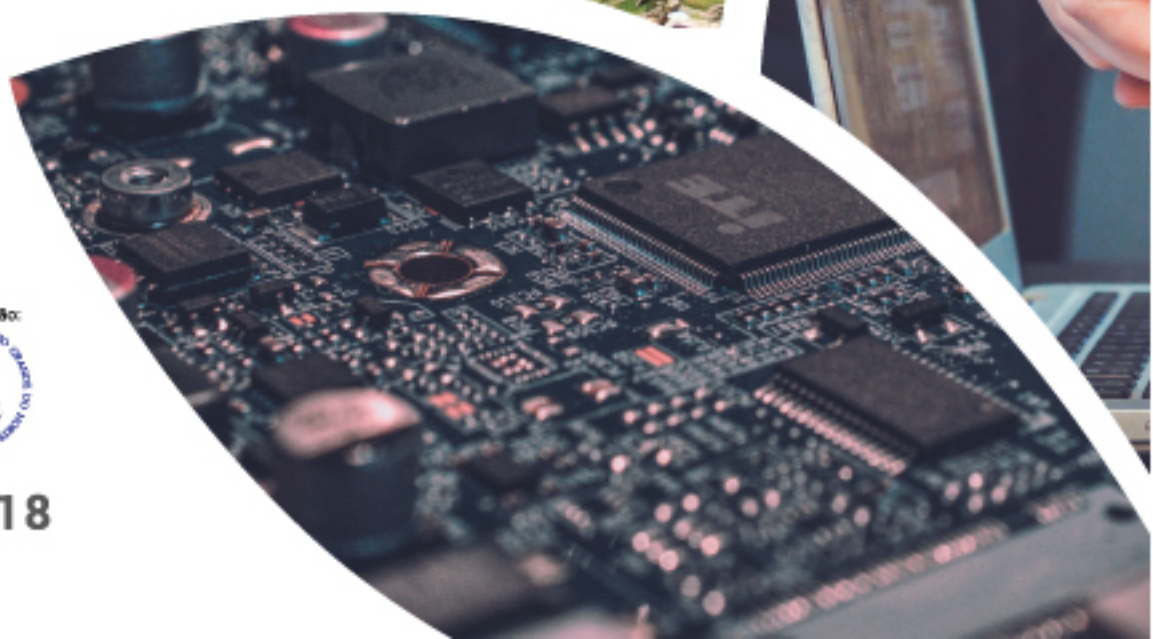


anais 2018

XXXVIII CONGRESSO DA SOCIEDADE BRASILEIRA DE COMPUTAÇÃO
17º WPERFORMANCE – WORKSHOP EM DESEMPENHO DE SISTEMAS COMPUTACIONAIS E DE COMUNICAÇÃO
CENTRO DE CONVENÇÕES | NATAL•RN | 22 A 26 DE JULHO DE 2018
#COMPUTAÇÃOESUSTENTABILIDADE



NATAL, 2018

cnais 2018

XXXVIII CONGRESSO DA SOCIEDADE BRASILEIRA DE COMPUTAÇÃO
CENTRO DE CONVENÇÕES | NATAL•RN | 22 A 26 DE JULHO DE 2018
#COMPUTAÇÃOESUSTENTABILIDADE



Coordenador Geral

Francisco Dantas de Medeiros Neto (UERN)

Comissão Organizadora

Bartira Paraguaçu Falcão Dantas Rocha (UERN)

Camila Araújo Sena (UERN)

Everton Ranielly de Sousa Cavalcante (UFRN)

Felipe Torres Leite (UFERSA)

Ilana Albuquerque (UERN)

Isaac de Lima Oliveira Filho (UERN)

Priscila Nogueira Krüger (UERN)

Realização

Sociedade Brasileira de Computação

Organização

Universidade do Estado do Rio Grande do Norte

CSBC 2018

XXXVIII Congresso da Sociedade Brasileira de Computação

Apresentação

Estes anais registram os trabalhos apresentados durante o XXXVIII Congresso da Sociedade Brasileira de Computação (CSBC 2018), realizado em Natal-RN, de 22 a 26 de julho 2018. O evento teve como tema central a Computação e Sustentabilidade, pois se compreende que o avanço da computação e as questões ambientais devem caminhar lado-a-lado, tendo em vista que as técnicas computacionais necessitam ser usadas para possibilitar o desenvolvimento sustentável, e, desse modo, equilibrar as necessidades ambientais, econômicas e sociais.

Organizar o maior evento acadêmico de Computação da América Latina foi um privilégio e um desafio. Foi enriquecedor promover e incentivar a troca de experiências entre estudantes, professores, profissionais, pesquisadores e entusiastas da área de Computação e Informática de todo o Brasil. Ao mesmo foi desafiador termos que lidar, principalmente, com às dificuldades impostas pelo momento de crise que o nosso Brasil vem enfrentando. Uma crise que afeta diretamente nossas pesquisas e, consequentemente, o desenvolvimento e inovação do nosso amado Brasil.

Por meio de seus 25 eventos, o CSBC 2018 apresentou mais de 300 trabalhos, várias palestras e mesas-redondas. O Congresso ainda abrigou diversas reuniões, que incluem a reunião do Fórum de Pós-Graduação, a reunião do CNPq/CAPES, a reunião dos Secretários Regionais SBC, a reunião das Comissões Especiais e a reunião do Fórum IFIP/SBC.

O sucesso do CSBC 2018 só foi possível devido à dedicação e entusiasmo de muitas pessoas. Gostaríamos de agradecer aos coordenadores dos 25 eventos e aos autores pelo envio de seus trabalhos. Além disso, gostaríamos de expressar nossa gratidão ao Comitê Organizador, por sua grande ajuda em dar forma ao evento; e, em especial, à equipe da Sociedade Brasileira de Computação (SBC), por todo apoio.

Por fim, reconhecemos a importância do apoio financeiro da CAPES, do CNPq, do CGI.br, do Governo do Estado do Rio Grande do Norte, da Prefeitura Municipal do Natal, da Prefeitura Municipal de Parnamirim, da CABO Telecom, da ESIG Software e Consultoria, da DynaVideo e do SENAI.

Natal (RN), 26 de julho de 2018.

Chico Dantas (UERN)
Coordenador Geral do CSBC 2018

Anais do CSBC 2018

**17º WPERFORMANCE – WORKSHOP EM
DESEMPENHO DE SISTEMAS
COMPUTACIONAIS E DE COMUNICAÇÃO**

17º WPERFORMANCE WORKSHOP EM DESEMPENHO DE SISTEMAS COMPUTACIONAIS E DE COMUNICAÇÃO

Apresentação

O Workshop em Desempenho de Sistemas Computacionais e de Comunicação, conhecido como WPerformance, é um fórum para discussão e disseminação de ideias sobre métodos, modelos, técnicas e ferramentas para avaliação de desempenho de sistemas computacionais e de comunicação. O WPerformance se propõe a receber artigos relacionados a problemas de desempenho das mais diversas áreas de hardware, software e aplicações da computação (por exemplo, redes de computadores, sistemas distribuídos, sistemas ciberfísicos, bancos de dados e sistemas operacionais).

Dezesseis edições consecutivas do WPerformance já foram realizadas pela comunidade, no período de 2002 a 2017, todas em conjunto com o Congresso anual da SBC (CSBC).

Na medida em que os sistemas de computação tornam-se cada vez maiores e mais complexos, faz-se necessária sua avaliação do desempenho usando abordagens empíricas e/ou modelos matemáticos. A atual diversidade dos sistemas computacionais e de comunicação envolve vários parâmetros que têm impactos diretos e indiretos em seu desempenho. O estudo destes sistemas via medição, emulação, simulação e modelagem permite não só o melhor entendimento dos mesmos, mas também sua otimização.

É com muita alegria que damos as boas vindas aos participantes do 17º Workshop em Desempenho de Sistemas Computacionais e de Comunicação (WPerformance 2018). O WPerformance é um fórum consolidado para discussão e disseminação de ideias sobre métodos, modelos, técnicas e ferramentas para avaliação de desempenho de sistemas computacionais e de comunicação que, ao longo dos anos, cumpre o papel importante de reunir especialistas de áreas diversas ao redor do tema chave: análise de desempenho.

Antes de comentarmos sobre os artigos e a fundamental contribuição dos autores, gostaríamos de compartilhar com todos um pouco do nosso trabalho como coordenadores de programa do WPerformance 2018. Este ano, abrimos a possibilidade de submissão de artigos para a trilha de artigos curtos, além da trilha de artigos completos. A ideia foi fomentar discussões entre autores que ainda estão em fase inicial em seus trabalhos. A trilha de artigos completos foi um sucesso, obtendo um número recorde de submissões, um total de 52 submissões, enquanto a de artigos curtos teve 4 submissões.

Ao todo, convidamos 72 professores/pesquisadores para comporem o comitê de programa, dos quais 59 aceitaram, 5 declinaram e os outros não responderam. Dos 59 membros do comitê, 52 membros completaram todas as revisões em tempo, uma grande maioria, aos quais somos muito gratos. Também agradecemos os 23 revisores adicionais que colaboraram com todo o processo.

Dos 52 artigos submetidos, 19 foram aceitos, com uma taxa de aceitação próxima de 36%. Devido ao grande potencial de contribuição de alguns artigos que não puderam ser aceitos como completos, esses foram convidados a enviar uma versão reduzida. Acreditamos que todos os artigos convidados como curto têm enorme potencial e podem estimular discussões bastante frutíferas durante o evento. Todos os artigos tiveram ao menos três revisões, sendo que a

maioria dos artigos tiveram 4 ou até 5 revisões. Cabe mencionar que as revisões do artigo no qual o Miguel é co-autor foram geridas de forma independente pelo Alfredo, que garantiu o anonimato dos revisores.

Para as apresentações orais os artigos foram divididos nas sessões técnicas: Computação em nuvem e em névoa, Segurança em redes, Economia de energia, Medidas em redes, IoT e redes sociais e Arquitetura de Computadores. Adicionalmente teremos mais duas seções com artigos curtos. A programação se apresenta, portanto, bastante rica, o que acreditamos ser um ponto de destaque adicional.

Por último, gostaríamos de agradecer a confiança em nós depositada e todo o apoio que recebemos da organização geral do CSBC. Gostaríamos também de agradecer aos membros da coordenação passada, professores Carlos Alberto Campos e Daniello Gomes, além do comitê consultivo do WPerformance, formado pelos professores Daniel Sadoc, Sidney Lucena, Pedro Velloso e Edmundo de Souza e Silva. Foi extremamente gratificante poder coordenar o comitê de programa do WPerformance deste ano e ainda ter a ajuda de todas essas pessoas admiráveis.

Miguel Elias M. Campista (UFRJ)
Alfredo Goldman (USP)
Coordenadores do WPerformance 2018

17º WORKSHOP EM DESEMPENHO DE SISTEMAS COMPUTACIONAIS E DE COMUNICAÇÃO (WPERFORMANCE)

Coordenação Geral

- Alfredo Goldman (USP)
- Miguel Elias M. Campista (UFRJ)

Coordenação Local

- André Gustavo Pereira da Silva (UERN)

Comitê de Programa

- Alex Borges Vieira (UFJF)
- Alfredo Goldman (USP)
- Ana Paula Couto da Silva (UFMG)
- Antônio Abelém (UFPA)
- Antonio Rocha (UFF)
- Antonio Tadeu Gomes (LNCC)
- Artur Ziviani (LNCC)
- Carlos Alberto Vieira Campos (UNIRIO)
- Christian Esteve Rothenberg (UNICAMP)
- Daniel Batista (USP)
- Daniel Cordeiro (USP)
- Daniel de Oliveira (UFF)
- Daniel Figueiredo (UFRJ)
- Daniel Menasche (UFRJ)
- Danielo G. Gomes (UFC)
- Dianne Medeiros (UFF)
- Djamel Fawzi Hadj Sadok (UFPE)
- Dorgival Guedes (UFMG)
- Edmundo de Souza e Silva (UFRJ)
- Edmundo Madeira (UNICAMP)
- Eduardo Cerqueira (UFPA)
- Eduardo Xavier (UNICAMP)
- Emilio Franceschini (UNICAMP)
- Fabio Porto (LNCC)
- Fabricio Murai (UFMG)
- Genaina Rodrigues (UnB)
- Gustavo Figueiredo (UFBA)
- Igor Moraes (UFF)
- Islene Garcia (UNICAMP)
- João Netto (UFRGS)
- José Ferreira de Rezende (UFRJ)
- José Augusto Suruagy Monteiro (UFPE)
- Jussara Almeida (UFMG)
- Lucas Schnorr (UFRGS)
- Luciano Paschoal Gaspary (UFRGS)

- Luidi Simonetti (UFRJ)
- Luis Henrique Costa (UFRJ)
- Luiz Fernando Bittencourt (UNICAMP)
- Luiz Manoel Rocha Gadelha Júnior (LNCC)
- Marcelo Carvalho (UnB)
- Miguel Elias Mitre Campista (UFRJ)
- Moises Ribeiro (UFES)
- Morganna Diniz (UNIRIO)
- Nelson Fonseca (UNICAMP)
- Paulo Aguiar (UFRJ)
- Paulo Maciel (UFPE)
- Pedro Velloso (UFRJ)
- Rafael Melo (UFBA)
- Rafael Pasquini (UFU)
- Raphael Camargo (UFABC)
- Ricardo Czekster (UNISC)
- Rodolfo Villaca (UFES)
- Rodrigo de Souza Couto (UERJ)
- Romulo Silva de Oliveira (UFSC)
- Rosa Leão (UFRJ)
- Sidney Lucena (UNIRIO)
- Thais Webber (UNISC)
- Thiago Noronha (UFMG)
- Tiberius Bonates (UFC)
- Wagner Meira Jr. (UFMG)

Revisores Adicionais

- Bruno Silva (IBM)
- Camilo Rodriguez (UFES)
- Carla Lintzmayer (UFABC)
- Carlos Senna (Unicamp)
- Caroline Maul de Araujo Lima (UFRJ)
- Cleomar Marques (UFF)
- Cristina Dominicini (IFES)
- Dalbert Mascarenhas (CEFET-RJ)
- Danilo Carastan dos Santos (UFABC)
- Denis Rosário (UFPA)
- Diego Cardoso (UFES)
- Erica Sousa (UFRPE)
- Gilmar Luiz Vassoler (IFES)
- Hugo Sadok (UFRJ)
- Ian Vilar Bastos (UFF)
- Jamilson Dantas (UFPE)
- João Gondim (UnB)
- Juliano Naves (IFRO)
- Luís Arcaro (UFSC)
- Luis Fernando Uria Garcia (UFES)
- Pedro Rezende (UFU)

- Rafael Guimarães (IFES)
- Roger Immich (Unicamp)
- Rubens de Souza Matos Júnior (IFS)
- Victor Fonseca (UFRJ)

SUMÁRIO

Artigos Completos

WorkflowSim4RL: Aprendizado por Reforço Aplicado a Escalonamento de Workflows Científicos em Nuvens	13
Victor Olimpio, André Nascimento, Aline Paes, Daniel de Oliveira	
Análise de Desempenho das Tecnologias de Virtualização de Rede da Plataforma OpenStack	27
Roseli Barbosa, Paulo Rego, Michel Bonfim, Arthur Callado	
Seshat: uma arquitetura de monitoração escalável para ambientes em nuvem	41
Vinicius Silva, Nestor Volpini, Dorgival Guedes	
Uma análise do consumo de energia de ambientes de processamento de dados massivos em nuvem	55
Nestor Volpini, Vinicius Silva, Raphael Pontes, Dorgival Guedes	
Uma Análise Competitiva entre as Tecnologias Blockchain e Tangle para o Projeto de Aplicações IoT	69
Rafael Zerbini, Carlo Rodrigues	
Contaminação Epidêmica em Redes: Imunidade Coletiva e Suas Implicações Frente a Atacantes Estratégicos	83
Vilc Rufino, Daniel Menasche, Italo Cunha, Cabral Lima, Leandro de Aguiar	
Detecção de Anomalias de Desempenho em Aplicações de Alto Desempenho baseadas em Tarefas em Clusters Híbridos	97
Vinicius Pinto, Lucas Schnorr, Arnaud Legrand, Samuel Thibault, Luka Stanisic, Vincent Danjean	
Análise de Características Comportamentais de Aplicações OpenMP para Redução do Consumo de Energia	111
Gabriel Moro, Lucas Schnorr	
Strategies to Improve the Performance and Energy Efficiency of Stencil Computations for NVIDIA GPUs	124
Pablo José Pavan, Matheus Serpa, Víctor Martínez, Edson Luiz Padoin, Jairo Panetta, Philippe Olivier Alexandre Navaux	
Consumo de Energia em Função da Taxa de Transmissão e do Tamanho do Pacote em Redes Ad Hoc Aloha de um Salto	135
Bruna da Silva, Renato de Moraes	
O Problema de Detecção e Localização de Eventos em Séries Temporais Aplicado a Redes de Computadores	149

Diego Ximenes Mendes, Gabriel Mendonça, Gustavo Santos, Edmundo de Souza e Silva, Rosa Leão, Daniel Menasche

Ranking Influential and Influenced Shares Based on the Transfer Entropy Network 163

José Neves Neto, Daniel Figueiredo

Caracterização do Uso de uma Rede Sem Fio de Grande Porte Distribuída por uma Ampla Área 177

Luiz Magalhaes, Diogo Mattos

Avaliação de Desempenho do IEEE 802.11 para Coleta de Grande Volume de Dados em Florestas com Auxílio de Drones 191

Caroline Maul de Araujo Lima, Eduardo Silva, Pedro Velloso

Um Mecanismo de Comutação de Servidores CoAP para Aumento de Disponibilidade dos Serviços de IoT 205

Jasiel G. Bahia, Miguel Elias M. Campista

Análise de Algoritmos de Clusterização para Experimentos Randomizados em Redes Sociais de Larga Escala 219

Bruno Nogueira, Francisco Azevedo, Fabricio Murai, Ana Paula Couto da Silva

Análise e Paralelização de um algoritmo para Griding 233

Carlos Aguni, Daniel Cordeiro

Back to the Past: When Segmentation Is More Efficient Than Paging 247

Lauri Laux, Roberto Hexsel

Método Rápido de Agrupamento de Vértices para Detecção de Comunidades em Redes Complexas de Larga-escala 260

Gustavo Carnivali, Alex Vieira, Paulo Antonio Esquef, Artur Ziviani

Artigos Curtos

Design, Implementation and Evaluation of IPv4/IPv6 Longest Prefix Match support in P4 Dataplanes 274

Fabricio Rodriguez Cesen, Pattam Gyanesh Kumar Patra, Christian Esteve Rothenberg, Gergely Pongracz

Network Address Translation using a Programmable Dataplane 280

Juan Mejia, Daniel Lazkani Feferman, Christian Esteve Rothenberg

Análise do Impacto da Elasticidade em Nuvens Computacionais Híbridas 286

Emanuel Coutinho, Danielo G. Gomes, Maurício Moreira Neto, Leonardo Moreira, José De Souza

Simulação escalável de epidemias em redes baseadas em passeios aleatórios com caracterização de transições de fase 292

João Vitor Tavares, Giulio Iacobelli, Daniel Figueiredo

Uma Análise de Desempenho sobre uma MicroFog 298
Douglas Gonçalves, Paulo Rocha, Tomaz Bisneto, Paulo Rego, Francisco Airton Silva

Uso de Simulador de Geração de Tráfego para a Melhoria de Desempenho de Sistemas Legados Críticos 304
Eduardo M. Teixeira, Mateus Bezerra, Renan Filgueiras

Modelos de Resposta para Experimentos Randomizados em Redes Sociais de Larga Escala 310
Francisco Azevedo, Bruno Nogueira, Fabricio Murai, Ana Paula Couto da Silva

SmartLB: Proposta de um balanceador de carga para redução de tempo de execução de aplicações em ambientes paralelos 316
Vinicius dos Santos, Edson Luiz Padoin, Philippe Olivier Alexandre Navaux, Jean-François Méhaut

Construção de um Modelo de Simulação para Escalonamento de Processos não-preemptivos 322
Jhonatan Thallisson Cabral Néry, Franciny Barreto, Joslaine Jeske de Freitas

Comparing Parallel Algorithms for Van der Waals Energy with Cell-List Technique for Protein Structure Prediction 328
Daniel Rodrigo Ferraz Bonetti, Gesiel Rios Lopes, Alexandre Delbem, Paulo Lopes de Souza, Kalinka Castelo Branco, Gonzalo Travieso

Geração de carga de trabalho ciente do desempenho dinâmico do sistema 334
Lourenço Pereira Jr, Regina H. Carlucci Santana, Marcos José Santana, Francisco José Monaco

WorkflowSim4RL: Aprendizado por Reforço Aplicado ao Escalonamento de Workflows Científicos em Nuvens *

Victor Olimpio¹, André Nascimento¹, Aline Paes¹, Daniel de Oliveira¹

¹Instituto de Computação – Universidade Federal Fluminense (UFF)
Av. Gal. Milton Tavares de Souza, s/nº
Boa Viagem - Niterói/RJ - Brasil

{victorolimpio, anascimento}@id.uff.br, {alineaes,danielcmo}@ic.uff.br

Resumo. Ambientes de Nuvem são amplamente utilizados para a execução de experimentos científicos modelados como workflows. Muitos Sistemas de Gerência de Workflows Científicos (SGWfC) oferecem apoio a execuções na nuvem. Em geral, esses SGWfCs possuem um escalonador de atividades que segue um modelo de custo bem definido. Um problema de se criar modelos de custos para escalonar workflows na nuvem é que precisamos modelar manualmente grande parte das características do ambiente. Entretanto, eventos como migração de máquinas virtuais ou flutuações de desempenho não são facilmente modelados. De forma a gerar um escalonamento sem a necessidade de modelar todas as características do ambiente de antemão, esse artigo propõe um simulador chamado WorkflowSim4RL que utiliza técnicas de Aprendizado por Reforço (AR - da área de aprendizado de máquina) para descobrir como melhor escalonar as ativações do workflow. Por meio do AR o escalonador não necessita conhecer todas as características do ambiente e do workflow sendo executado. Experimentos com traces de workflows reais mostraram a viabilidade da proposta.

1. Introdução

Nas últimas décadas o modo de se 'fazer ciência' mudou radicalmente. Experimentos científicos passaram a se beneficiar fortemente da infraestrutura computacional existente. Uma nova categoria de experimentos surgiu: os experimentos *in silico* [Travassos and Barros 2003]. Nesse tipo de experimento, todo o processo de experimentação é baseado em simulações computacionais. Muitas dessas simulações são compostas pela execução de diversas aplicações científicas que possuem dependência de dados entre si, *i.e.*, os dados produzidos por uma aplicação são consumidos pela próxima aplicação no fluxo [Mattoso et al. 2009].

Apesar de existirem diversas formas de implementar o encadeamento de programas dentro desses experimentos (*scripts* Python, *bash scripts*, etc.), uma das formas mais utilizadas são os chamados *workflows* científicos [Deelman et al. 2009]. Os *workflows* científicos permitem aos cientistas modelar as etapas do experimento como um grafo, no qual vértices representam atividades de processamento de dados e arestas representam dependências entre as mesmas. Cada atividade do *workflow* está associada a um programa de computador (ou um serviço *Web*), e cada execução de uma atividade com um determinado conjunto de parâmetros é chamada de *ativação*. Cada ativação processa um conjunto de dados de entrada e produz um conjunto de dados de saída [Ogasawara et al. 2011].

Os *workflows* são gerenciados por sistemas de gerência de *workflows* (SGWfC) que são sistemas complexos que apoiam a composição, execução e análise dos resultados dos

*Este artigo foi financiado parcialmente pelo CNPq, CAPES e FAPERJ

experimentos [Deelman et al. 2009]. Considerando que muitos *workflows* consomem e produzem um grande volume de dados e podem processar durante muito tempo, ambientes de processamento de alto desempenho (PAD) devem ser empregados na execução dos *workflows*. Tradicionalmente, esses experimentos têm sido executados em *clusters* de computadores e grades computacionais, mas nos últimos anos as nuvens de computadores têm ganhado importância no cenário científico.

As nuvens de computadores são capazes de fornecer um ambiente favorável para a execução de *workflows*, pois além da disponibilidade, os cientistas podem explorar a elasticidade, *i.e.* a capacidade de aumentar e/ou diminuir a quantidade de máquinas virtuais (VMs) criadas para executar o *workflow*. Na nuvem, o cientista paga apenas pelos recursos que consumir (*pay-per-use*) [Vaquero et al. 2008]. Para que a nuvem possa ser utilizada em sua capacidade máxima, os SGWfCs existentes devem ser capazes de distribuir as execuções dos *workflows* nestes ambientes. Existem diversos SGWfCs que apoiam a execução em nuvens como o Pegasus [Deelman et al. 2007], o Swift/T [Zhao et al. 2007] e o SciCumulus [Oliveira et al. 2012], que foi o SGWfC utilizado neste artigo.

Todos esses SGWfCs possuem escalonadores específicos para o ambiente de nuvem, com modelos de custo definidos de antemão, e considerando critérios como desempenho, custo financeiro, disponibilidade, etc. Cada escalonador é baseado em um modelo de custo que busca formalizar as características do ambiente de nuvem e distribuir as ativações nesse ambiente. Entretanto, existem características do ambiente que são difíceis de serem modeladas ou formalizadas, como por exemplo flutuações de desempenho das VMs e migrações de VMs durante a execução do experimento (*live migration*). Além disso, tais escalonadores dependem de uma boa estimativa do desempenho das ativações (estimativa de tempo de execução), o que pode ser complicado caso o cientista não possua um histórico de execuções anteriores do mesmo *workflow*, ou caso o ambiente não possua recursos de monitoramento. Dessa forma, idealmente, o escalonador dos SGWfCs deveria ser adaptativo ao ambiente, sem que seja imperativo modelar ou formalizar de antemão todas as características do ambiente.

Tal necessidade vai ao encontro a uma área de Aprendizado de Máquina chamada Aprendizado por Reforço [Sutton and Barto 1998]. No Aprendizado por Reforço, um agente específico (no contexto desse artigo o escalonador) deve ser capaz de aprender como se comportar em um ambiente dinâmico (ambiente de nuvem, nesse caso) por meio de interações do tipo “tentativa e erro” (possíveis escalonamentos). A cada tentativa, o agente deve avaliar se a mesma foi uma boa escolha ou não. Em caso de resposta positiva, o agente recebe uma *recompensa*, caso contrário uma *punição*. Tal técnica se encaixa na necessidade anteriormente levantada: o escalonador não precisa conhecer o ambiente e nem estimar o desempenho de ativações, e sim tem que ser capaz de avaliar se uma determinada tentativa (escalonamento) foi boa ou ruim. Tal avaliação é possível uma vez que é simples calcular se uma execução foi mais rápida que a anterior, apresentou um custo financeiro menor, *etc.*

Nesse artigo, apresentamos uma abordagem que propõe o uso de um algoritmo clássico de Aprendizado por Reforço, mais especificamente o *Q-Learning* [Sutton and Barto 1998], aplicada ao problema de escalonamento de *workflows* em recursos computacionais no ambiente de nuvem. Tal abordagem foi implementada no simulador de *workflows* em nuvem chamado *WorkflowSim* [Chen and Deelman 2012] gerando o *WorkflowsimARL*. A ideia é se aproveitar de um simulador que seja capaz de gerar um plano de escalonamento *a priori* que possa ser informado e seguido pelo SGWfCs. O desempenho da abordagem proposta foi avaliado tanto de forma simulada quanto com a execução do plano de escalonamento gerado

pelo *Workflowsim4RL* em um ambiente de nuvem real utilizando o SGWfC SciCumulus. Foram realizadas avaliações com traces sintéticos de *workflows* e com um *workflow* real da área de astronomia. Os resultados demonstram a viabilidade da abordagem proposta.

O restante deste artigo está estruturado da seguinte maneira. A Seção 2 apresenta referencial teórico. A Seção 3 apresenta a abordagem proposta. A Seção 4 apresenta a avaliação experimental. A Seção 5 aborda trabalhos relacionados, e, finalmente, a Seção 6 conclui o artigo e discute possíveis trabalhos futuros.

2. Referencial Teórico

Nessa seção abordaremos os conceitos básicos de Aprendizado por Reforço, o algoritmo *Q-Learning* utilizado no artigo, e definições básicas referentes a *workflows* científicos.

2.1. Aprendizado por Reforço

Quando estamos aprendendo alguma atividade nova, comumente tendemos a repetir e recombinar os passos para executar determinada tarefa até que consigamos de fato nos convencer de que a atividade foi aprendida. Esse processo acontece dentro de um ciclo de laços de repetições. Além disso, quem está aprendendo, não deve ter influência externa quanto ao que deve ou não fazer, este deve por conta própria descobrir quais decisões tomar para que consiga aumentar as recompensas que recebe do ambiente. E por fim, as ações podem não apenas influenciar a recompensa imediata, mas também situações que se sucederão e, por consequência, suas recompensas. Essas três características - ser por essência fechado em repetições, não ter nenhum tipo de instrução prévia relacionada à escolha de uma ação, e as consequências de ações serem capazes de ter efeito longínquo em situações vindouras - são as três características mais importantes que distinguem o Aprendizado por Reforço [Sutton and Barto 1998] das demais técnicas de Aprendizado de Máquina.

Assim, enquanto as técnicas baseadas em *aprendizado supervisionado* são as mais utilizadas para resolver problemas que requerem Aprendizado de Máquina, elas requerem um conjunto de exemplos anotados previamente, não sendo adequadas para induzir hipóteses em ambientes dinâmicos sem supervisão. Por outro lado, técnicas de *aprendizado não-supervisionado* focam em descobrir padrões ocultos em dados não anotados, mas possuem dificuldade em descobrir funções regendo a dinamicidade do ambiente, incluindo sequências de percepções e ações, e maximizar a valoração de tais ações para alcançar um objetivo. Um dos desafios que surgem no Aprendizado por Reforço, e não nos outros paradigmas, é a troca entre exploração (*exploration*) e exploração (*exploitation*). Para que o agente de Aprendizado por Reforço não perca valores já testados de recompensa, este pode preferir ações que já foram previamente testadas e que se provaram efetivas. No entanto, o agente deve tentar ações que não foram testadas anteriormente a fim de não ficar preso em valores sub-ótimos.

Desta forma, o agente de Aprendizado por Reforço deve explorar o que já conhece a fim de obter recompensas, ao passo que também deve explorar ações que não foram escolhidas anteriormente para que melhores escolhas de ações sejam realizadas futuramente. No entanto, o agente de Aprendizado por Reforço não está isento de falhar em suas escolhas durante os processos de exploração e exploração. Portanto, o agente deve tentar uma variedade de ações e favorecer progressivamente aquelas que oferecem melhores recompensas. Uma estratégia simples pode ser escolher a melhor ação, digamos, 80% das vezes e escolher uma ação nova, selecionada aleatoriamente no restante. Outra característica importante em aprendizado por reforço consiste no agente de aprendizado estar focado diretamente em alcançar um objetivo, interagindo com um ambiente dinâmico, como pode ser observado na Figura 3. Todos os



Figura 1. *Interação do agente de aprendizado por reforço com o ambiente*

agentes de Aprendizado por Reforço possuem objetivos explícitos, são capazes de captar aspectos do ambiente onde se encontram, e podem escolher ações para influenciar seus ambientes, inclusive sendo capazes de atuar em ambientes que envolvem incerteza.

Política Em sistemas de aprendizado envolvendo agentes, dizemos que o agente é o responsável pelo que ele aprende e pelas decisões tomadas, de forma autônoma, enquanto o *ambiente* comprime tudo o que se encontra fora do agente. Ambos interagem entre si, com o agente escolhendo ações a serem executadas no ambiente e este por sua vez respondendo a estas ações. Além disso, o ambiente é capaz de retornar uma recompensa (um valor numérico) para cada ação do agente, que por sua vez tentará maximizar a recompensa recebida ao longo do tempo. A interação entre o agente e o ambiente ocorre ao longo do tempo, por meio de sequências de passos temporais $t = 0, 1, 2, 3, 4, \dots$, onde em cada passo t o ambiente envia ao agente uma representação de seu *estado*. Um agente, a cada passo, precisa decidir que ação executar, dentre as disponíveis, dado o estado do ambiente. A função que mapeia os estados do ambiente nas ações que o agente deve tomar é definida como a sua *política*. Uma decisão π_t em um estado de tempo t , oriunda de uma política π , define a probabilidade de se escolher uma ação a , dado um estado s . A política de um agente de aprendizado por reforço pode mudar ao longo do tempo devido a experiência que este vai adquirindo. Além disso, é objetivo do agente ser capaz de aumentar a quantidade de recompensa que recebe ao longo do tempo.

Episódios Em geral se deseja maximizar o *retorno esperado*, sendo G_t , definido como uma função a partir da sequência de recompensas. Em outras palavras $G_t = R_{t+1} + R_{t+2} + R_{t+3} + R_{t+4} + \dots + R_T$, onde T é o ultimo passo temporal. Isso faz sentido ao considerarmos situações onde se tem noção de fim, em que a interação entre o ambiente e o agente pode ser separada em subsequências, chamadas de *episódios*. Durante o aprendizado do agente, consideramos que existe uma sequência de episódios, onde cada um consiste em uma sequência finita de passos temporais, sendo que numeramos cada passo temporal a partir do zero no início de cada episódio. Assim, nos referimos a um estado em um episódio não apenas como S_t mas sim como $S_{t,i}$ ou seja, esse é o estado S_t do episódio i .

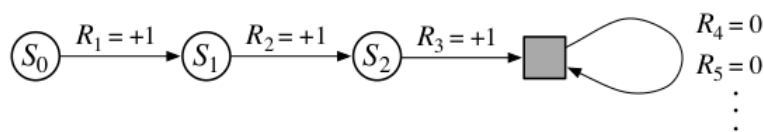


Figura 2. *Autômato que representa o retorno esperado [Sutton and Barto 1998]*

A propriedade de Markov Para descobrir a política a ser seguida no ambiente, o agente de Aprendizado assume que o ambiente é regido pela propriedade de Markov, e temos então um processo de decisão de Markov (MDP). Por simplicidade, vamos assumir que o número de estados e recompensas é finito. Tomando como exemplo um ambiente que responda no tempo $t + 1$ a uma ação tomada no tempo t , no caso mais simples, essa resposta depende de tudo que ocorreu anteriormente. Para definir essa situação podemos utilizar a seguinte distribuição de probabilidade conjunta:

$$Pr\{S_{t+1} = s', R_{t+1} = r | S_0, A_0, R_0, \dots, S_{t-1}, A_{t-1}, R_{t-1}, S_t, A_t, R_t\} \quad (1)$$

para todo valor de recompensa r , estado s' e todos os possíveis valores dos eventos anteriores: $S_0, A_0, R_0, \dots, S_{t-1}, A_{t-1}, R_{t-1}, S_t, A_t, R_t$.

No entanto, se o estado do ambiente possui a propriedade de Markov, quando o ambiente responde no tempo $t + 1$, a resposta depende apenas do evento anterior, em outras palavras, depende apenas do estado s e ação a no tempo t , conforme a equação 2:

$$p(s', r | s, a) = Pr\{S_{t+1} = s', R_{t+1} = r | S_t = s, A_t = a\} \quad (2)$$

Se o ambiente possui a propriedade Markov, podemos prever qual o próximo estado do ambiente e a recompensa esperada, dado o estado atual do ambiente e a ação tomada pelo agente. Assim, podemos definir o conceito de Processo de Definição de Markov (MDP) que inclui: (i) Um conjunto finito de estados S ; (ii) Um conjunto de ações disponíveis para cada estado A ; (iii) Transições entre os estados, usualmente dispostas como uma distribuição de probabilidades ligando um estado $s \in S$ aos seus possíveis sucessores $s' \in S$; (iv) Uma função de recompensa $R(s, a)$, $a \in A$ e $s \in S$; e (v) Um fator de desconto $0 \leq \gamma \leq 1$. Conforme dito anteriormente, um processo de decisão de Markov segue a premissa de que o futuro é independente do passado, dado o presente.

Q-Learning Dados os conceitos de política, episódios e processo de decisão de Markov, podemos apresentar o algoritmo de Aprendizado por Reforço utilizado neste artigo, a saber o algoritmo Q-learning [Sutton and Barto 1998]. Em Aprendizado por Reforço, tentamos maximizar a soma das recompensas ao longo do tempo, ou seja:

$$\sum_{t=0}^{t=\infty} \gamma^t r_t(s, a) \quad (3)$$

Como o valor das recompensas futuras não é conhecido, podemos usar um algoritmo de iteração de valor para estimá-lo. Assim, o algoritmo *Q-learning* tem como objetivo aprender uma estimativa do valor do melhor mapeamento entre estados e ações, usando uma abordagem livre de modelo que se baseia em uma função de valoração de ações chamada Q . A função Q recebe um estado e uma ação e retorna a recompensa futura esperada para aquela ação naquele estado. A ideia é que o valor retornado pela função Q cada vez se aproximará mais do valor real, dada a exploração no ambiente. A Equação 4 exhibe como esse valor é calculado e atualizado.

$$Q(s_t, a_t) \leftarrow Q(s_t, a_t) + \alpha^t \times (r(s_t, a_t) + \gamma^t \times \max_a Q(s_{t+1}, a) - Q(s_t, a_t)) \quad (4)$$

onde $s \in S$, $a \in A$, r é a recompensa e α é a taxa de aprendizado, que define o quanto queremos atualizar o valor corrente de Q (quando α é próximo de 1, o valor antigo é

simplesmente substituído pelo novo). Observe que ambas a recompensa atual ($r(s_t, a_t)$) e a recompensa futura estimada (o máximo valor esperado no estado sucessor, para cada ação, $\max_a Q(s_{t+1}, a)$) são levados em consideração ao se computar o valor da função Q . O valor encontrado pela da função é usado para determinar a ação ser executada em cada estado, usando a abordagem ϵ -greedy, em que, a cada iteração, com uma certa probabilidade é escolhida a melhor ação, ou uma ação aleatória, caso contrário. Conforme pode ser visto no Algoritmo 1, o processo de estimação do valor da função Q se repete para um certo número de episódios.

Algoritmo 1: Algoritmo Q-learning

Entrada: γ, α, ϵ

```

1 Inicialize  $Q(s, a) \forall s, a, s \in S, a \in A$ , de forma arbitrária
2 para cada episódio e faça
3    $t \leftarrow 0$ 
4   escolha um estado inicial aleatório
5   enquanto não atingir o objetivo faça
6     observe o estado  $s_t$ 
7     com uma probabilidade  $\epsilon$  escolha a melhor ação  $a$  para  $s_t$  de acordo com
        $Q(s, a)$ ; caso contrário escolha uma ação aleatória  $a$ 
8     execute  $a$  no ambiente
9     receba a recompensa  $r(s, a)$ 
10    observe o próximo estado  $s_{t+1}$ 
11     $\delta \leftarrow (r(s_t, a_t) + \gamma^t \times \max_a Q(s_{t+1}, a) - Q(s_t, a_t))$ 
12     $Q(s, a) \leftarrow Q(s_t, a_t) + \alpha^t \delta$ 
13     $t \leftarrow t + 1$ 
14  fim
15 fim

```

2.2. Workflows Científicos em Nuvens

Muitos experimentos científicos *in silico* são compostos pelo encadeamento de múltiplas combinações de programas que podem consumir grande quantidade de dados, e são comumente executados em ambientes de processamento de alto desempenho (PAD). Nesses experimentos, cada programa é executado consumindo um grupo específico de parâmetros e dados, cada qual com a sua própria semântica e sintaxe. A saída de um programa é normalmente utilizada como entrada para outro programa no encadeamento [Deelman et al. 2009, Mattoso et al. 2010]. *Workflows* são abstrações utilizadas para estruturar esse encadeamento de forma que possa ser executado e gerenciado por sistemas de *workflows*. O formalismo exposto nessa seção é baseado em [Oliveira et al. 2012].

Dado um *workflow* W , um conjunto $Ac = ac_1, ac_2, \dots, ac_k$ de ativações é criado para a execução paralela de W . Cada ativação ac_i está associada a uma atividade a_i específica que é representada como $Act(ac_i) = a_i$. O conjunto de ativações associadas a uma determinada atividade a_i do *workflow* W é denotada como $Ac(a_i)$. Consideremos também $VM = \{VM_0, VM_1, \dots, VM_{n-1}\}$ como o conjunto de n VMs criadas na nuvem e disponíveis para execução do *workflow*. Considere também $\varphi(W, VM)$ como sendo o escalonamento total das ativações pertencentes ao conjunto AC do *workflow* W em VM . Dado um *workflow* W que possui uma série de atividades $A = \{a_1, a_2, \dots, a_n\}$ e um conjunto de ativações $AC = \{ac_1, ac_2, \dots, ac_k\}$, seja $\varphi(W, VM) = \{sched_1, sched_2, \dots, sched_k\}$ o conjunto dos escalonamentos para execução paralela do conjunto de ativações AC . O escalonamento de uma ativação é definido como $sched(ac_i, VM_j, inicio, fim)$, em que *início* é o tempo de

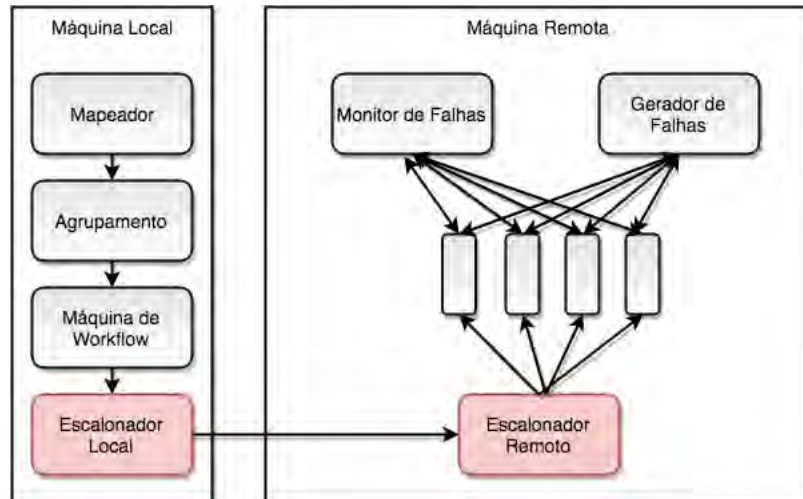


Figura 3. Arquitetura do WorkflowSim

início de execução de uma ativação ac_i em uma dada VM_j e fin o tempo fim. Com o uso de algoritmos de aprendizado por reforço são definidos diversos escalonamentos (*sched*) ao longo da execução do *workflow*, a fim de 'aprender' qual o melhor escalonamento para o *workflow* e o ambiente.

3. Abordagem Proposta: *WorkflowSim4RL*

Nessa seção apresentamos como o *WorkflowSim4RL* foi estendido a partir do *WorkflowSim* e como o mesmo realiza o escalonamento baseado em aprendizado por reforço.

3.1. O *WorkflowSim*

Segundo [Chen and Deelman 2012, Travassos and Barros 2003], a simulação é um dos métodos de avaliação mais populares em estudos baseados em *Workflows* Científicos, uma vez que a execução de testes reais em larga escala pode ser custosa tanto financeiramente quanto em relação ao tempo necessário. Dessa forma, uma vez que a proposta de algoritmos de aprendizado por reforço é executar diversas vezes um *Workflow* em um tempo hábil, para simular um ambiente com sobrecargas provenientes de um sistema heterogêneo e sujeito a falhas, optamos pela extensão de um simulador já amplamente conhecido, o *WorkflowSim* [Chen and Deelman 2012], justamente por fornecer um arcabouço capaz de lidar com tais questões. A arquitetura original do *WorkflowSim* é apresentada na Figura 3, onde os componentes em vermelho representam as extensões propostas nesse artigo (*WorkflowSim4RL*).

O *WorkflowSim* é composto por 7 componentes principais: (i) o mapeador que lê um arquivo de entrada e cria os objetos em memória com a estrutura do *workflow*, (ii) o componente de agrupamento que agrupa ativações para tornar o escalonamento mais eficiente, (iii) a máquina de *workflow* que controla as dependências e informa quais ativações podem executar, (iv) o escalonador local, que decide quais ativações são despachadas para o ambiente de processamento de alto desempenho, (v) o monitor de falhas que verifica a ocorrência de falhas nas máquinas remotas, (vi) o gerador de falhas que se encarrega de gerar falhas na execução, simulando um ambiente de nuvem real e (vii) o escalonador remoto que controla a execução das ativações nas máquinas remotas. Uma das grandes vantagens de se utilizar o *WorkflowSim* é que o mesmo é capaz de simular questões como falhas e

flutuações de desempenho de forma bastante consistente e perto do ambiente real, fazendo com que ele seja uma ferramenta bastante apropriada para utilizarmos nesse artigo. A seguir, apresentamos como o *WorkflowSim* foi estendido para acomodar o algoritmo *Q-learning* para o escalonamento de ativações.

3.2. *WorkflowSim4RL: Estendendo o WorkflowSim*

Inicialmente, para que seja possível utilizar a abordagem de Aprendizado por Reforço no *WorkflowSim* temos que verificar quais componentes devem ser estendidos. Os únicos componentes que necessitam extensões são o escalonador local e o escalonador remoto, uma vez que são esses componentes que decidem qual máquina executará quais ativações. Dessa forma, antes de se estender os escalonadores, é necessário definir uma forma de avaliar o ambiente. As máquinas virtuais são os recursos do ambiente, logo, o desempenho de cada uma delas é avaliado individualmente após executar uma ativação, bem como globalmente, considerando todo o conjunto de máquinas virtuais da nuvem. Isso é necessário para que possamos calcular o valor de recompensa/punição a cada decisão tomada pelo agente de Aprendizado por Reforço (*i.e.* o escalonador). Para tanto, foram definidas funções para calcular o desempenho de cada máquina virtual ao executar as ativações do *workflow*. Como métrica, foram utilizados o tempo de execução e o tempo de fila de espera.

Inspirados na função de desempenho proposta por [Costa et al. 2009] para ambientes de grade, definimos o desempenho de uma máquina virtual de acordo com o tempo de execução de uma ativação ac_i na máquina (te) em relação ao tempo de espera em fila (tf) para ser executada. Considerando tt (tempo total) como $tt = te + tf$, tais valores são usados para calcular o índice de desempenho (P_i) de uma máquina virtual ao executar uma ativação, $P_i = te_i * \alpha + (1 - \alpha) * (tf)$. No caso do desempenho global (todas as máquinas virtuais envolvidas na execução do *workflow*, já que a solução gerada deve ser boa para máquinas individuais, mas também o deve ser de forma global) $\overline{P_i} = \overline{te} * \alpha + (1 - \alpha) * (\overline{tf})$, sendo $\overline{te}$ e $\overline{tf}$ as médias dos tempos de execução e tempo de fila respectivamente. Além do desempenho individual da máquina e o desempenho global em uma execução, calculamos também o desempenho acumulado de cada máquina virtual, *i.e.* para calcular o desempenho de todas as execuções de uma determinada máquina vm_i , como $\overline{P_{i_i}} = \overline{te_i} * \alpha + (1 - \alpha) * (\overline{tf_i})$, onde $\overline{te_i}$ e $\overline{tf_i}$ são as médias dos tempos de execução e de fila de todas as execuções da máquina virtual respectivamente.

Foi definida também a função P_i que retorna um valor numérico que representa um índice de desempenho da máquina virtual ao executar uma ativação, dado o tempo em que tal ativação permaneceu na fila de espera da máquina para execução. Em suma, P_i representa o quão rápido uma máquina virtual foi capaz de executar uma ativação em relação ao tempo em que a ativação ficou sem ser executada. Desta forma, somos capazes de calcular o desvio padrão ($stdv$) de $\overline{P_i}$, e avaliamos a alocação da ativação para ser executada na máquina da seguinte forma: $r = -1$ se $P_i > (P_{im} + stdv)$ (é aplicada uma punição) e $r = 1$ se $P_i < (P_{im} - stdv)$ (é aplicada uma recompensa);

Por fim, calculamos a eficiência de uma máquina virtual atribuindo um valor chamado de (*nrle*) (nova eficiência) definido como $nrle = rle + \alpha * (r - rle)$, sendo rle o valor prévio de eficiência da máquina e α o parâmetro associado ao *Q-Learning* para definir a velocidade de aprendizado. O valor retornado pela equação de eficiência (*nrle*) nos dá a recompensa ou punição que são dados ao agente por uma escolha boa ou ruim de escalonamento, a serem utilizados no Algoritmo Q-Learning implementado neste artigo. De forma a se avaliar as ações do escalonador com aprendizado por reforço, foi necessário um mapeamento de todos os possíveis estados do ambiente em que o mesmo se encontra. Para isso, consideramos

que cada estado de uma tarefa dentro do grafo de dependências do *workflow* combinado com cada estado de uma máquina virtual formam um estado do próprio grafo, em outras palavras, o estado do grafo do *workflow* depende do estado da máquina e da ativação. A seguir apresentamos todos os possíveis estados do ambiente de nuvem, assumindo s como estados das ativações e S o conjunto de estados das ativações do *workflow*. Essa tabela é gerada e salva em arquivo após o término de cada simulação utilizando o agente de Aprendizado por Reforço de escalonador proposto neste artigo.

Definição dos Elementos do MDP para o Escalonamento em Workflows. Para aplicar o algoritmo $Q - Learning$ que assume que o ambiente se comporta como um MDP, é necessário definir os estados, ações e o modelo de transição. Considerando o conjunto M que define os estados dos recursos, ou seja, das máquinas disponíveis, temos que $M = \{idle, running\}$. Por outro lado, a definição do conjunto de estados referente ao Workflow requer mais cuidado, uma vez que este depende dos estados das ativações que o compõem. Assim, definimos o conjunto $S = \{ready, running, finished-with-failure, finished-with-success\}$ como sendo o conjunto de estados em que o workflow pode se encontrar em um determinado instante de tempo. As ativações do workflow, por sua vez, podem estar em um dos estados pertencentes ao conjunto $S' = \{ready, locked, running, finished-with-success, finished-with-failure\}$. Assim, os estados do workflow são definidos a partir dos estados de suas ativações, conforme abaixo:

- $s = ready \in S$ se $\exists$ uma tarefa com $s' = ready \in S'$
- $s = finished-with-success \in S$ se $\forall s' \in S', s' = finished-with-success$
- $s = finished-with-failure \in S$ se $\exists s' \in S', s' = finished-with-failure$
- $s = running \in S$ se $\nexists s' \in S' \mid s' = ready$

Vale ressaltar que o objetivo do Aprendizado por Reforço é que o workflow alcance o estado de *finished – with – success*. Como as ações envolvem alocar uma ativação do Workflow em uma máquina, temos o conjunto de ações $A = \{\{act_i \text{ to } m_j\}, end\}$, onde act_i é uma ativação do Workflow e m_j é uma máquina, $act_i \text{ to } m_j$ representa a alocação de uma ativação i para uma máquina j , e *end* significa encerrar a execução do Workflow. A transição de estados do Workflow encontra-se descrita na Tabela 1.

Tabela 1. Tabela de Transição de Estados do Workflow

Máquina	Estado	Próximo Estado	Ação
idle	ready	ready	$act_i \text{ to } m_j$
idle	ready	ready	-
idle	ready	running	$act_i \text{ to } m_j$
idle	running	ready	-
idle	running	running	-
idle	running	finished-with-success	end
idle	running	finished-with-failure	end
running	ready	ready	-
running	running	ready	-
running	running	running	-
running	running	finished-with-success	end
running	running	finished-with-failure	end

Consideramos que cada execução do *workflow* representa um episódio para o algoritmo *Q-Learning*. Desta forma, ao término de uma simulação, a tabela de estados de cada máquina virtual é salva em arquivo, também é atualizada para cada par (*estado*, *ação*), o valor da função de avaliação *Q* do *Q-Learning*. Além disso, o arquivo também contém o plano de escalonamento gerado naquele episódio, o tempo total de simulação até o fim do episódio correspondente ao arquivo, e o tempo simulado do *workflow* naquele episódio, a fim de auxiliar as avaliações realizadas a seguir.

Conforme discutido anteriormente, o *Q-Learning* depende da progressão dos episódios ao longo do tempo, tomando cada execução do simulador como um episódio. Assim, cada simulação é alimentada com dados da execução imediatamente anterior. Entretanto, isso gera um desafio no caso do primeiro episódio. Dados como o tempo de execução de uma ativação precisam ser fornecidos no primeiro episódio, afim de que seja possível calcular o índice de desempenho P_i , para que seja possível calcular o novo valor da função *Q* do *Q-Learning*. Como os métodos apresentados anteriormente estão em função do tempo de execução e tempo de espera em fila, esses dados de tempo precisam ser coletados antes do início de uma simulação. Assim, antes de iniciar o primeiro episódio, cada ativação foi executada individualmente a fim de coletarmos o tempo de execução simulado da ativação gerado pelo *WorkflowSim4RL*.

O escalonador *RFLSchedulingAlgorithm* implementa a utilização do escalonador aprendido, bem como o algoritmo *Q-learning* usado para aprendê-lo. O *RFLSchedulingAlgorithm* recebe uma lista de ativações ainda não escalonadas, e de máquinas virtuais disponíveis. As máquinas virtuais são escolhidas de acordo com o estado do grafo de dependências do *workflow* e o valor da função de avaliação *Q* do algoritmo *Q-Learning*, presente no arquivo gerado no episódio anterior.

Escolhida a máquina virtual para uma determinada ativação, é feita a alocação da ativação para ser executada, o valor de tempo de fila é calculado conforme mencionado anteriormente, o novo estado do *workflow* é calculado, e então o valor da função de avaliação *Q* é calculado para esse novo estado. Ao término desse processo, o simulador inicia a simulação normalmente, e ao término dela os dados obtidos no episódio atual são persistidos para serem usados no episódio seguinte.

4. Avaliação Experimental

De forma a avaliar o escalonamento com Aprendizado por Reforço, executamos uma série de experimentos simulados e um experimento real para avaliar a qualidade do plano de escalonamento aprendido. Para avaliar o *WorkflowSim4RL* utilizamos um conjunto variado de *traces* de *workflows* disponibilizados pela equipe do SGWfC Pegasus¹, a saber: (i) **Montage**: *workflow* criado pela NASA que cria um mosaico de várias imagens de entrada para criar uma foto personalizada do céu; (ii) **Cybershake**: *workflow* usado pelo Centro de Prevenções de Terremotos da Califórnia para caracterizar riscos de terremoto na região; (iii) **Epigenomics**: *workflow* que automatiza tarefas de mapeamento do genoma; (iv) **LIGO Inspiral**: *workflow* usado para gerar e analisar formas de onda gravitacionais de dados coletados durante o coalescência de sistemas binários compactos; e (v) **SIPHT**: *workflow* usado para automatizar a pesquisa de RNAs não traduzidos (sRNAs) para bactérias no banco de dados NCBI. Cada um desses *workflows* apresenta diferentes características, como ativações intensivas em CPU ou ativações intensivas em dados, por exemplo.

Além de variar o *workflow*, variamos a quantidade de ativações que cada *workflow*

¹<https://confluence.pegasus.isi.edu/display/pegasus/WorkflowGenerator>

executou, a saber: Montage (25, 50, 100 e 1000 ativações), Cybershake (30, 50, 100 e 1000 ativações), Epigenomics (24, 46, 100 e 997 ativações), LIGO Inspiral (30, 50, 100 e 1000 ativações) e SIPHT (30, 60, 100 e 1000 ativações) conforme apresentado na Tabela 3. Além disso, cada *workflow* foi escalonado em 5 conjuntos distintos de máquinas virtuais (Tabela 2). Em cada execução, foram gerados 100 episódios. Cada conjunto de máquina foi elaborado de modo que se obtenham 8, 16, 32, 64, e 128 CPUs virtualizadas.

Foram simulados dois tipos de máquinas dentro do *WorkflowSim4RL* seguindo o padrão da Amazon AWS, a *m3.medium* com 1 CPU virtualizada, e a *m3.2xlarge* com 8 CPUs virtualizadas. Em termos de memória RAM, a máquina *m3.medium* possui 3,75 GB e a máquina *m3.2xlarge* possui 30 GB. Um fator importante para o *WorkflowSim4RL* é o valor de micro instruções por segundo (MIPS). O parâmetro MIPS no *WorkflowSim4RL* é que de fato define a heterogeneidade do ambiente. Definimos o valor de MIPS para a máquina *m3.medium* como 1.000 e para a máquina *m3.2xlarge* 4.156 conforme definido pela Amazon².

Tabela 2. Conjuntos de máquinas

8 vCPUs	16 vCPUs	32 vCPUs	64 vCPUs	128 vCPUs
-	8 - <i>m3.medium</i>	8 - <i>m3.medium</i>	8 - <i>m3.medium</i>	8 - <i>m3.medium</i>
1 - <i>m3.2xlarge</i>	1 - <i>m3.2xlarge</i>	3 - <i>m3.2xlarge</i>	7 - <i>m3.2xlarge</i>	15 - <i>m3.2xlarge</i>

Em cada uma das execuções, o *Q-Learning* e a tabela com os valores de $Q(s, a)$ são inicializadas com o valor de 0, 2 e aplicamos um fator de aleatoriedade na escolha da máquina virtual a ser usada para a alocação das ativações. Desta forma, o escalonador escolhe a máquina virtual por meio de um fator probabilístico, ou o agente escolhe a máquina que possuir o maior valor da função de avaliação Q que se encontra na tabela presente no arquivo do episódio anterior, seguindo a abordagem ϵ -greedy. Além disso, o parâmetro de peso α foi definido na função P_i como 0,5, o valor de γ do algoritmo *Q-Learning* como 0,5 e o fator de velocidade de aprendizado l como igual a 1, 0. Os tempos de execução em segundos de cada um dos *workflows* no ambiente definido são apresentados na Tabela 3. Podemos verificar que as execuções com mais vCPUs apresentaram um tempo de execução reduzido (conforme esperado), porém, em muitos casos, o valor do *speedup* não foi alto, em especial para o *workflow* Epigenomics, que apresentou as menores reduções de tempo de execução ao se adicionar mais máquinas virtuais.

Além da avaliação do tempo de execução simulado, executamos um dos planos de escalonamento gerado pelo *WorkflowSim4RL* em um experimento real utilizando o SGWfC SciCumulus. O workflow escolhido foi o Montage com 25 ativações associadas. Comparamos o tempo de execução do workflow escalonado utilizando aprendizado por reforço com o escalonamento FCFS *First Come First Served*. Conforme apresentado na Figura 4, o escalonamento com aprendizado por reforço apresentou um tempo de execução menor que o FCFS em todas as combinações de máquinas virtuais analisadas. Apesar de novos experimentos mais complexos se mostrarem necessários, os resultados aqui apresentados mostram a viabilidade do *WorkflowSim4RL*.

5. Trabalhos Relacionados

Em [Barrett et al. 2011] foi proposta a utilização de um algoritmo genético para a evolução do aprendizado do agente, utilizando MDP para a escolha dos melhores planos de escalonamento

²www.cmips.net/2014/04/01/amazon-40-cut-and-cmips-benchmarks-with-amazon-m3-2xlarge-and-c3-4xlarge/

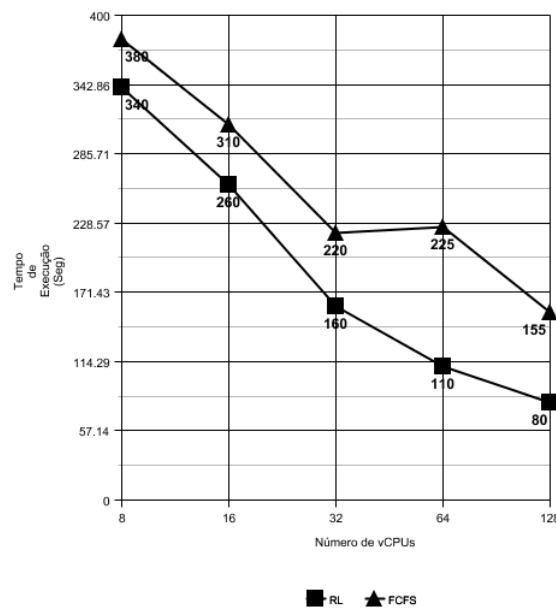


Figura 4. Tempo de execução do Montage_25 executado com o SciCumulus

para *Workflows* em conjunto com um modelo Bayesiano. Apesar de ter objetivo semelhante a abordagem proposta nesse artigo (*i.e.*, reduzir o tempo de execução do *Workflow*), a abordagem proposta por [Barrett et al. 2011] foca também em reduzir custos financeiros. Nossa abordagem também se diferencia por utilizar um escalonador que implementa o *Q-Learning* no simulador *WorkflowSim*, evitando que o tempo de aprendizado do agente seja contabilizado na execução do *workflow* de fato.

Apesar da abordagem proposta nesse artigo se inspirar no trabalho proposto por [Costa et al. 2009], utilizando a ideia de funções de desempenho por recurso e a definição do valor de recompensa/punição, a abordagem de [Costa et al. 2009] é focada no ambiente em Grades Computacionais. Além disso, nossa abordagem se difere na implementação do algoritmo de aprendizado por reforço *Q-Learning* juntamente com as métricas geradas pelas funções de desempenho propostas por [Costa et al. 2009].

6. Conclusão e Trabalhos Futuros

Neste artigo, apresentamos uma abordagem para escalonamento de *workflows* científicos em nuvens de computadores, a qual utilizou Aprendizado por Reforço como técnica para a descoberta de um escalonador das máquinas virtuais disponíveis para execução. A abordagem proposta é uma alternativa aos escalonadores existentes nos SGWfCs para nuvem, pois não necessita que características do ambiente sejam formalizadas de antemão. Ao mesmo tempo, a abordagem proposta é de simples implementação, necessitando somente ser capaz de avaliar se um determinado plano de escalonamento teve um bom resultado ou não. Nesse trabalho, estendemos o *WorkflowSim*, criando assim o *WorkflowSimARL* que implementa a abordagem proposta nesse artigo. Os resultados experimentais mostram que o simulador baseado em aprendizado por reforço é capaz de se beneficiar do ambiente heterogêneo, gerando escalonamentos com tempos menores a medida que a quantidade de recursos aumenta.

Além disso, realizamos uma execução real do *workflow* Montage com 25 ativações utilizando o plano de escalonamento da abordagem proposta e o gerado pelo FCFS

Tabela 3. Tempo de Execução dos Workflows no WorkflowSim4RL (em segundos)

Workflows	Número de Núcleos Virtuais			
	16	32	64	128
CyberShake_30	6.559,71	6.517,74	6.278,79	6.166,39
CyberShake_50	12.762,63	12.550,50	12.316,62	12.183,19
CyberShake_100	25.354,15	25.113,91	24.562,39	24.199,95
CyberShake_1000	47.807,61	44.118,30	40.629,30	38.257,61
Epigenomics_24	15.085,01	9.496,22	9.798,79	7.766,11
Epigenomics_46	33.232,34	31.314,53	22.698,94	17.552,66
Epigenomics_100	320.684,21	254.145,95	197.687,41	201.803,33
Epigenomics_997	3.006.902,18	2.246.804,08	1.707.749,73	1.422.339,80
Inspirar_30	6.176,08	4.454,18	4.586,26	3.374,25
Inspirar_50	9.720,62	8.972,35	7.286,11	6.228,70
Inspirar_100	16.078,31	12.785,12	10.975,93	10.184,21
Inspirar_1000	172.362,68	126.508,68	95.578,53	77.739,38
Montage_25	226,70	174,01	175,80	114,62
Montage_50	477,42	424,33	330,19	250,10
Montage_100	950,57	774,25	647,64	520,82
Montage_1000	9.938,84	7.834,66	6.347,65	5.264,15
Sipht_30	3.733,51	5.180,81	5.213,3	1.388,70
Sipht_60	9.643,12	7.425,38	6.878,84	2.968,70
Sipht_100	15.379,13	14.702,93	10.397,34	5.846,22
Sipht_1000	131.305,53	102.202,37	75.976,43	66.467,61

(padrão do *WorkflowSim*). Verificamos que a abordagem proposta apresentou um tempo de execução menor para todas as combinações de máquinas exploradas. Dessa forma, verificamos que em ambientes dinâmicos e heterogêneos como as nuvens de computadores, estratégias adaptativas como aprendizado por reforço são importantes, mas acreditamos serem necessários mais experimentos com *workflows* reais e em maior escala para analisarmos o impacto do escalonamento com aprendizado por reforço em uma variedade de perfis de *workflows*. Como trabalhos futuros, poderíamos intercalar execuções simuladas no *WorkflowSim4RL* com execuções reais no SciCumulus, de forma que a tabela do *Q-Learning* pudesse ser realimentada de acordo com os tempo reais, pois o tempo simulado pode ser ligeiramente diferente do tempo de execução real.

Referências

- Barrett, E., Howley, E., and Duggan, J. (2011). A learning architecture for scheduling workflow applications in the cloud. In *Web Services (ECOWS), 2011 Ninth IEEE European Conference on*, pages 83–90. IEEE.
- Chen, W. and Deelman, E. (2012). Workflowsim: A toolkit for simulating scientific workflows in distributed environments. In *E-science (e-science), 2012 IEEE 8th International Conference on*, pages 1–8. IEEE.
- Costa, B. F., Mattoso, M., and Dutra, I. (2009). Applying reinforcement learning to scheduling strategies in an actual grid environment. *International Journal of High Performance Systems Architecture*, 2(2):116–128.

- Deelman, E., Gannon, D., Shields, M., and Taylor, I. (2009). Workflows and e-science: An overview of workflow system features and capabilities. *Future Generation Computer Systems*, 25(5):528–540.
- Deelman, E., Mehta, G., Singh, G., Su, M.-H., and Vahi, K. (2007). Pegasus: mapping large-scale workflows to distributed resources. In *Workflows for e-Science*, pages 376–394. Springer.
- Mattoso, M., Werner, C., Travassos, G., Braganholo, V., Murta, L., Ogasawara, E., Oliveira, F., and Martinho, W. (2009). Desafios no apoio à composição de experimentos científicos em larga escala. *Seminário Integrado de Software e Hardware, SEMISH*, 9:36.
- Mattoso, M., Werner, C., Travassos, G. H., Braganholo, V., Ogasawara, E., Oliveira, D., Cruz, S., Martinho, W., and Murta, L. (2010). Towards supporting the life cycle of large scale scientific experiments. *International Journal of Business Process Integration and Management*, 5(1):79–92.
- Ogasawara, E., Dias, J., Oliveira, D., Porto, F., Valduriez, P., and Mattoso, M. (2011). An algebraic approach for data-centric scientific workflows. *Proc. of VLDB Endowment*, 4(12):1328–1339.
- Oliveira, D., Ogasawara, E., Ocaña, K., Baião, F., and Mattoso, M. (2012). An adaptive parallel execution strategy for cloud-based scientific workflows. *Concurrency and Computation: Practice and Experience*, 24(13):1531–1550.
- Sutton, R. S. and Barto, A. G. (1998). *Reinforcement learning: An introduction*, volume 1. MIT press Cambridge.
- Travassos, G. H. and Barros, M. O. (2003). Contributions of in virtuo and in silico experiments for the future of empirical studies in software engineering. In *2nd Workshop on Empirical Software Engineering the Future of Empirical Studies in Software Engineering*, pages 117–130.
- Vaquero, L. M., Rodero-Merino, L., Caceres, J., and Lindner, M. (2008). A break in the clouds: towards a cloud definition. *ACM SIGCOMM Computer Communication Review*, 39(1):50–55.
- Zhao, Y., Hategan, M., Clifford, B., Foster, I., Von Laszewski, G., Nefedova, V., Raicu, I., Stef-Praun, T., and Wilde, M. (2007). Swift: Fast, reliable, loosely coupled parallel computation. In *Services, 2007 IEEE Congress on*, pages 199–206. IEEE.

Análise de Desempenho das Tecnologias de Virtualização de Rede da Plataforma *OpenStack*

Roseli da R. Barbosa¹, Paulo A. L. Rego¹, Michel S. Bonfim¹ e Arthur de C. Callado¹

¹Redes de Computadores – Universidade Federal do Ceará (UFC)
Av. José de Freitas Queiroz, 5003 – 63902-580 – Quixadá – CE – Brasil

roselirocha@alu.ufc.br, {pauloalr, michel, arthur}@ufc.br

Abstract. *With the popularity of cloud computing, the need to support multiple tenants on logically independent networks has increased, fostering the advent of network virtualization. In order to evaluate the performance of network virtualization technologies, this paper uses the OpenStack platform to compare the performance of virtual networks using the VXLAN and GRE tunneling protocols with Linux Bridge and Open vSwitch. With the experiments, it was possible to verify that scenarios using Open vSwitch with VXLAN have obtained higher throughput rates between different compute nodes, while Linux Bridge with VXLAN has shown lower latency and jitter rates in most scenarios.*

Resumo. *Com a popularidade da computação em nuvem, surgiu a necessidade de ampliar o suporte a múltiplos inquilinos pertencentes a redes logicamente independentes, fomentando o advento da virtualização de redes. Com o intuito de avaliar o desempenho das tecnologias de virtualização de redes, este trabalho utilizou a plataforma OpenStack visando comparar o desempenho de redes virtuais utilizando os protocolos de tunelamento VXLAN e GRE, com Linux Bridge e Open vSwitch. Com os experimentos, foi possível constatar que cenários utilizando Open vSwitch com VXLAN obtiveram maiores taxas de vazão entre nós de computação diferentes. No entanto, Linux Bridge com VXLAN apresentou menores taxas de latência e jitter na maioria dos cenários.*

1. Introdução

A virtualização de redes vem sendo considerada uma excelente alternativa para os grandes centros de dados. Embora não seja um conceito totalmente novo, ainda se tem buscado aprimorar a sua aplicabilidade. Com a virtualização de redes, tornou-se possível fornecer uma plataforma onde múltiplas redes coexistem em uma mesma infraestrutura com características distintas (CARAPINHA; JIMÉNEZ, 2009).

Em ambientes de computação em nuvem, a virtualização de redes é essencial para suportar múltiplos inquilinos pertencentes a redes logicamente independentes. Nesse cenário, surgiu a necessidade de adequar o uso das LANs virtuais para lidar com as limitações no número de quadros de dados Ethernet disponíveis, quando é necessária a sua adoção nos grandes centros de dados. Vale ressaltar que infraestruturas de grandes centros de dados são compostas de várias máquinas virtuais e de muitos usuários, uma vez que o ambiente é multi-inquilino. Diante disso, é comum a comunicação entre máquinas virtuais pertencentes a diferentes servidores, necessitando a adesão de tecnologias que possibilitem o tráfego dos dados em uma rede pública, dando a ideia de conexão ponto a ponto.

Visando contornar as limitações das arquiteturas de redes de camada 2, surgiu o conceito de redes de virtualização por sobreposição. Na prática, isso consiste na sobreposição da camada 2 em uma rede de camada 3. Consequentemente, tornou-se possível a conectividade entre diferentes máquinas virtuais que podem estar localizadas em diferentes centros de dados ou em diferentes servidores de um mesmo centro de dados.

Dentre as diversas soluções de nuvem existentes, a plataforma *OpenStack* vem ganhando destaque nos últimos anos devido a sua alta escalabilidade, grande flexibilidade e por ser uma solução de código aberto (BARROS et al., 2015). O *OpenStack* utiliza uma variedade de tecnologias para criação de infraestruturas de redes virtualizadas (e.g., *Open vSwitch*, *Linux Bridge*, *VXLAN* e *GRE*), ficando o administrador da nuvem responsável por escolher a tecnologia apropriada para seu ambiente.

Diversos estudos com plataformas de nuvem vêm sendo realizados buscando aperfeiçoar as técnicas de virtualização de redes existentes, apontando a melhor solução para cada ambiente proposto. Nesse contexto, o objetivo deste trabalho é avaliar o desempenho das tecnologias de virtualização de rede disponíveis na plataforma *OpenStack*. Para tal, o presente estudo implantou uma nuvem privada com a versão Mitaka do *OpenStack* e realizou diferentes experimentos.

O restante deste trabalho está organizado da seguinte forma: na Seção 2, são apresentados conceitos fundamentais referentes à virtualização de redes, com foco na plataforma *OpenStack*; a Seção 3 apresenta metodologia e resultados da avaliação de desempenho; a Seção 4 discute os estudos relacionados a este trabalho; e por fim, Seção 5 apresenta as considerações finais e trabalhos futuros.

2. *OpenStack*

O *OpenStack* é uma plataforma de nuvem de código aberto que surgiu em 2010 através de uma iniciativa da *Rackspace Hosting*¹ e da NASA (LEDYAYEV; RICHTER, 2014). Sua estrutura era baseada na plataforma *Nebula*, da NASA, e no sistema de arquivos em nuvem da *Rackspace*. Ele foi escrito na linguagem de programação Python e atualmente implementa duas APIs de controle, a API EC2 e a *Rackspace*. Fornece ainda suporte a diferentes hipervisores (Xen, KVM, HyperV, Qemu).

Atualmente, o *OpenStack* tem uma das maiores comunidades de código aberto, que conta com o apoio de grandes organizações, como VMware, IBM, Cisco, Dell, Red Hat e Canonical, contribuindo para o seu desenvolvimento (SINGH; SINGH; PACHAURI, 2014). O *OpenStack* foi projetado para fornecer serviços em um ambiente de *IaaS* (do inglês *Infrastructure as a Service*) que tem por finalidade a implantação de nuvens públicas e privadas. Desde que se iniciou, seguiu um padrão de lançamentos a cada seis meses, fornecendo bom desempenho em suas funcionalidades.

Dentre os módulos fornecidos pelo *OpenStack* estão os do tipo Core, compostos pelo *Nova*, *Keystone*, *Neutron*, *Glance*, *Swift* e *Cinder*, além de outros módulos essenciais como o *Horizon*, *Ceilometer* e o *Heat*. Em uma infraestrutura organizacional, esses são os principais elementos que devem existir para a utilização do ambiente. Na Tabela 2.1 são definidas as principais características de cada módulo.

¹Rackspace: <https://www.rackspace.com>.

Tabela 2.1. Principais módulos do *OpenStack*

Nova (<i>Cloud Management</i>)	Gerencia o ciclo de vida das instâncias de máquinas virtuais.
Keystone (<i>Identity service</i>)	Valida as credenciais e informações dos usuários e grupos de usuários; Provê dados sobre projetos e domínios; Fornece dados sobre <i>roles</i> às entidades geridas pelos serviços de <i>Identity</i> e de <i>Resource</i> ; Valida e administra solicitações de autenticação; Fornece o catálogo de todos os serviços, e por fim fornece um mecanismo de autorização baseado em regras.
Glance (<i>Image Service</i>)	Armazena e recupera imagens de disco das máquinas virtuais, através de uma API RESTful.
Swift (<i>Object Storage</i>)	Armazena e recupera objetos de dados não estruturados por meio de uma API RESTful.
Cinder (<i>Block Storage</i>)	Prover um <i>block storage</i> persistente para as instâncias em execução, garantindo que continue existindo mesmo após uma máquina virtual seja apagada.
Horizon (<i>Control Panel/Dashboard</i>)	Interface gráfica para todos os módulos, dinamizando o gerenciamento dos recursos da nuvem.
Ceilometer (<i>Accounting</i>)	Fornece o monitoramento e medição dos recursos da nuvem, facilitando tomadas de decisões de prevenção e correção quando ocorrer possíveis falhas.
Heat (<i>Orchestration</i>)	Mecanismo que permite uma automatização da implantação da infraestrutura, visto que gerencia o ciclo de vida do ambiente.

A próxima seção irá descrever detalhadamente o funcionamento do módulo *Neutron*, também considerado o módulo de rede do *OpenStack*. O *Neutron* é responsável por todos os aspectos para a criação de infraestruturas de redes virtuais, por esse motivo é composto de diversos componentes que são descritos a seguir.

2.1. *OpenStack Neutron*

O módulo de rede do *OpenStack* passou por grandes mudanças desde sua idealização, visando se adequar à inclusão de novos protocolos de roteamento e a novas regras de segurança (GEBREYOHANNES, 2014). A proposta do módulo *Neutron* é prover rede como serviço, permitindo a criação de infraestruturas de redes dinâmicas. Por meio de uma API, os demais módulos do *OpenStack* utilizam o *Neutron* para fornecer conectividade entre as VMs.

Em redes *OpenStack*, é essencial entender a variedade de tecnologias que podem ser utilizadas. O *plugin ML2* (*Modular Layer 2*) permite definir qual tecnologia será configurada para cada instalação, seja para implementar serviços de camada 2 ou até mesmo fornecer serviços de redes adicionais, como balanceamento de carga, *firewall* e VPN.

A arquitetura do *plugin ML2* é definida com base no *driver* que será gerenciado para cada tipo de rede, e o mecanismo que será aplicado para acessar a rede. A Tabela 2.2 relaciona os *drivers* suportados por cada mecanismo.

- **Open vSwitch:** *Switch* virtual *multi-layer*, que pode operar nas camadas 2, 3 e 4.
- **Linux Bridge:** Agente de camada 2 para gerenciar *bridges* em cada nó computação, ou em qualquer nó que forneça serviços de camada 3.
- **SRIOV:** Funcionalidade de virtualização e compartilhamento, que permite virtualizar controladores PCIe Ethernet.
- **MacVTap:** Agente que usa os dispositivos MacVTap do kernel para implementar redes L2. Além disso, ele é usado em conjunto com o *Open vSwitch* ou *Linux Bridge* para fornecer serviços de camada 3.

Tabela 2.2. Drivers e mecanismos de acesso

<i>Drivers / Mecanismos</i>	Flat	VLAN	VXLAN	GRE
<i>Open vSwitch</i>	Sim	Sim	Sim	Sim
<i>Linux bridge</i>	Sim	Sim	Sim	Não
SRIOV	Sim	Sim	Não	Não
MacVTap	Sim	Sim	Não	Não
<i>L2 population</i>	Não	Não	Sim	Sim

- ***L2 population***: Mecanismo com a função de otimizar o tráfego nas redes de sobreposição VXLAN e GRE, por preencher as tabelas de encaminhamento dos *switches* virtuais, diminuindo o tráfego em broadcast.

Em redes utilizando o *driver* Flat, todos os inquilinos compartilham o mesmo segmento de rede. Por outro lado, quando utilizado o tipo de rede VLAN, todos os inquilinos são separados logicamente conforme a necessidade. Os protocolos VXLAN e GRE também implementam separações lógicas, mas são utilizados para a criação de redes virtualizadas por sobreposição.

Todos os mecanismos de acesso listados na Tabela 2.2 permitem a criação de redes L2 utilizando os recursos de rede do *OpenStack*, porém o *Open vSwitch* e o *Linux Bridge* suportam ainda mais recursos de rede e por esse motivo são ainda mais aplicados.

Como a solução de nuvem escolhida para este trabalho é o *OpenStack*, as próximas subseções abordam em detalhes os protocolos de virtualização de redes suportados pela plataforma e que estão diretamente relacionados à proposta de avaliação deste estudo.

2.2. VXLAN

O protocolo VXLAN (*Virtual Extensible Local Area Network*) buscou resolver o problema de limitação da quantidade de VLANs suportadas em um mesmo *switch*. Como as VLANs possuem um campo de identificação de 12 bits, é possível criar até 4094 redes virtuais. Embora seja um valor consideravelmente alto, não estava satisfazendo a demanda dos grandes centros de dados. Partindo dessa premissa, surgiu a padronização do protocolo VXLAN, que utiliza um identificador com 24 bits, o que equivale a 16 milhões de redes (MAHALINGAM et al., 2014).

Em VXLAN, cada rede sobreposta é chamada de segmento VXLAN, garantindo a comunicação apenas entre VMs que pertencem ao mesmo segmento. Para cada segmento, é concedido um ID único, chamado de identificador de rede VXLAN (VNI, do inglês *VXLAN Network Identifier*). Logo, para identificar uma máquina, é realizada uma associação do endereço MAC com o VNI. Em resumo, quando um pacote é enviado, é atribuído um cabeçalho VXLAN com um identificador relacionado. Em seguida, é adicionado o MAC e encapsulado o quadro pela extremidade do túnel, chamado VTEP (do inglês *VXLAN Tunnel Endpoint*) (SENGÉS; ALVARENGA; MOREIRA, 2017).

Para que haja comunicação entre as VMs, é preciso verificar os segmentos de origem e destino através do VTEP, que também tem o papel de certificar se existe algum mapeamento do endereço MAC. Após o envio, é feita uma validação do VNI pelo VTEP de destino, que irá identificar se existe alguma VM que possui o VNI e MAC associados.

2.3. NVGRE

O protocolo de encapsulamento de roteamento genérico (GRE, do inglês *Generic Routing Encapsulation*) foi desenvolvido pela Cisco com a finalidade de encapsular múltiplos protocolos da camada de rede, criando conexões ponto a ponto. O GRE funciona da seguinte forma: o cabeçalho GRE e o cabeçalho IP são adicionados ao pacote e encaminhados para o destino. Ao chegar no destino, ocorre o desencapsulamento, retornando ao estado original do pacote (FARINACCI et al., 1994).

Com os bons resultados no desempenho do protocolo GRE, foi idealizado um novo protocolo para virtualização de redes usando encapsulamento de roteamento genérico (NVGRE, do inglês *Network Virtualization Using Generic Routing Encapsulation*). Assim como o padrão VXLAN, o protocolo NVGRE surgiu com o objetivo de atender às deficiências dos modelos arquiteturais das redes virtuais. Ele foi especificado pela RFC 7637 e desenvolvido em setembro de 2011 pelas empresas Microsoft, Arista Networks, Intel, Dell, Hewlett-Packard, Broadcom e Emulex.

Semelhante ao protocolo VXLAN, o NVGRE possui um identificador de 24 bits para cada sub-rede (VSID, do inglês *Virtual Stain Identifier*) que permite a identificação única de uma rede virtual. Com o uso do VSID, foi adaptado o cabeçalho original do GRE para que incorporasse esse identificador. Portanto, o cabeçalho do NVGRE consiste no cabeçalho original do protocolo GRE com um VSID vinculado (SENGÉS; ALVARENGA; MOREIRA, 2017).

O cabeçalho é organizado em duas extremidades: a primeira corresponde à extremidade externa, que consiste no cabeçalho Ethernet e nos prefixos de origem e destino. Esta extremidade possui ainda o cabeçalho do protocolo GRE e o cabeçalho IP com o endereço de destino correspondente, além de uma chave de 24 bits com mais 8 bits referente a um identificador de fluxo virtual (FlowID). Já a extremidade interna, embora também possua o cabeçalho Ethernet com os prefixos de origem e destino, nela esses são referentes às interfaces virtuais (SRIDHARAN et al., 2011).

3. Análise de Desempenho

Para avaliar as tecnologias de virtualização de redes do *OpenStack*, uma nuvem privada foi implantada com a versão Mitaka do *OpenStack* no Laboratório de Sistemas e Banco de Dados da Universidade Federal do Ceará. A instalação se deu de forma manual, seguindo o tutorial oficial², com o objetivo de incorporar todas as funcionalidades fornecidas pelos seus módulos. Para tal, foram utilizadas duas máquinas com sistema operacional Ubuntu Server 14.04 LTS de 64 bits com as seguintes configurações: 32 GB de memória RAM, 100 GB de disco e 2 processadores com 12 núcleos cada.

Para a implantação, foram utilizados dois nós. O *host01* implementou a função de *controller*, executando o módulo *Keystone* para fornecer autenticação, autorização, catálogo de serviços e APIs para acesso às funcionalidades. O *Glance* foi usado para armazenar e recuperar as imagens de máquinas virtuais. O *Neutron* e o *Nova* foram configurados no *host01* e no *host02*, onde parte do *Neutron* estava no *host01* para configurar e acomodar os equipamentos de rede e outra parte no *host02* com alguns agentes de rede.

²Guia de instalação do OpenStack: <https://docs.openstack.org/mitaka/install-guide-ubuntu>.

O *Nova* também foi configurado no *host01* e no *host02*, mas como nó computação para que fosse possível instanciar VMs. Além disso, o *host01* estava executando o *Horizon* para fornecer uma interface gráfica de acesso aos módulos. Uma ilustração da nuvem implantada pode ser vista na Figura 1, mostrando em detalhes a organização e a disposição de cada módulo implantado.

Foram criadas duas redes virtuais, uma do tipo *provider*, para acesso à rede externa, e uma do tipo *self-service*, como rede privada. Um roteador foi adicionado para fazer a conexão entre as redes externa e interna, e um *floating IP* foi atribuído a uma instância em cada rede privada criada, servindo como um NAT para acessar a instância externamente. A quantidade de instâncias variou entre 2, 4 e 8, onde elas foram alocadas no mesmo nó computação e em nós de computação diferentes (e.g., 1 instância em cada nó, 2 instâncias em cada nó ou 4 instâncias em cada nó).

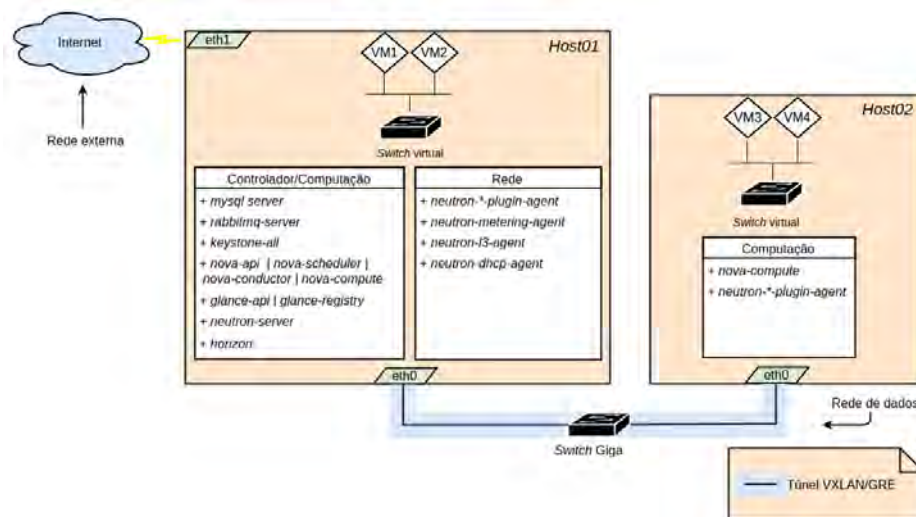


Figura 1. Nuvem OpenStack implantada

3.1. Experimentos

Três ambientes foram configurados, alternando o mecanismo de acesso e o protocolo de virtualização de redes utilizados: (i) *Open vSwitch* com VXLAN, (ii) *Open vSwitch* com GRE, e (iii) *Linux Bridge* com VXLAN. De acordo com a documentação do *OpenStack*, o agente *Linux Bridge* não suporta redes com o GRE, e por isso cenários com tais tecnologias não foram utilizados. As configurações necessárias para cada implantação estão disponíveis no repositório do GitHub³.

Para cada ambiente criado, as seguintes métricas foram avaliadas: (i) vazão TCP e UDP, (ii) latência e jitter, (iii) taxa de perda de pacotes. Tratando-se dos fatores e níveis, consideramos o número de VMs (2, 4 e 8), a disposição ou alocação das VMs (mesmo nó computação e em nós de computação diferentes), mecanismo de comunicação (*Open vSwitch* ou *Linux Bridge*), tecnologia de virtualização (VXLAN ou GRE), tipo do tráfego (TCP ou UDP) e a quantidade de execuções (30 execuções unidirecionais e bidirecionais).

³Repositório: <https://github.com/rose36/RedesVirtuaisOpenstack>.

A criação de cada cenário foi feita de maneira automatizada usando um *script* desenvolvido na linguagem Shell Script, que pode ser executado para a criação de redes, roteadores, adicionar interfaces aos roteadores e para a criação de instâncias. Outro *script* foi criado para avaliar o ambiente por meio da ferramenta de geração e medição de tráfego Uperf⁴. Algumas métricas foram avaliadas através da execução de um ping durante o andamento do experimento, capturando o valor da latência e o jitter, além da taxa de perda de pacotes. Foi considerada ainda a vazão após a execução de cada experimento, coletada diretamente da saída do Uperf.

Para executar o *script*, é preciso passar por parâmetro a quantidade de vezes que será realizado o experimento, se será unidirecional ou bidirecional, o IP do *host* remoto, o protocolo que será utilizado, o tamanho do buffer, a duração de cada transação em segundos e o tamanho da mensagem. Após testar diferentes valores para os parâmetros a fim de maximizar a vazão, os seguintes valores foram fixados para todos os experimentos: (i) 30 repetições, (ii) 10 *threads* ou processos de execução, (iii) 256KB de tamanho do buffer, (iv) 30 segundos de duração, (v) 56KB o tamanho da mensagem. Os parâmetros que sofreram variações foram: o tipo de execução (unidirecional ou bidirecional), o IP do *host* remoto e o protocolo que seria utilizado (TCP ou UDP).

3.2. Resultados

Para cada avaliação realizada, buscamos determinar a vazão, latência, jitter e taxa de perda de pacotes em fluxos TCP e UDP considerando a quantidade de conexões simultâneas e a alocação das máquinas virtuais. Outro fator analisado foi o desempenho apresentado entre as técnicas de virtualização em execuções unidirecionais e bidirecionais (média apresentada em ambas direções). Além disso, avaliou-se o comportamento do tráfego através da variação da localização das instâncias, visando comparar o desempenho apresentado quando as instâncias são alocadas no mesmo nó computação e entre diferentes nós de computação.

As subseções a seguir exibem os gráficos gerados para a análise comparativa dos resultados. Em experimentos realizados entre instâncias do mesmo nó computação, as taxas de vazão estão em Gigabits por segundo (Gb/s). Já entre nós de computação diferentes, estão em Megabits por segundo (Mb/s). Para criar os gráficos, a média foi calculada de acordo com o número de amostras e fluxos gerados, sendo 30 amostras com 2 VMs (uma conexão), 60 amostras com 4 VMs (duas conexões) e 120 amostras com 8 VMs (quatro conexões). Além disso, foi calculado também o intervalo de confiança utilizando um nível de confiança de 95%.

Nas subseções 3.2.1, 3.2.2 e 3.2.3 são apresentados todos os resultados de conexões utilizando o protocolo TCP. Já para as conexões utilizando o UDP, apenas os resultados dos experimentos entre instâncias alocadas no mesmo nó computação são apresentados, pois foram detectados alguns problemas durante os experimentos utilizando UDP entre nós de computação diferentes. Tais problemas são retratados na Seção 3.2.4, onde são detalhadas as dificuldades encontradas, as medidas tomadas e os resultados obtidos.

⁴Uperf: <http://uperf.org>.

3.2.1. Vazão TCP e UDP

Em todos os experimentos entre instâncias alocadas no mesmo nó computação, as taxas de vazão em conexões UDP foram maiores quando comparadas com TCP. Os resultados obtidos foram ainda mais significativos em experimentos bidirecionais.

Os gráficos da Figura 2 mostram um comparativo entre as médias da vazão do TCP e do UDP em conexões unidirecionais e bidirecionais, levando em consideração a variação na quantidade de conexões simultâneas (isto é, o valor é por conexão). É possível observar que em conexões unidirecionais a única tecnologia que apresentou maior diferença na média entre fluxos TCP e o UDP foi *Open vSwitch* com VXLAN quando utilizou 8 instâncias. A média do UDP obtida foi em torno de 20% a mais quando comparada à média do TCP. Além disso, em todos os casos, os intervalos de confiança ficaram muito próximos à média.

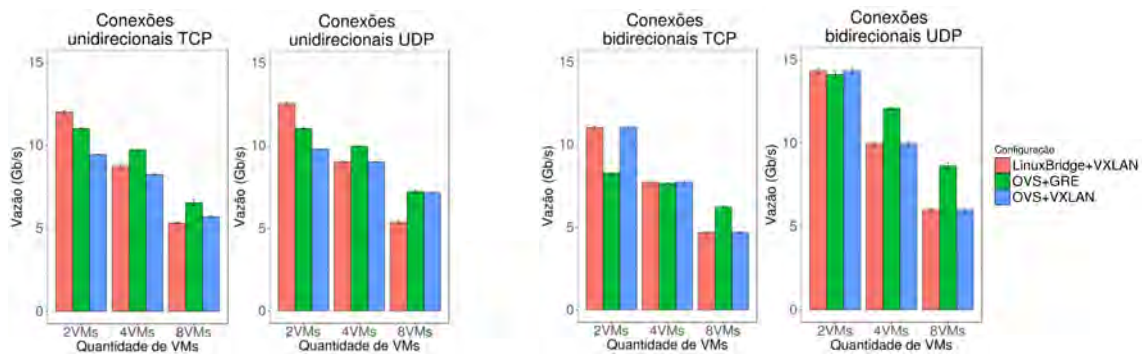


Figura 2. Vazão em Gb/s (mesmo nó computação)

Em experimentos bidirecionais, a maior diferença entre as médias TCP e UDP também foi ocasionada utilizando *Open vSwitch* com VXLAN. Diferente dos experimentos de conexões unidirecionais, os maiores resultados foram em testes com 2 instâncias, onde a diferença foi de 43%. A Figura 3 apresenta o comportamento do tráfego apenas com fluxos TCP entre nós diferentes. É possível observar que não houve diferenças significativas entre conexões unidirecionais e bidirecionais, e os resultados atingidos foram semelhantes.

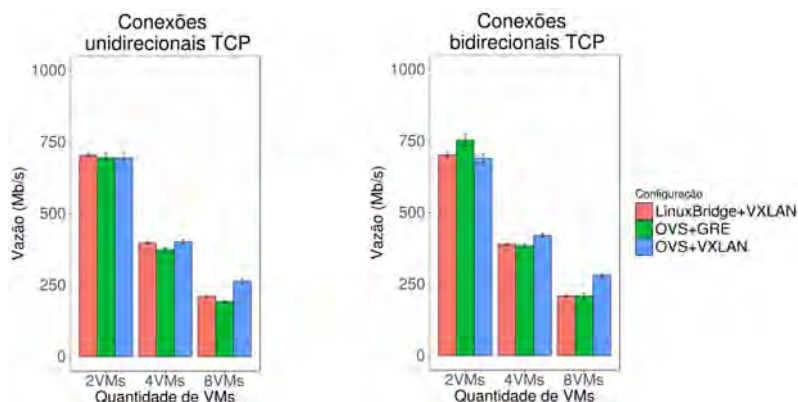


Figura 3. Vazão em Mb/s (diferentes nós de computação)

Outro fator importante de ser destacado nas Figuras 2 e 3 é que o uso do *Open vSwitch* com VXLAN e GRE obteve maiores taxas de vazão na maioria dos resultados.

Além disso, cenários com tais tecnologias apresentaram um melhor desempenho quando utilizados com um maior número de instâncias.

3.2.2. Taxa de Perda de Pacotes UDP

As maiores taxas de perda de pacotes no mesmo nó computação ocorreram nos experimentos unidirecionais, onde cenários utilizando *Linux Bridge* com VXLAN obtiveram maiores taxas. Os resultados mais significativos foram em experimentos com 8 instâncias, tendo aproximadamente 6% de pacotes perdidos. Tal valor representa 93% a mais de perda em experimentos usando *Open vSwitch* com VXLAN, e 84% a mais se comparado a *Open vSwitch* com GRE. Em experimentos bidirecionais, não houve perdas de pacotes.

Os gráficos da Figura 4 representam um comparativo entre as taxas de perda de pacotes em experimentos unidirecionais e bidirecionais, onde os resultados foram agregados com base na quantidade de instâncias e na tecnologia utilizada.

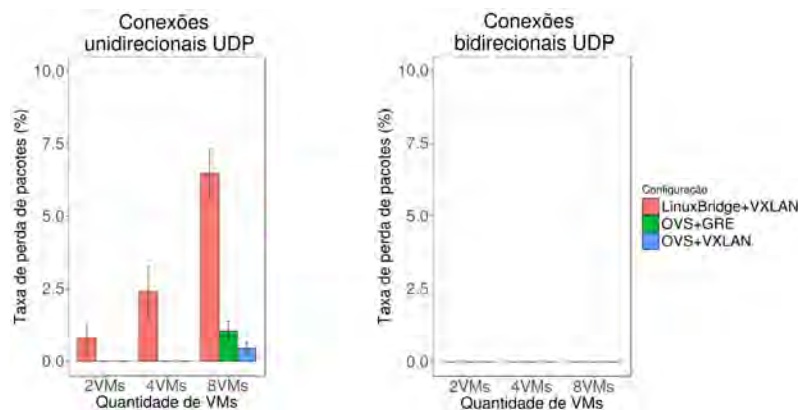


Figura 4. Taxa de perda de pacotes (mesmo nó computação)

3.2.3. Latência e Jitter TCP e UDP

Um comparativo dos resultados da latência e o jitter são apresentados nas Figuras 5, 6, 7 e 8. É possível observar nos gráficos que cenários utilizando *Linux Bridge* com VXLAN obtiveram menores taxas de latência em conexões unidirecionais, sendo superior às demais configurações apenas quando 4 e 8 instâncias são utilizadas em diferentes nós com tráfego TCP, e com 4 e 8 instâncias no mesmo nó com fluxos UDP.

Em conexões bidirecionais usando tráfego TCP, foram alcançados resultados equivalentes aos de conexões unidirecionais. A única diferença identificada foi em experimentos com 8 instâncias no mesmo nó computação. Já para as conexões bidirecionais UDP, as menores taxas de latência foram alcançadas com *Linux Bridge* e VXLAN (com 2 e 4 VMs).

Com relação ao jitter, na maioria dos resultados as menores médias foram obtidas utilizando *Open vSwitch* com GRE e VXLAN, tanto em conexões unidirecionais como bidirecionais.

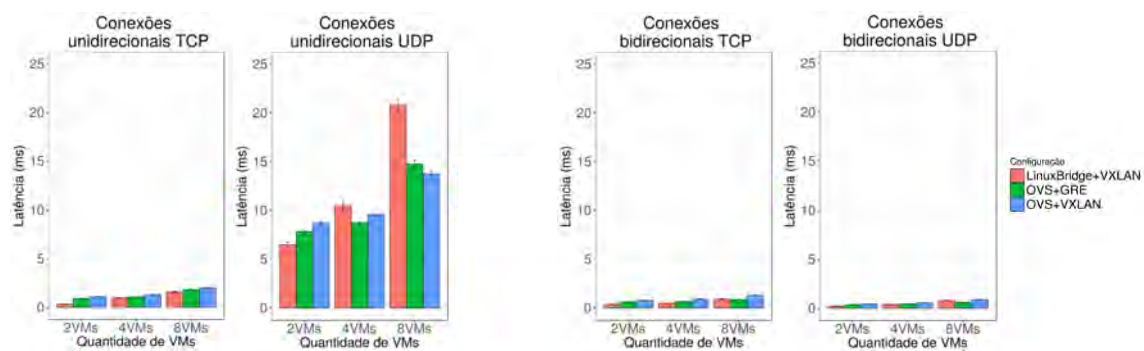


Figura 5. Latência (mesmo nó computação)

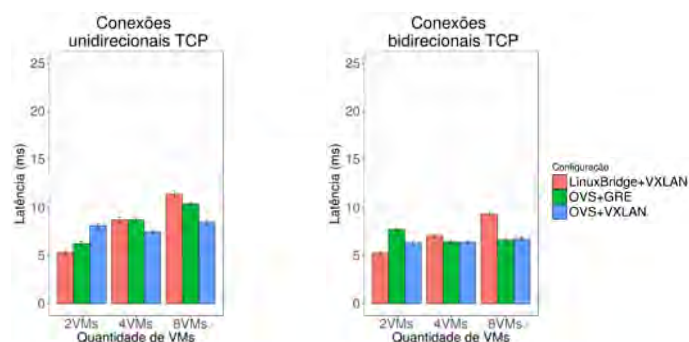


Figura 6. Latência em conexões TCP (diferentes nós de computação)

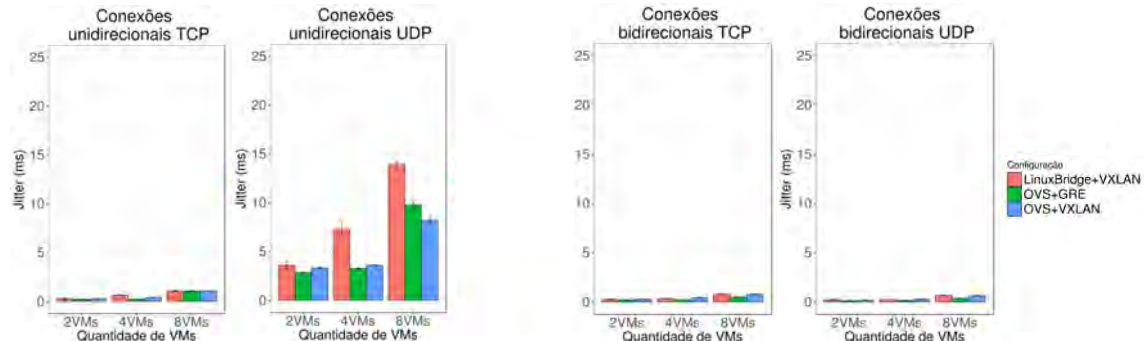


Figura 7. Jitter (mesmo nó computação)

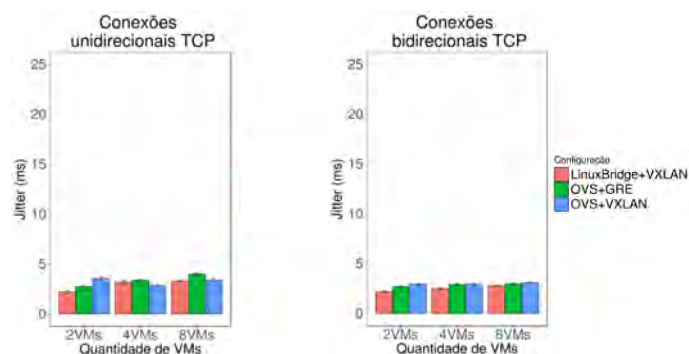


Figura 8. Jitter em conexões TCP (diferentes nós de computação)

3.2.4. UDP entre nós de computação diferentes

A ideia inicial era realizar o mesmo número e os mesmos tipos de experimentos para cada configuração, levando em consideração a alocação das VMs. No entanto, ao realizar experimentos gerando fluxos UDP entre nós de computação diferentes, foi identificada uma instabilidade na rede, a ponto de impedir a comunicação entre as instâncias, o que inviabilizou a execução dos experimentos. Ao avaliar o problema ocorrido, constatou-se que o experimento com UDP estava gerando tráfego como um ataque de negação de serviço, *UDP Flood Attack*, que é caracterizado pelo envio de um grande número de pacotes UDP para portas aleatórias de um determinado *host* remoto.

Um novo experimento foi realizado para investigar se o problema afetava também redes virtuais de diferentes clientes do *OpenStack*, que deveriam ser isoladas. O novo cenário possuía duas redes privadas com instâncias alocadas entre nós de computação diferentes. Constatou-se que o problema persistiu, mesmo utilizando apenas uma conexão para cada rede privada. Quando um dos clientes gera muito tráfego UDP, é possível congestionar e criar instabilidade nos nós de computação envolvidos na comunicação, a ponto de afetar outros clientes.

Diante do problema detectado, outro experimento foi planejado com UDP, mas usando a ferramenta *Iperf*⁵, que permite controlar a taxa de geração de tráfego. Foram avaliadas apenas conexões unidirecionais, com taxas de transferências de 100, 150 e 200 Mb/s. Para a escolha desses valores, experimentos com três conexões foram realizados a fim de maximizar a taxa de vazão. Com os experimentos, foi constatado que acima de 200 Mb/s já apresentava instabilidade entre as conexões.

A Figura 9 exibe os gráficos com a taxa de perdas de pacotes em conexões unidirecionais UDP entre nós de computação diferentes. Em todos os cenários avaliados, a configuração *Linux Bridge* com *VXLAN* obteve menores índices de perda de pacotes. Já as Figuras 10 e 11 contêm a latência e o jitter dos experimentos com a vazão controlada. É possível constatar que os testes usando *Linux Bridge* com *VXLAN* obtiveram menores taxas.

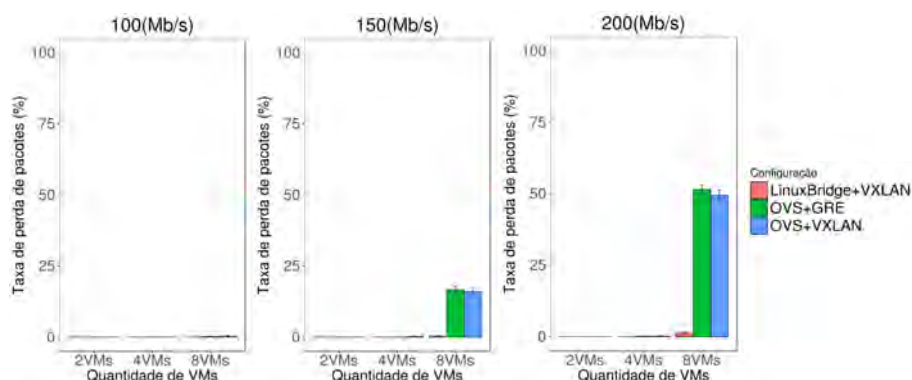


Figura 9. Taxa de perdas de pacotes em conexões unidirecionais UDP (diferentes nós de computação)

A Tabela 3.1 apresenta um resumo dos resultados dos experimentos, visando

⁵Iperf: <https://iperf.fr>.

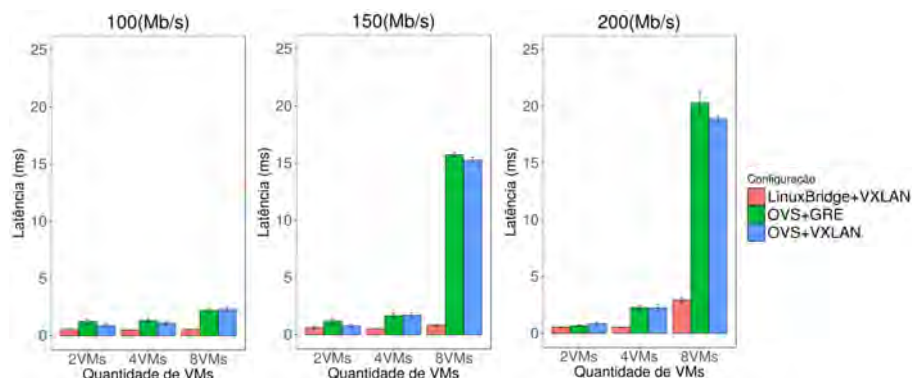


Figura 10. Latência em conexões unidirecionais UDP (diferentes nós de computação)

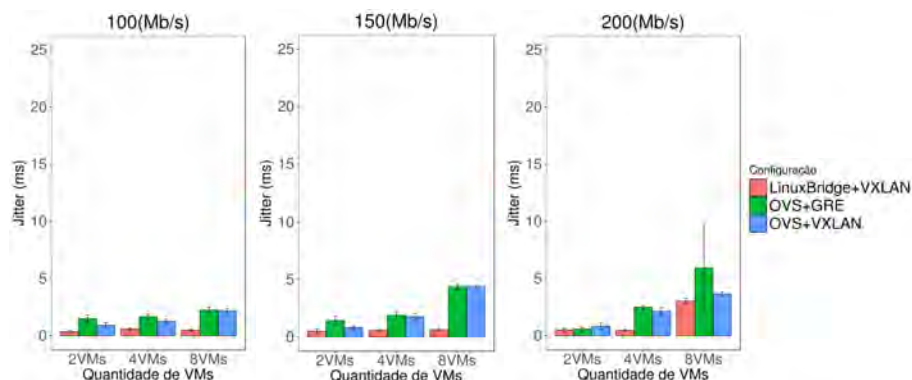


Figura 11. Jitter em conexões unidirecionais UDP (diferentes nós de computação)

demonstrar as tecnologias que obtiveram melhor desempenho para cada métrica avaliada.

4. Trabalhos Relacionados

No trabalho de Callegati et al. (2014), foi abordado o desempenho da virtualização de redes em ambientes com múltiplos inquilinos. Para avaliar o ambiente, foi implantada uma arquitetura de rede baseada em NFV (do inglês *Network Functions Virtualization*) considerando uma inspeção profunda dos pacotes (DPI, do inglês *Deep Packet Inspection*).

Em (KAWASHIMA; MATSUO, 2013), os autores apresentaram um novo modelo de sobreposição que não utiliza os protocolos de encapsulamento habitualmente aplicados em infraestruturas de nuvem. A solução apresentada pelos autores partiu de dois métodos principais. O primeiro método consiste na reescrita do cabeçalho do quadro em *switches* OpenFlow para evitar a fragmentação e problemas de aprendizagem do MAC. Já no segundo método, é utilizado o identificador de VLANs baseado no *host*, permitindo a escalabilidade do número de redes virtuais.

Já Kawashima e Matsuo (2014) se basearam em novas perspectivas para aprimorar os estudos já consumados em (KAWASHIMA; MATSUO, 2013). O novo ambiente para a execução dos experimentos passaram de 1GbE para 40GbE, onde foram avaliados novos tamanhos de pacotes e os efeitos da segmentação.

Para avaliação do modelo proposto em (KAWASHIMA; MATSUO, 2013) e adap-

Tabela 3.1. Comparativo dos resultados

Resultados	Mesmo nó computação			Nós de computação diferentes		
	2 VMs	4 VMs	8 VMs	2 VMs	4 VMs	8 VMs
Menor latência	<i>Linux Bridge</i> VXLAN	<i>Linux Bridge</i> VXLAN	<i>Open vSwitch</i> GRE	<i>Linux Bridge</i> VXLAN	<i>Open vSwitch</i> GRE	<i>Open vSwitch</i> GRE
Menor jitter	<i>Open vSwitch</i> GRE	<i>Open vSwitch</i> GRE	<i>Open vSwitch</i> GRE	<i>Linux Bridge</i> VXLAN	<i>Linux Bridge</i> VXLAN	<i>Open vSwitch</i> VXLAN
Menor perda	<i>Open vSwitch</i> VXLAN	<i>Open vSwitch</i> VXLAN	<i>Open vSwitch</i> VXLAN	<i>Linux Bridge</i> VXLAN	<i>Linux Bridge</i> VXLAN	<i>Linux Bridge</i> VXLAN
Maior vazão TCP	<i>Linux Bridge</i> VXLAN	<i>Open vSwitch</i> GRE	<i>Open vSwitch</i> GRE	<i>Open vSwitch</i> GRE	<i>Open vSwitch</i> VXLAN	<i>Open vSwitch</i> VXLAN
Maior vazão UDP	<i>Linux Bridge</i> VXLAN	<i>Open vSwitch</i> GRE	<i>Open vSwitch</i> GRE	N.A.	N.A.	N.A.

Nota - As células que apresentam N.A. representa os cenários onde a taxa de vazão foi controlada.

tado logo seguinte em (KAWASHIMA; MATSUO, 2014), foi utilizada a ferramenta Iperf para geração de fluxos TCP e UDP, comparando os resultados obtidos com as técnicas de virtualização de rede GRE e VXLAN. Assim como no nosso trabalho, os autores utilizaram o *Open vSwitch*, mas não utilizaram *Linux Bridge* nem deixaram claro no texto qual a solução de nuvem utilizada.

Gebreyohannes (2014) apresentou uma análise de desempenho em redes *OpenStack*, na qual avaliou fluxos de conexão TCP e UDP abordando o desempenho conforme o tipo de alocação das instâncias. Diferente deste trabalho, os autores utilizaram a ferramenta Iperf para geração e medição do tráfego, e criaram cenários apenas com GRE.

5. Conclusão e Trabalhos Futuros

Este trabalho abordou a importância da virtualização para a computação em nuvem, com foco nas redes de virtualização por sobreposição. Neste trabalho, foi implantada uma nuvem privada usando a plataforma *OpenStack*, e foram desenvolvidos *scripts* para automatizar a criação de cenários de experimentação e a realização de um grande número de testes, considerando diferentes métricas, fatores e níveis. Com os experimentos, foi possível comparar o desempenho das tecnologias suportadas pelo *OpenStack* (*Open vSwitch*, *Linux Bridge*, VXLAN e GRE), visando avaliar o uso de redes virtuais na plataforma.

Como pode-se observar, os cenários utilizando a configuração *Open vSwitch* com GRE obtiveram maiores taxas de vazão em experimentos executados no mesmo nó computação. No entanto, o uso de *Open vSwitch* com VXLAN foi melhor entre nós de computação diferentes. Dessa forma, estas tecnologias são recomendadas para ambientes que necessitem de maiores taxas de vazão para um bom funcionamento.

Em contrapartida, em relação às taxas de latência e jitter, o uso de *Linux Bridge* com VXLAN apresentou menores taxas nos experimentos entre nós de computação diferentes. Por outro lado, ao se verificar as perdas de pacotes, o mesmo atingiu maiores índices em experimentos no mesmo nó computação, enquanto o *Open vSwitch* atingiu maiores índices de perdas com VXLAN e GRE entre nós de computação diferentes.

Uma nova avaliação com mais nós de computação e instâncias é sugerida como trabalhos futuros, mensurando o grau de instabilidade da rede para ambientes multi-

inquilinos. Além disso, é importante investigar o problema do tráfego UDP entre nós de computação diferentes em diversos contextos, buscando mecanismos de detecção e prevenção.

Os *scripts* desenvolvidos também são uma contribuição deste trabalho e estão disponíveis para a comunidade, conforme Seção 3.1. Eles podem ser utilizados para automatizar a criação de cenários e a realização de experimentos, a fim de apresentar um relatório que pode auxiliar o administrador da nuvem a escolher as tecnologias de virtualização de redes a serem utilizadas no seu ambiente. Além disso, os *scripts* podem ser estendidos para utilizar outros módulos do *OpenStack* e realizar diferentes configurações no sistema operacional das máquinas físicas e virtuais, bem como nos mecanismos e *drivers* do *OpenStack*.

Referências

- BARROS, S. et al. Applying software-defined networks to cloud computing. In: *33rd Brazilian Symposium on Computer Networks and Distributed Systems*. [S.l.: s.n.], 2015.
- CALLEGATI, F. et al. Performance of multi-tenant virtual networks in openstack-based cloud infrastructures. In: IEEE. *Globecom Workshops, 2014*. [S.l.], 2014. p. 81–85.
- CARAPINHA, J.; JIMÉNEZ, J. Network virtualization: a view from the bottom. In: ACM. *Proceedings of the 1st ACM workshop on Virtualized infrastructure systems and architectures*. [S.l.], 2009. p. 73–80.
- FARINACCI, D. et al. Generic routing encapsulation (gre). 1994.
- GEBREYOHANNES, M. B. *Network Performance study on OpenStack Cloud Computing*. Dissertação (Mestrado), 2014.
- KAWASHIMA, R.; MATSUO, H. Non-tunneling edge-overlay model using openflow for cloud datacenter networks. In: IEEE. *Cloud Computing Technology and Science (CloudCom), 2013 IEEE 5th International Conference on*. [S.l.], 2013. v. 2, p. 176–181.
- KAWASHIMA, R.; MATSUO, H. Performance evaluation of non-tunneling edge-overlay model on 40gbe environment. In: IEEE. *Network Cloud Computing and Applications (NCCA), 2014 IEEE 3rd Symposium on*. [S.l.], 2014. p. 68–74.
- LEDYAYEV, R.; RICHTER, H. High performance computing in a cloud using openstack. *CLOUD COMPUTING*, p. 108–113, 2014.
- MAHALINGAM, M. et al. *Virtual extensible local area network (VXLAN): A framework for overlaying virtualized layer 2 networks over layer 3 networks*. [S.l.], 2014.
- SENGÉS, G.; ALVARENGA, I.; MOREIRA, J. *Redes de Virtualização por Sobreposição: VXLAN, NVGRE e STT*. 2017. Disponível em: https://www.gta.ufjf.br/ensino/ee1879/trabalhos_vf_2012_2/nvgre/. Acesso em: 15/06/2017.
- SINGH, P.; SINGH, V. P.; PACHAURI, G. Critical analysis of cloud computing using openstack. *International Journal of Computer Science and Mobile Computing*, v. 3, n. 3, 2014.
- SRIDHARAN, M. et al. Nvgre: Network virtualization using generic routing encapsulation. *IETF draft*, 2011.

Seshat: uma arquitetura de monitoração escalável para ambientes em nuvem

Vinicius S. Conceição¹, Nestor D. O. Volpini^{1,2}, Dorgival Guedes¹

¹Departamento de Ciência da Computação
Universidade Federal de Minas Gerais (UFMG) – Belo Horizonte, MG

{vconceicao, nestor, dorgival}@dcc.ufmg.br

²Centro Federal de Educação Tecnológica
de Minas Gerais (CEFET-MG) – Divinópolis, MG

nestor@div.cefetmg.br

Resumo. *A monitoração de sistemas computacionais cumpre papel essencial para gerir, viabilizar a manutenção e oferecer feedback através da coleta de dados de seus usuários. Especialmente para nuvens, a monitoração deve tratar diferentes tipos de recursos virtualizados de comportamento dinâmico, como métricas de desempenho da infraestrutura, logs de aplicações e dados sobre execução em ambientes distribuídos. Este artigo apresenta uma arquitetura de monitoração em nuvens concebida em camadas que é escalável, elástica, responsiva e extensível. A arquitetura foi validada através de uma implementação baseada em ferramentas open-source para atuar sobre um ambiente de processamento de dados massivos, onde foi possível comprovar sua escalabilidade.*

Abstract. *System monitoring has an essential role in management, maintenance and to provide feedback through data collection. Particularly, cloud monitoring must address several virtualized resources of dynamic behavior, such as infrastructure performance metrics, application logs, and data produced in distributed environments. This paper presents a layered cloud monitoring architecture which is scalable, elastic, responsive and extensible. Such architecture has been tested through an implementation based on open-source tools to monitor a massive data processing environment in which its scalability has been confirmed.*

1. Introdução

A facilidade e o baixo custo para se obter recursos computacionais em ambientes de nuvem faz com que soluções desse tipo se tornem cada vez mais populares, oferecendo disponibilidade de processamento, comunicação e armazenamento virtualmente infinitos. A flexibilidade no provisionamento de recursos favorece o modelo de negócios *pay-per-use*, no qual consumidores são tarifados por demanda. Qualquer que seja sua finalidade e o nível do serviço ofertado, nuvens se beneficiam do uso de máquinas virtuais (VMs) interconectadas por canais de comunicação de alta velocidade. Seu uso favorece tanto provedores como seus clientes. Nuvens são complexas se considerarmos sua infraestrutura de *hardware* (constituída por servidores, estrutura de armazenamento, ativos de redes, refrigeração, etc.), as plataformas que suportam (muitas vezes constituídas por múltiplos sistemas operacionais) e as aplicações que executam.

Dada essa complexidade, administrar os usuários, processos e consumo de recursos em qualquer nível, seja do ponto de vista do provedor da nuvem, ou de um cliente que tenha muitas VMs, se torna impraticável sem um sistema adequado de **monitoração** que possa gerir essa estrutura de maneira eficiente e escalável. A criação de recursos virtuais acrescenta ainda mais um nível de complexidade, já que permite a criação e remoção de VMs a qualquer momento. Sistemas de monitoração para nuvem, de maneira diferente de sistemas de monitoração usuais, precisam ser capazes de lidar com a volatilidade no estado do sistema [Park et al. 2011].

Estima-se que ao menos 25% dos dados corporativos seja proveniente de sistemas de monitoração, com previsão de crescimento seguindo o aumento de tamanho dos *datacenters* [Kutare et al. 2010]. Com o intuito de monitorar informações, existem diversas abordagens no mercado [Ryder 2016, Massie et al. 2004, VMWare 2018, Prometheus 2018]. Na maioria das vezes as soluções existentes surgem na forma de ferramentas que desempenham diversas funções e acompanham uma variedade de *plugins*, mas que apresentam limitações quando aplicadas ao ambiente de nuvem. Além disso, por se tratar de uma área multidisciplinar, um grande desafio na monitoração reside na forma de integração da variedade de ferramentas e informações a serem coletadas.

A contribuição deste trabalho é apresentar *Seshat*, uma arquitetura de monitoração elástica, escalável, resiliente e extensível para sistemas em nuvem¹ que oferece flexibilidade na montagem de soluções. Ao identificar os diversos elementos de um sistema de monitoração de forma abstrata e explicitar os seus requisitos em termos da monitoração pretendida, *Seshat* permite reunir diferentes ferramentas e componentes com funções específicas de forma que seja possível avaliar o estado de um *cluster* como um todo ou de uma aplicação implementada em nuvem e fornecer *insights* relevantes sobre seu desempenho. A arquitetura proposta foi validada em um cenário real, para a monitoração de aplicações desenvolvidas para mineração de dados, as quais são executadas em ambiente Spark sobre HDFS, em um conjunto de VMs gerenciadas pelo controlador de nuvens Openstack². Outra contribuição do trabalho foi implementar um sistema de monitoração composto por ferramentas *open source* que foram configuradas e integradas, inclusive contornando algumas de suas limitações, estruturada via rede distribuída de contêineres, atendendo aos requisitos da arquitetura. De acordo com nosso conhecimento, essa abordagem ainda não foi considerada.

2. Requisitos para um ambiente de monitoração em nuvem

Ao considerarmos uma solução para monitoração de sistemas em nuvem, devemos abordar tanto as características esperadas do sistema como um todo, quanto as demandas específicas geradas pelos dados que devem ser tratados.

2.1. Demandas da arquitetura

A monitoração de ambientes em nuvem pode ser abordada sob duas perspectivas: uma *visão macro* sobre as propriedades gerais que um sistema precisa apresentar, e uma *visão micro* que apresenta a complexidade imposta pelos componentes necessários à construção

¹*Seshat* é o nome de uma deusa egípcia considerada “guardadora de registros, responsável por registrar a passagem do tempo” (Wikipedia, <https://en.wikipedia.org/wiki/Seshat>)

²(Openstack, <https://www.openstack.org/>)

de sistemas de monitoração e a forma como são interligados. Na **visão macro**, sistemas em nuvem, se comparados a sistemas de hospedagem tradicionais, exigem monitoração mais complexa, escalável e robusta. Portanto, esses sistemas precisam ser capazes de atender a uma série de atributos, tais como **escalabilidade**, **elasticidade**, **responsividade** e **extensibilidade** [Aceto et al. 2013]. No que se refere a uma **visão micro**, é necessário conhecer a infraestrutura e os sistemas para a aplicação correta de agentes coletores de dados. Além disso, é preciso conhecer os protocolos usados na comunicação entre coletores e as outras partes do sistema de monitoração. O armazenamento das informações também possui papel relevante, o que requer a aplicação de um banco de dados que seja capaz de lidar com o volume de informações coletadas. Outro desafio que surge a partir do armazenamento, é que dados coletados e armazenados não fornecem conhecimento por si só. Dessa forma, técnicas de mineração e visualização de dados permitem extrair mais informação a partir dos dados coletados.

Por se tratar de um sistema direcionado à monitoração de parâmetros de computação em Nuvem, **escalabilidade** passa a ser não só uma característica desejável, mas essencial quando se considera cargas de elevados volumes e variabilidades. A escalabilidade pode ser obtida de diversas formas através da arquitetura proposta, como será ilustrado na Seção 4.2.

Um sistema de monitoração também deve permitir avaliar periodicamente o estado do sistema computacional e facilitar o rastreamento das causas de falhas. Esse desafio requer que uma série de componentes sejam analisados, englobando desde servidores físicos, redes e máquinas virtuais até *logs* de aplicações. Fica evidente, portanto, a necessidade de uma plataforma abrangente e confiável de monitoração para que provedores consigam localizar problemas dentro de sua infraestrutura e para que consumidores compreendam se as causas de problemas relacionados a desempenho são provenientes da infraestrutura ou da própria aplicação [Romano et al. 2011].

2.2. Métricas e logs

Em sistemas de monitoração para ambientes em nuvem, os dados de interesse podem ser classificados segundo essas duas categorias: *métricas* são dados simples, normalmente de tipos numéricos (inteiros ou de ponto flutuante), obtidos periodicamente. Nessa categoria se enquadram dados como a porcentagem de utilização de uma CPU, o volume de dados recebidos ou enviados por unidade de tempo, etc. Normalmente métricas são disponibilizadas através de pontos de medição que podem ser acessados para se obter o valor corrente de uma certa grandeza. O produto resultante da monitoração periódica é uma série temporal. Cada dado isoladamente é bastante simples e a evolução de seus valores ao longo do tempo tende a ser mais relevante que valores individuais. Por outro lado, *logs* (ou registros de execução) são normalmente mensagens textuais de formato mais complexo, sem um padrão global pré-definido, que podem trazer em uma única linha uma variedade de informações, como detalhes de uma requisição de um cliente recebida pelo sistema, ou as condições de um erro detectado durante a operação, por exemplo. Normalmente são produzidos por comandos inseridos em uma aplicação ou no sistema operacional pelo programador para registrar a ocorrência de determinados eventos e são armazenados inicialmente em arquivos de *logs* configurados pelo administrador do sistema.

Por suas naturezas diversas, todo sistema de monitoração em nuvem deve reconhecer as diferenças entre essas duas categorias e estar preparado para tratá-las de forma

diferenciada. Métricas, pela sua simplicidade, ocupam isoladamente pouco espaço, mas devem ser facilmente recuperadas em longas sequências, sempre com a informação de tempo associada. *Logs* precisam ser tratados como registros complexos que ocupam individualmente mais espaço, com padrões que podem variar entre uma e outra mensagem. Nesse caso, o acesso a eles podem exigir mecanismos de detecção de padrões complexos para extrair as diversas informações que podem estar contidas em uma única mensagem.

3. A arquitetura de monitoração *Seshat*

Para simplificar o processo de identificação das diversas atividades envolvidas no processo de monitoração em um ambiente em nuvem, *Seshat* foi organizada com base em quatro camadas principais: (i) Coleta, (ii) Transporte, (iii) Armazenamento e (iv) Visualização de dados. Além dessas, duas camadas opcionais oferecem funcionalidades extras: (v) Alertas e (vi) Correlação de Eventos. Essa organização é ilustrada na Figura. 1.

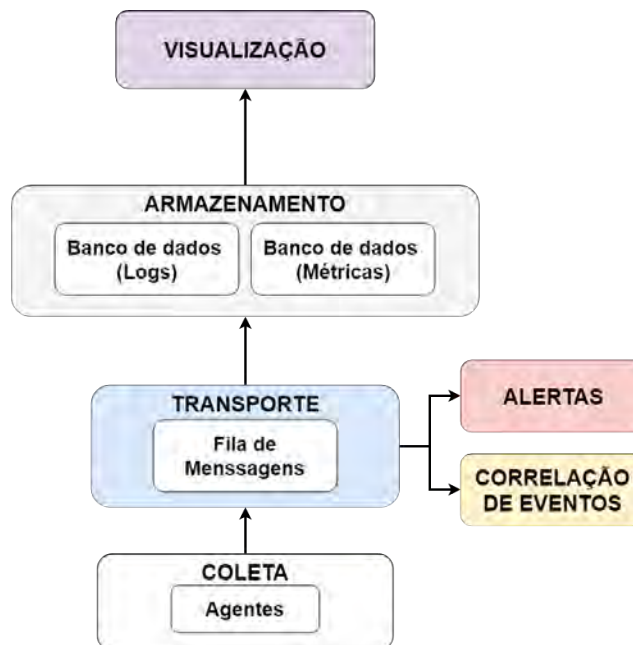


Figura 1. Elementos da arquitetura de monitoração de nuvem *Seshat*.

O fluxo de dados se inicia na camada de **Coleta**, que faz a aquisição dos dados a serem monitorados, estabelecendo o primeiro contato entre a monitoração e o serviço ou *host* alvo. Nessa camada, agentes coletores de dados são inseridos em *hosts* e aplicações para que suas informações sejam enviadas para a camada de **Transporte**. Nesta, são reunidas as informações de todos os agentes coletores através de mensagens. Em seguida, essas mensagens são disponibilizadas para serem processadas. Mensagens brutas adquiridas dessa forma passam por transformações como filtragens e agregações para se adequarem ao formato desejado. Logo depois todas as mensagens são entregues para a camada de **Armazenamento**, onde são inseridas em um bancos de dados adequado ao tipo de informação (métrica ou *log*). Finalmente, na camada de **Visualização** informações gerais e específicas a respeito de toda a infraestrutura são apresentadas de forma sistematizada e de fácil compreensão. Adicionalmente, a camada de **Alertas** é responsável por avisar o administrador caso certas combinações de valores sejam encontradas nos dados monitorados. Paralelamente, a camada de **Correlação de Eventos** manipula todas as informações

recebidas em busca de padrões mais complexos, a fim de produzir conhecimento mais elaborado sobre a condição do sistema. As próximas seções detalham o funcionamento de cada camada.

3.1. Coleta

A camada de Coleta faz a interface entre o sistema e as entidades monitoradas. Assim, para monitorar um determinado recurso é necessário instalar um agente adequado. A aquisição de dados nesse nível pode se dar de duas formas básicas: *push* (o coletor obtém e envia os dados monitorados de forma autônoma ou pró-ativa) ou *pull* (a infraestrutura de coleta aciona o coletor para cada dado que deseja obter). *Seshat* adota o modelo *push*, pois ele distribui a tarefa de coleta, reduzindo a ocorrência de gargalos. Cada agente coleta os dados sob sua responsabilidade e os envia para o restante do sistema sem a necessidade de uma entidade central realizar consultas sobre agentes. Assim, a coleta de dados apresenta escalabilidade horizontal, diferindo da escalabilidade vertical de sistemas de monitoração de propósito geral. Os agentes precisam ser leves e eficientes para se adequar à propriedade de adaptabilidade. Quanto mais simples as instruções executadas, menos invasivo é o agente e consequentemente menor o custo computacional de instrumentação.

Quando um agente coleta métricas, ele executa verificações (*checks*) continuamente que por sua vez produzem eventos (ou simplesmente mensagens). Um evento é um bloco de dados que possui *timestamp*, informações a respeito do host de origem, versão do agente e o resultado de cada *check* executado. Um *check* pode verificar variadas métricas como: uso de CPU, ocupação do disco rígido ou uso da interface de rede. O resultado de um *check* pode ser em formato binário ou, na maioria das vezes, um valor numérico.

Quando um agente coleta *logs*, ele é configurado para observar mudanças em arquivos de *logs* de aplicações ou do sistema operacional. Para cada nova linha adicionada ao arquivo, o agente cria um novo evento com o conteúdo da linha (sequência de caracteres), *timestamp* e outras informações relacionadas.

3.2. Transporte

Devido à natureza imprevisível do fluxo de dados de eventos e *logs* de aplicações, as rajadas de dados (*bursts*) são comuns em sistemas de grande porte e podem causar perdas de dados se tratadas incorretamente [Armbrust et al. 2010]. Assim, para evitar gargalos de recepção na camada de Armazenamento, filas de mensagens (também chamadas de *brokers*) são introduzidas para o armazenamento temporário a fim de garantir o transporte de informações cruciais aos banco de dados. No entanto, *brokers* normalmente não possuem mecanismos configurados para o envio de informações aos demais elementos da arquitetura, exigindo a inclusão de dois novos componentes: o *Shipper* e o *Indexer*.

3.2.1. Fila de mensagens

Devido à sua alta taxa de produção, é inviável que os dados sejam enviados diretamente para a camada de armazenamento. Em geral, a taxa de ingestão de bancos de dados tende a ser menor que a taxa de produção de dados pelos agentes em momentos de pico. Assim, diversas mensagens seriam perdidas no processo de envio, e ocasionalmente, ocorreria

degradação de desempenho. Dessa forma, filas de mensagens atuam como um intermediário (*buffer*) para garantir que os dados coletados cheguem eventualmente à camada de armazenamento.

Filas de mensagem funcionam segundo o paradigma *Publish/Subscribe*. Produtores publicam suas informações em filas e consumidores se inscrevem nessas filas para receberem informações [Eugster et al. 2003]. Em geral, as filas são divididas em tópicos de acordo com o assunto das mensagens. A separação entre *logs* de sistema e métricas é um exemplo de divisão de tópicos para o caso da monitoração. Vale citar que as interações entre produtores e consumidores através da fila de mensagens evita que seja preciso bloquear um produtor ou consumidor. Desta forma, a velocidade com que cada produtor ou consumidor gera ou consome as mensagens não tem relação direta, resolvendo o problema da disparidade entre as taxas de produção da camada de coleta e inserção da camada armazenamento.

Por fim, vale mencionar o atributo de persistência. Em geral, filas de mensagem mantêm seus dados armazenados em memória. No entanto, para situações de *stress* onde ocorre o uso crítico de memória, a fila é capaz de armazenar parte dos dados no disco rígido, oferecendo ainda mais disponibilidade na recepção de mensagens.

3.2.2. Shipper e Indexer

Uma vez adicionada ao sistema, a fila de mensagens é incapaz de buscar informações dos componentes da camada de coleta e também de enviá-las diretamente à camada de Armazenamento. Cada um desses sistemas pode ter suas peculiaridades, que não necessariamente se adaptam ao modelo da fila. Dessa forma, as informações devem passar por dois componentes que fazem a adaptação entre esses sistemas, como ilustrado na Figura. 2.

O primeiro componente é o *shipper*, responsável pela recepção de dados obtidos por agentes em variados tipos de protocolos e imediatamente enviá-los à fila, desempenhando assim o papel de produtor. Sua função é servir como um receptor universal e atuar como ponte na comunicação entre os agentes e a fila. É importante observar que existem casos onde os agentes coletores já possuem *plugins* de comunicação com a fila, tornando o uso do *shipper* opcional.

O segundo componente, o *indexer*, é mais complexo e é responsável por consumir as mensagens da fila e enviá-las à camada de Armazenamento. Além disso, realiza operações mais custosas como filtragem e *parsing* de mensagens. De maneira análoga ao *shipper*, componentes da camada de Armazenamento também podem ter *plugins* para acessar diretamente a fila. No entanto, para esses casos, o *indexer* ainda é necessário para tratar dados brutos.

3.3. Armazenamento

Estabelecidos a coleta e o transporte, é necessário registrar/armazenar os dados. Em vista disso, a camada de Armazenamento guarda de forma estruturada as informações recebidas das camadas inferiores e permite a criação de um histórico para análises mais detalhadas. Para se adequar ao volume de informações e as necessidades de escalabilidade, *Seshtat* prevê o emprego de bancos de dados especializado para séries temporais.

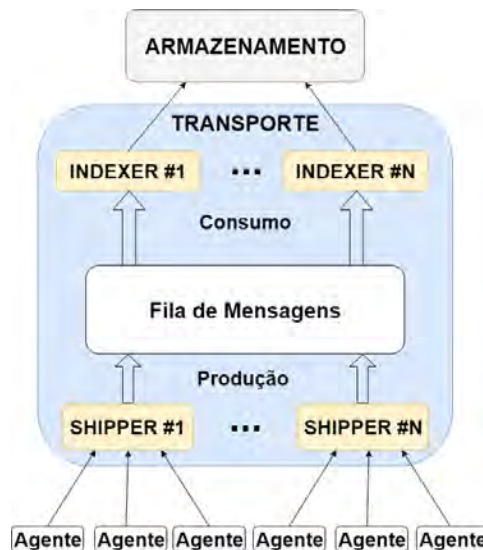


Figura 2. Estrutura interna da camada de Transporte

Independente do tipo de dado considerado, um sistema de monitoração deve ser capaz de armazenar todos os dados recebidos, atender a consultas eventuais sobre os valores armazenados e apenas raramente executar operações de exclusão de dados. Por sua natureza, dados resultantes de monitoração não devem ser alterados. Por isso, há elevada demanda por escrita, eventual leitura e muito raramente exclusões. Esse comportamento define a necessidade de uso de bancos de dados específico para séries temporais. Adicionalmente, sistemas de monitoração de nuvem também possuem grande demanda por armazenamento escalável e robusto [Goldschmidt et al. 2014]. Bancos de dados relacionais padrão são melhor executados em um único servidor a fim de manter a integridade dos mapeamentos de tabela e evitar a complexidade envolvida no computação distribuída, apresentando baixa elasticidade. Portanto, um banco de dados do tipo *NoSQL* é mais adequado por sua facilidade em se adaptar a variações da carga de trabalho através de escalabilidade horizontal.

Apesar de tanto métricas quanto *logs* poderem ser considerados séries temporais, existem peculiaridades no tratamento de cada um. Por exemplo: (i) após o processo de *parsing* de *logs* podem possuir vários campos indexáveis, enquanto métricas possuem apenas um ou dois campos indexáveis além do *timestamp*; (ii) em geral, *logs* possuem diversos caracteres em cada campo, enquanto métricas possuem apenas um único valor numérico; (iii) consultas relacionadas a cadeias de caracteres são bem distintas de consultas relacionadas a valores numéricos e, potencialmente, mais complexas. Considerando todas essas particularidades, não existe necessariamente um componente de armazenamento capaz de endereçar de maneira eficiente e simultânea todas essas questões. Por isso, podem ser necessários bancos de dados distintos para *logs* e métricas.

3.4. Visualização

A camada de visualização permite que os dados armazenados sejam apresentados de forma gráfica. Esse é o ponto da arquitetura onde *dashboards* interativos e técnicas estatísticas podem ser usados para avaliação das informações e acompanhamento contínuo do estado sistema. Dessa maneira, o administrador/usuário é capaz de receber notificações e tomar decisões. Dada a variedade de opções para ferramentas gráficas, não há uma escolha fixa ou limitação de componentes a serem utilizados para apresentar os dados que

estão armazenados. Como será explicado na Seção 4, neste trabalho foram utilizadas ferramentas distintas para apresentação de métricas e *logs* em função de suas diferenças.

3.5. Alertas

Alertas têm a função de chamar a atenção do administrador a respeito de algum evento inesperado. Por exemplo, se a utilização de CPU de um *host* monitorado alcançar 99% quando na verdade não deveria passar dos 80%, o sistema deve reportar a ocorrência incomum ao administrador. Dessa forma, esta camada realiza suas funções através da configuração de *alarmes* e envio de *notificações*. Alarmes são checagens pré-estabelecidas pelo administrador, por exemplo, para métricas que ultrapassem um dado limiar, ou algum recurso que falhe. A partir desses eventos, notificações são disparadas sob forma de e-mails, mensagens SMS ou qualquer tipo de comunicação que informe o administrador sobre o contexto do sistema.

3.6. Correlação de eventos

Ainda derivando da camada de transporte, muita “inteligência” pode ser obtida a partir do uso de uma camada de correlação de eventos. A correlação de eventos (*Event Correlation*) serve como um componente de manipulação de *streams* de eventos, capaz de agregar informações, criar filtros e produzir conhecimento mais elaborado sobre a situação do sistema, permitindo um melhor tratamento da informação durante sua coleta.

4. Aplicação da arquitetura a um caso real

Para avaliar a arquitetura *Seshat*, ela foi instanciada em uma estrutura completa para monitoração de um ambiente em nuvem real: um *cluster* de processamento de dados massivos em ambiente Spark sobre HDFS em máquinas virtuais orquestradas pela ferramenta Openstack. Nesse processo, consideramos os requisitos discutidos na Seção 2 e os detalhes da arquitetura descritos na Seção 3. Para a camada de **coleta**, consideramos ferramentas de coleta que operam no modelo *push*, por serem mais adequadas para garantir escalabilidade. Usamos agentes *Sensu*³ como a ferramenta básica para extração de métricas, servidor *Sensu* como um ambiente integrado mais flexível para configuração de agentes de coleta, e quando possível, *Beaver*⁴ como extrator de *logs*, responsável por ler os arquivos de *logs* gerados pelos diversos sistemas e encaminhar as novas mensagens encontradas para a camada de transporte. Para a camada de **Transporte** utilizamos *RabbitMQ*⁵ como a fila de mensagens e *Logstash*⁶ para implementar os *shippers* e *indexers* para o processamento de *logs*. Na camada de **armazenamento**, utilizamos *Influxdb*⁷ para o armazenamento de métricas e *ElasticSearch*⁸ para o armazenamento de *logs* e também métricas. As ferramentas de **visualização** utilizadas foram *Kibana*⁹ (*logs*), *Grafana*¹⁰ (métricas em geral) e *Uchiwa*¹¹ (acompanhamento dos agentes *Sensu*). *Riemann*¹² foi

³(*Sensu*, <https://sensu.io/>)

⁴(*Beaver*, <https://python-beaver.readthedocs.io/en/latest/>)

⁵(*RabbitMQ*, <https://www.rabbitmq.com/>)

⁶(*Logstash*, <https://www.elastic.co/products/logstash>)

⁷(*Influxdb*, <https://www.influxdata.com/time-series-platform/influxdb/>)

⁸(*ElasticSearch*, <https://www.elastic.co/products/elasticsearch>)

⁹(*Kibana*, <https://www.elastic.co/products/kibana>)

¹⁰(*Grafana*, <https://grafana.com/>)

¹¹(*Uchiwa*, <https://uchiwa.io/#/>)

¹²(*Riemann*, <http://riemann.io/>)

utilizada como a ferramenta de correlação de eventos. Para nossos testes, não foi necessário utilizar uma ferramenta para geração de alarmes, apenas os recursos de geração de alarmes das ferramentas de visualização utilizadas já foram suficientes.

Para a implantação (*deployment*) do sistema, os serviços foram disparados em contêineres *Docker*¹³ e lançados como serviços em um *cluster Docker Swarm*¹⁴. Para alguns serviços foram utilizadas imagens *Docker* disponibilizadas pelas próprias empresas desenvolvedoras, enquanto outras precisaram ser "transformadas" em contêineres no formato *Docker*. É importante ressaltar que, mesmo quando disponíveis em imagens pré-construídas, existe um considerável esforço para configurar e compatibilizar formatos de entrada e saída de dados esperados por cada serviço. Além disso, uma série de redes inter-contêineres foram configuradas para orquestrar a comunicação entre os serviços. Adicionalmente, foram incluídas configurações para permitir que os contêineres que fornecem esses serviços sejam disparados e configurados automaticamente entre os servidores dedicados à monitoração através da plataforma *Swarm*. Dessa forma, o sistema se ajusta a um aumento de carga. Como produto final, foi desenvolvida uma receita em *Docker Swarm* onde todas as configurações e alocações de serviços são instanciadas de forma automática e distribuída. Podemos analogamente afirmar que, assim como um sistema é composto por vários programas, *Seshat* é composto por diversas ferramentas e a integração entre elas é determinada pela arquitetura proposta.

4.1. Casos de uso

Com a monitoração em operação, todos os nossos experimentos na área de processamento de dados massivos com Spark passaram a ser monitorados: uma vez configurado e disparado um *cluster* virtualizado para execução de uma aplicação, os *logs* das máquinas virtuais Java, do ambiente Spark e do HDFS são coletados, assim como diversas métricas de interesse de cada máquina virtual do *cluster*, bem como as máquinas físicas que as hospedam e os dados de consumo de energia de cada *host*, obtidos de um PDU (*Power Distribution Unit*) inteligente. A Figura 3 apresenta um dos *dashboards*¹⁵ desenvolvidos no ambiente para a exploração visual dos dados. Nela podemos observar como se comportam ao longo do tempo as métricas de uso de CPU, memória RAM, escritas em disco e tráfego de rede nos servidores de monitoração durante a execução de uma aplicação Spark.

Ao longo da nossa experiência com o sistema, diversos foram os casos onde a adoção da arquitetura simplificaram tarefas de administração e gerência, e onde as informações coletadas foram úteis para na análise de comportamento de diferentes aplicações. Três desses casos são descritos a seguir.

Coletas de lixo excessivas por erros de configuração: Ao investigar casos em que algumas execuções de uma aplicação Spark levavam muito mais tempo que outras para terminar, utilizamos um *dashboard* Grafana para acompanhar a carga das máquinas virtuais. Verificamos então que naquelas execuções, a aplicação era suspensa temporariamente. Buscando nos *logs* armazenados os eventos que ocorreram no momento da suspensão, foi possível observar que o problema estava associado ao processo de coleta

¹³(*Docker*, <https://www.docker.com/what-docker>)

¹⁴(*Docker Swarm*, <https://docs.docker.com/engine/swarm/key-concepts/>)

¹⁵Os *dashboards* foram gerados diretamente pela ferramenta e seguem sua identidade visual



Figura 3. *Dashboard* de monitoração acompanhando um grupo de hosts, onde pode-se acompanhar a utilização de CPU, consumo de memória, escritas em disco e tráfego de rede enquanto uma aplicação Spark é executada em um *cluster* virtualizado.

de lixo das JVMs envolvidas. Uma reconfiguração da memória e outros elementos das JVMs resolveu o problema.

Perfis de consumo de energia imprevistos: Em outra situação, observamos um consumo anormal de energia nas máquinas do *cluster*, com um comportamento periódico e de longa duração. Montamos então um *dashboard* com métricas de carga de CPU, tráfego de rede e consumo de energia das máquinas, onde pudemos verificar que o consumo de energia não tinha relação direta com as aplicações em execução. Observamos que o aumento de consumo tinha relação direta com mensagens enviadas pelo controlador OpenStack do *cluster* para o restante do sistema, o que permitiu identificar uma tarefa de atualização periódica que precisou ser reconfigurada.

A modularidade da arquitetura simplifica a mudança de escolhas: Outra situação em que a aplicação da arquitetura se mostrou extremamente útil foi durante o início dos testes de escalabilidade descritos a seguir. Inicialmente, havíamos escolhido *Influxdb* como nosso sistema de armazenamento de métricas. Quando iniciamos os testes de escalabilidade, que exigiam a instalação em um conjunto de máquinas para distribuir a tarefa de armazenamento, verificamos que a versão de código aberto do *Influxdb* não inclui os recursos para operação distribuída. Depois de alguma análise, decidimos adotar um outro *Elastic Search* com pequenas otimizações como base de dados para métricas, já que ele inclui recursos para operação distribuída. Graças à estrutura modular ditada pelo uso da arquitetura *Seshat*, a retirada de um sistema e a inclusão do outro foi feita de forma simples, exigindo poucas alterações no restante do sistema.

4.2. Desempenho e escalabilidade

Dois experimentos foram realizados sobre a ferramenta de monitoração. O primeiro avaliou o comportamento do sistema sob *stress* devido à inserção de um elevado volume de

métricas. Foram testadas 2 configurações: (i) todo o sistema executando em um servidor único com processador core-i7 2600 3,4GHz com 16 GB de memória RAM e (ii) executando de forma distribuída em 3 servidores idênticos ao anterior dedicados exclusivamente à monitoração. Para um servidor único, havia apenas uma instância de cada componente do sistema(*shipper*, *indexer*, fila de mensagem, etc), enquanto o distribuído contou com instâncias da fila de mensagens e de bancos de dados em cada uma de 3 máquinas disponíveis. Além disso, foram utilizadas 9 instâncias de *indexers*, também distribuídos entre as 3 máquinas, para consumir da fila. Para os testes foram coletadas 7 tipos de métricas diferentes de cada uma das 24 máquinas virtualizadas na nuvem cliente, composta por 6 máquinas servidoras Intel Xeon E5-2620v4 2.1GHz com 32 GB de RAM. A orquestração da nuvem cliente foi feita pelo Openstack v2.3.1 em outro servidor, também com processador core-i7 2600 3,4GHz com 16 GB de memória RAM. Toda essa estrutura se encontra interligada por um *switch gigabit* 3Com Baseline 2928-SFP Plus *ethernet*. O intervalo das medições de métrica foi variado entre 1, 2 e 5 segundos. Foram realizadas medições em 5 intervalos de 30 minutos espaçados por uma janela de descanso de 5 minutos. A Tabela 1 apresenta os resultados obtidos nesse teste com a média das 5 execuções.

Tabela 1. Desempenho e escalabilidade do sistema de monitoração sob volume elevado de dados de métricas

Servidor	Interv.	Total msgs.	Msgs. perdidas	% perdas
Único	1 s	280800	5795.8 ± 250.8	2.06%
	2 s	151200	111.4 ± 37.0	0.07%
	5 s	60480	22.2 ± 12.1	0.04%
Distribuído	1 s	280800	4535.2 ± 79.3	1.62%
	2 s	151200	90.8 ± 29.6	0.06%
	5 s	60480	6.6 ± 10.9	0.01%

A Tabela 1 apresenta a comparação entre os intervalos de medição em cada configuração de servidor, a quantidade média de mensagens perdidas $\bar{x}$ junto ao comprimento de seu intervalo em um nível de confiança de 90% e a porcentagem das mensagens perdidas em função do total de mensagens esperadas. Nesse experimento, sabemos o total de mensagens que deveriam ser armazenadas, já que o experimento é determinístico nesse sentido. Ao consultar o sistema após o experimento para verificar quantas mensagens foram realmente armazenadas, podemos determinar o número de mensagens perdidas.

Podemos observar que até nas configurações com maior volume de mensagens por unidade de tempo, as perdas foram baixas, mesmo no ambiente com um único servidor. Com os recursos disponíveis no nosso laboratório, não fomos capazes de gerar uma configuração que levasse o serviço a um colapso. Apesar disso, podemos ver que a versão distribuída do sistema se mostrou capaz de lidar melhor com a carga, tendo perdas menores e exibindo uma variabilidade menor no seu comportamento.

O segundo experimento avaliou o comportamento do sistema sob stress quando submetido a um elevado volume de mensagens de *log*. De maneira similar ao primeiro foram utilizadas as mesmas configurações de servidor único e distribuído, com exceção dos *indexers*, que foram aumentados para 21 instâncias no caso distribuído. Como carga, usamos uma aplicação Spark sobre HDFS executando em 10 máquinas virtuais da nuvem cliente. Também variamos a verbosidade do sistema de *logs* do Spark entre INFO e

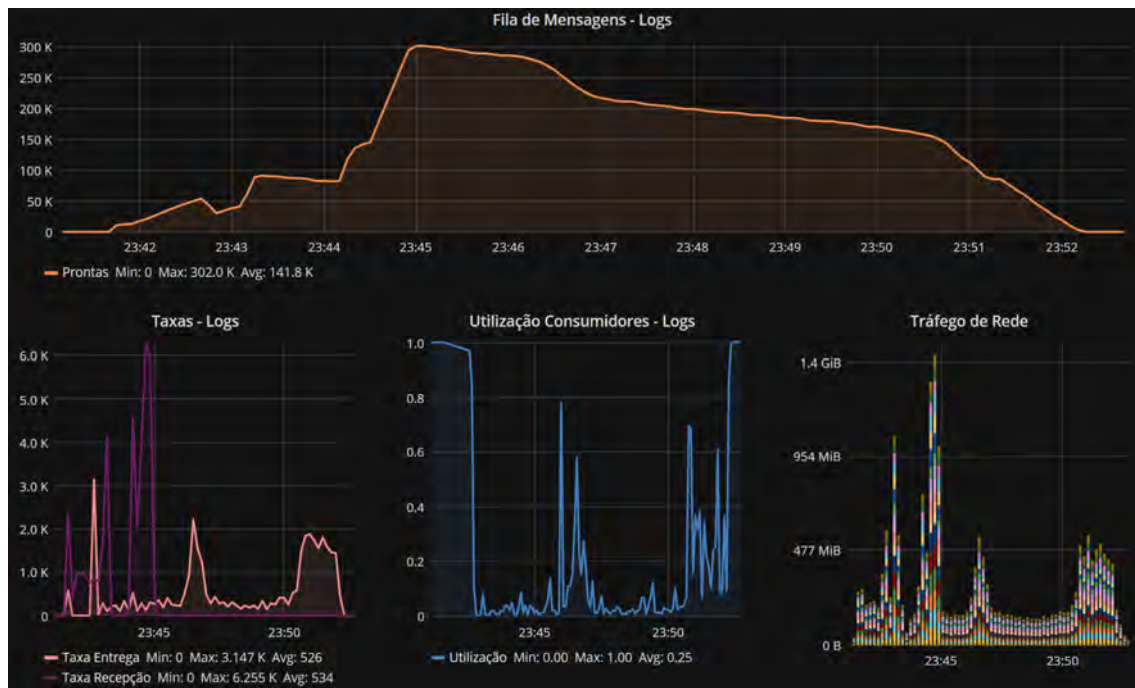


Figura 4. Dashboard de monitoração da fila de mensagens durante o experimento sobre a escalabilidade no sistema de processamento de logs.

DEBUG. Cada teste foi executado 10 vezes em cada um dos níveis de verbosidade e intervalados de 15 minutos para permitir que todas as mensagens presentes na fila fossem consumidas. O *dashboard* da Figura 4 apresenta um exemplo das informações disponíveis a respeito da fila de mensagens na versão distribuída durante uma execução da aplicação Spark na verbosidade DEBUG. Nela é possível observar ao longo do tempo o acúmulo de mensagens na fila (gráfico de linha laranja no topo), como variam as taxas de entrega e recepção de mensagens (gráfico de linhas rosa, à esquerda), a eficiência em que as mensagens estão sendo consumidas (gráfico de linha azul, no centro) e o tráfego de rede agregado dos 3 servidores da monitoração (gráfico de barras coloridas à direita).

A Tabela 2 apresenta uma comparação entre as configurações de servidor para dois níveis de verbosidade diferentes em uma aplicação Spark. São eles: (i) INFO, que é o volume comum de mensagens de *log* de uma aplicação Spark e (ii) DEBUG com um elevado volume de mensagens de *log*. Para o experimento foi avaliado o atraso médio das mensagens dado em segundos, a taxa de entrega das mensagens por segundo aos consumidores e a taxa de recepção das mensagens por segundo na fila. Esse atraso é caracterizado pela diferença de tempo entre o momento que a mensagem é produzida pelo agente e o momento que é inserida no banco de dados. Todos os valores são acompanhados de seus respectivos comprimentos de intervalo em um nível de confiança de 90%.

Tabela 2. Desempenho e escalabilidade do sistema de monitoração sob volume elevado de mensagens de logs.

Servidor	Verbosidade	Núm. msg	Atraso	Taxa de Entrega	Taxa de Recepção
Único	INFO	128769	$163.9_s \pm 0.8$	188.3 ± 4.3	510.5 ± 26.8
	DEBUG	2726140	$449.5_s \pm 0.5$	356.9 ± 16.6	1804.4 ± 146.3
Distribuído	INFO	128209	$107.0_s \pm 0.5$	250.4 ± 12.1	493.6 ± 25.2
	DEBUG	2699265	$192.7_s \pm 0.2$	778.9 ± 52.6	2044.7 ± 182.4

Ao analisar a Tabela podemos constatar que o atraso médio chega a ser o dobro para o servidor único quando comparado ao servidor distribuído. Além disso, a taxa de entrega chega a ser o dobro dada a maior quantidade de consumidores na fila. Quanto a

taxa de recepção não há diferença significativa entre as configurações de servidor único e distribuído. No entanto, quando comparados entre a verbosidade a diferença chega a aproximadamente o triplo. Isso se deve principalmente ao volume de mensagens enviadas por cada agente de coleta.

5. Trabalhos Relacionados

Monitoração é o processo pelo qual se utiliza ferramentas e medições para o gerenciamento de sistemas de TI [Turnbull 2014], sendo uma área em Tecnologia da Informação quase tão antiga quanto a própria criação de sistemas de computação. Uma contextualização da necessidade de se monitorar sistemas deste tipo foi descrita por [Aceto et al. 2013], onde também apresenta uma série de propriedades chave desejáveis em qualquer sistema de monitoração. A monitoração em nuvem definida por [Spring 2011] apresenta desejavelmente sete camadas. No entanto, a descrição de cada uma delas é feita de maneira abstrata, por vezes considerando somente o ponto de vista dos provedores. Adicionalmente, não existe qualquer menção a respeito de uma camada responsável pela visualização das informações.

Do ponto de vista tecnológico, a monitoração de informações em sistemas distribuídos vem sendo tratada de diferentes formas e ferramentas. Nagios [Ryder 2016] é um exemplo já consolidado no mercado que oferece monitoração em larga escala. No entanto, sua arquitetura centralizada e baseada em *polling* não se adequa ao dinamismo na monitoração de recursos virtuais, que são frequentemente utilizados no ambiente de nuvem. Outro problema apresentado em Nagios, que também é recorrente no Hyperic [VMWare 2018] e Prometheus [Prometheus 2018], são as limitações de escalabilidade. Todos os três sistemas foram projetados para funcionar em um único servidor, e para aumentar sua capacidade de monitoração é necessário criar novas instâncias autônomas que não compartilham dados, dificultando o gerenciamento. Por outro lado, Ganglia [Massie et al. 2004] apresenta uma arquitetura distribuída que permite escalabilidade. No entanto, foi projetado para a monitoração de clusters bem definidos e com poucas mudanças na infraestrutura, dificultando também o trato de recursos virtuais dinâmicos. Finalmente, Ganglia e Prometheus não realizam a coleta de *logs*, assim como outras ferramentas no mercado que por padrão coletam somente métricas ou somente *logs*.

6. Conclusão e Trabalhos Futuros

Monitoração de ambientes e aplicações é uma demanda crescente na comunidade de computação em nuvem. Neste trabalho apresentamos *Seshat*, uma arquitetura de monitoração concebida em camadas e organizada em componentes, com uma definição clara dos comportamentos e funcionalidades esperados em cada caso. Uma implementação da arquitetura foi realizada com base em ferramentas de código aberto e aplicada a uma nuvem computacional voltada para o processamento de dados massivos. A experiência com o sistema tem gerado diversos casos de uso bem sucedidos, como discutidos no texto.

Experimentos para avaliar a escalabilidade do sistema sob condições de carga elevada mostraram que o sistema se comporta bem, com perdas baixas. Testes com uma versão distribuída do sistema mostraram uma redução nas mensagens perdidas, redução de atrasos e redução da variabilidade. Esses resultados indicam que o sistema tem potencial para escalar horizontalmente com o aumento da carga.

Como trabalhos futuros, pretendemos realizar mais experimentos para variar as configurações do sistema distribuído para melhor compreender o impacto de elementos como a quantidade de *Shippers* e *Indexers*.

Agradecimentos

Este trabalho foi parcialmente financiado por Fapemig, CAPES, CNPq, e pelos projetos MCT/CNPq-InWeb (573871/2008-6), FAPEMIG-PRONEX-MASWeb (APQ-01400-14), H2020-EUBR-2015 EUBra-BIGSEA e H2020-EUBR-2017 Atmosphere.

Referências

- Aceto, G., Botta, A., De Donato, W., and Pescapè, A. (2013). Cloud monitoring: A survey. *Computer Networks*, 57(9):2093–2115.
- Armbrust, M., Fox, A., Griffith, R., Joseph, A. D., Katz, R., Konwinski, A., Lee, G., Patterson, D., Rabkin, A., Stoica, I., et al. (2010). A view of cloud computing. *Communications of the ACM*, 53(4):50–58.
- Eugster, P. T., Felber, P. A., Guerraoui, R., and Kermarrec, A.-M. (2003). The many faces of publish/subscribe. *ACM computing surveys (CSUR)*, 35(2):114–131.
- Goldschmidt, T., Jansen, A., Koziol, H., Doppelhamer, J., and Breivold, H. P. (2014). Scalability and robustness of time-series databases for cloud-native monitoring of industrial processes. In *Cloud Computing (CLOUD), 2014 IEEE 7th International Conference on*, pages 602–609. IEEE.
- Kutare, M., Eisenhauer, G., Wang, C., Schwan, K., Talwar, V., and Wolf, M. (2010). Monalytics: online monitoring and analytics for managing large scale data centers. In *Proceedings of the 7th international conference on Autonomic computing*, pages 141–150. ACM.
- Massie, M. L., Chun, B. N., and Culler, D. E. (2004). The ganglia distributed monitoring system: design, implementation, and experience. *Parallel Computing*, 30(7):817–840.
- Park, J., Yu, H., Chung, K., and Lee, E. (2011). Markov chain based monitoring service for fault tolerance in mobile cloud computing. In *Advanced Information Networking and Applications (WAINA), 2011 IEEE Workshops of International Conference on*, pages 520–525. IEEE.
- Prometheus (2018). Prometheus overview. <https://prometheus.io/docs/introduction/overview/>. Acessado em Abril de 2018.
- Romano, L., De Mari, D., Jerzak, Z., and Fetzer, C. (2011). A novel approach to qos monitoring in the cloud. In *Data Compression, Communications and Processing (CCP), 2011 First International Conference on*, pages 45–51. IEEE.
- Ryder, T. (2016). *Nagios core administration cookbook*. Packt Publishing Ltd.
- Spring, J. (2011). Monitoring cloud computing by layer, part 1. *IEEE Security & Privacy*, 9(2):66–68.
- Turnbull, J. (2014). *The Art of Monitoring*. James Turnbull.
- VMWare (2018). VMware vFabric Hyperic, manage application performance across physical, virtual and cloud infrastructures. <https://docs.vmware.com/en/vRealize-Hyperic/index.html>. Acessado em Abril de 2018.

Uma análise do consumo de energia de ambientes de processamento de dados massivos em nuvem

Nestor D. O. Volpini^{1,2}, Vinicius S. Conceição¹, Raphael L. Pontes¹, Dorgival Guedes¹

¹Departamento de Ciência da Computação
Universidade Federal de Minas Gerais (UFMG) – Belo Horizonte, MG

{nestor,vconceicao,raphaelluciano,dorgival}@dcc.ufmg.br

²Centro Federal de Educação
Tecnológica de Minas Gerais (CEFET-MG) – Divinópolis, MG

nestor@div.cefetmg.br

Resumo. As áreas de processamento de dados massivos (PDM, ou big-data) e de computação em nuvem têm se desenvolvido de forma integrada. Assim, PDM vem se consolidando como um dos maiores consumidores de recursos em datacenters que, por sua vez, são responsáveis por demandar em torno de 2% da energia global. Entender como elementos como o ambiente de virtualização e grau de paralelização dessas aplicações afetam esse consumo é, portanto, uma necessidade premente. Este artigo utiliza uma solução de monitoração abrangente para avaliar como o consumo de energia de aplicações big-data virtualizadas varia em função dos recursos alocados.

Abstract. Massive data processing (big-data) related fields and cloud computing have been growing conjointly. Thus, data processing is among the largest resource consumers in datacenters, consuming around 2% of global energy. Comprehension of how elements such as virtualized environments and applications' parallelization degree affect such consumption is therefore an urgent need. This article relies on a monitoring solution that provides performance metrics, data mining application logs, and data produced in distributed environments to assess how power consumption of virtualized big-data applications varies on allocated resources.

1. Introdução

O modelo de computação em nuvem tem se tornado cada vez mais presente, já que oferece acesso compartilhado via rede a um conjunto de recursos computacionais de forma conveniente, com esforço reduzido para sua configuração, além de ser escalável. A evolução da Internet permitiu a oferta de nuvens computacionais, que possuem vantagens para quem precisa destes serviços, mas não tem condições ou interesse de fazer grandes investimentos em infraestrutura. Computação em nuvem viabiliza a escalabilidade imprevisível de um negócio sem preocupar seu dono com o provisionamento extra de recursos. Além disso, o uso de computação em nuvem estabelece um modelo de remuneração por serviços, ou seja, pague pelo que processar (*pay-per-use*).

Qualquer que seja o modelo de serviço oferecido, seja *Software as a Service* (SaaS), *Platform as a Service* (PaaS) ou *Infrastructure as a Service* (IaaS), o processamento de qualquer tarefa efetivamente acontece dentro da nuvem, que por sua vez está

fisicamente estabelecida em *datacenters*. *Datacenters* podem ser caracterizados por ocupar desde de um pequeno conjunto de poucos servidores até alguns milhares de servidores acondicionados em instalações elevadas [Kooimey 2011].

O consumo de energia em *datacenters* sempre mereceu atenção. Além de ser um recurso de custo relevante, de acordo com o relatório publicado por Berkeley Lab [Shehabi et al. 2016], no ano de 2014 1,8% de toda energia consumida apenas nos Estados Unidos foi utilizada por *datacenters*. No período entre os anos de 2010 a 2014 houve um aumento de 4% na demanda. Espera-se o mesmo crescimento entre os anos de 2014 a 2020 (4%). Além disso, a matriz de produção energética se sustenta emitindo preocupantes quantidades de carbono na atmosfera de acordo com o relatório do NRDC (Natural Resources Council) [Whitney and Delforge 2014].

O processamento em *datacenters* pode ser caracterizado por dois tipos de cargas bem diferentes: podem ser utilizados para serviços *web* e para processamento de aplicações sobre grandes volumes de dados conhecidos como *big data*, também denominados dados massivos. Serviços *web* são executados predominantemente em máquinas virtuais (ou V.M.'s) ¹ há um bom tempo [Zhang et al. 2010]. Já o processamento de *big data* normalmente é realizado sem virtualização (ou *bare metal*), mas já se percebe uma tendência de também se tornarem virtualizados [Hashem et al. 2015].

Independente da carga, há consideráveis vantagens em se trabalhar com sistemas virtualizados: flexibilidade, disponibilidade, escalabilidade, melhor aproveitamento do *hardware*, segurança, custo, adaptabilidade a variações de carga de trabalho, facilidade no balanceamento de carga e também na execução de versões antigas de aplicações que por algum motivo não serão reescritas mas que ainda sejam eventualmente necessárias ² [Sahoo et al. 2010].

Muito já se estudou em economia de energia voltada para o uso de máquinas virtuais em *datacenters*. Como exemplo, podemos citar [Dupont et al. 2012] que apresenta um *framework* para alocar máquinas virtuais em um *datacenter* de modo a respeitar um conjunto de regras, entre elas economizar energia. Existe também um estudo que propõe uma arquitetura de *datacenter* eficiente energeticamente, que avalia o custo de migração de máquinas virtuais e a relação de compromisso entre consumo e desempenho [Ye et al. 2010]. No estudo feito por [Kansal et al. 2010] é apresentada uma abordagem para medir o consumo de uma V.M. em um sistema virtualizado sem o uso de instrumentos externos como wattímetros, para fins de provisionamento de carga e avaliação de custo com energia.

Portanto muito se estudou sobre as cargas tipicamente processadas em *datacenters* que lidam com máquinas virtuais, porém é importante observar um considerável crescimento no processamento de outro tipo de carga que tem se tornado comum, que lida com processamento de dados massivos [Mashayekhy et al. 2014, Gu et al. 2011]. Para processar dados massivos usualmente um conjunto de computadores físicos são ocupados de forma colaborativa, trabalhando em paralelo, para que cada um assuma uma parte da carga de processamento. Isso caracteriza uma mudança de carga, uma vez que o volume de dados a ser processado não é suportado por sistemas de servidores convencionais.

¹ Abreviatura de uso frequente, referenciando em inglês: Virtual Machines

² Também chamadas de “aplicações legadas”

Observa-se nos últimos anos um aumento do uso de sistemas de processamento de dados massivos, tais como Hadoop e mais recentemente Spark, além de outros. Hadoop pode ser visto como a implementação de *software* livre do paradigma de programação MapReduce, que se caracteriza por ser um modelo de processamento distribuído de grandes volumes de dados, de elevada robustez e escalabilidade. Já Spark se propõe a ser uma plataforma de processamento de dados massivos que adota um modelo de programação mais flexível. Spark se apresenta como um *framework* veloz por processar dados em memória, e de uso geral para processamento de dados massivos.

Portanto, há estudos que visam economizar energia em ambientes virtualizados, no entanto, nem sempre o escalonamento adequado de recursos para processar dados massivos é eficiente, especialmente no consumo de energia [Leverich and Kozyrakis 2010]. Ao longo deste artigo apresentaremos resultados obtidos através de experimentos, que demonstram que não é trivial definir a uma boa escolha na configuração de V.M.'s, ou mesmo sua política de distribuição dentro da estrutura, seja visando obter desempenho na conclusão da tarefa, ou para economizar energia, ao processar dados massivos.

Para isso, variadas configurações foram testadas em um ambiente de nuvem, sempre com seu consumo sistematicamente monitorado, além de outras métricas e *logs*, ao executar aplicações PDM. Para monitorá-las foi utilizado *emphSeshat* [Conceição et al. 2018], um sistema robusto e modularizado de monitoração desenvolvido em camadas e baseado em *softwares open-source*. Como carga para os testes, dois algoritmos de mineração de dados, Twidd e Eclat, em implementações paralelas foram testados sobre uma base de dados de 8 GB coletada a partir do Twitter. Pretendemos com esse estudo mostrar que ainda há oportunidades que consideramos significativas para economizar energia ao processar dados massivos de forma virtualizada em uma nuvem, por meio do escalonamento adequado de recursos.

2. Trabalhos relacionados à economia de energia em *datacenters*

Um maneira simples de economizar energia em *datacenters* vem da própria evolução da indústria de processadores: a incorporação de múltiplos núcleos em processadores os torna capazes de executar mais tarefas paralelas sem aumentar significativamente seu consumo [Ramanathan 2006]. Nos últimos anos é possível observar o desenvolvimento de *chips* multiprocessados de elevado desempenho e grau de paralelismo consumindo até 50 % menos energia [Kgil et al. 2006]. Também é possível atuar variando de forma combinada a frequência e a tensão de operação nos processadores do *datacenter*, técnica denominada DVFS (*Digital Voltage Frequency Scaling*), em função da carga do ambiente em que se encontram [Kim et al. 2009]. Processadores gastam menos energia ao trabalhar com um *clock* mais baixo. Mais recentemente, GPU's tem apresentado crescente utilização para processar grande volumes de dados e também são motivo de estudos do ponto de vista de seu consumo [Ferro et al. 2017].

Outra forma de economizar energia consiste em estabelecer arquiteturas de *datacenters* voltadas a um perfil energeticamente eficiente também, seja pela modificação da estrutura de rede, pela escolha dos processadores ou mesmo da forma de acesso a dados persistentes e a maneira como são replicados. Também é possível atuar nas redes de comunicação dentro do *datacenter* escolhendo sua topologia [Abts et al. 2010], ou mesmo provando que há ganhos significativos em possuir controle independente de cada

link de comunicação. Há estudos em que se opta por adotar processadores de menor capacidade, mas com baixo consumo na composição de um *datacenter*, ainda que o objetivo seja processar grandes volumes de dados. Há interessantes resultados em [Gu et al. 2011] para o processamento de uma carga MapReduce, onde está evidente que a melhor arquitetura para economizar energia com esse tipo de processamento é uma construção híbrida de servidores de pequeno porte, constituídos por processadores Atom trabalhando em conjunto com unidades de aceleração de compressão e descompressão de dados. Em um sistema denominado FAWN [Andersen et al. 2009], nodos de processamento de dados massivos são elaborados de modo que a informação persistente fica armazenada em memórias Flash e com acesso puramente log-estruturado. Os resultados obtidos mostram que FAWN é capaz de economizar até duas ordens de grandeza em consumo quando comparado com sistemas que usam discos para armazenamento.

Outra alternativa que não pode ser descartada é a possibilidade de gerar sua própria energia para diminuir ou até eliminar a dependência de concessionárias de energia, haja vista os resultados obtidos com o Parasol [Goiri et al. 2013], vantajoso inclusive na diminuição de emissão de carbono na atmosfera.

Caracterizar a forma como a energia está sendo consumida em um *datacenter* é importante, uma vez que todo sistema computacional consome energia para operar. A condição desejável é que o consumo seja sempre uma função proporcional à carga em processamento e claro com menor taxa de crescimento possível. *Datacenters* são dimensionados para suportar picos de demanda de processamento, mas na maior parte do tempo, trabalham com cargas bem menores. Então é natural que se desligue parcialmente recursos que não estejam sendo utilizados e que os mesmos sejam reativados assim que necessário. Partindo desse ponto de vista, desligar parte dos equipamentos em um *datacenter* é uma forma direta de se atuar no escalonamento de recursos. Trabalhos nessa linha podem ser encontrados, atuando de várias formas sobre escalonamento de recursos com sucesso. Um interessante trabalho desenvolvido em economia de energia procura tirar proveito de *datacenters* distribuídos geograficamente [Le et al. 2011]. Nele um estudo considerou que existem tarifas de energia diferenciadas ao longo de um dia, e que *datacenters* apresentam considerável consumo com refrigeração, e que as temperaturas dos ambientes geograficamente distribuídos também variam. Reunindo estes fatores, um modelo foi estabelecido para caracterizar seu comportamento. Partindo de *traces* reais, o modelo obtido nos levantamentos feitos foi simulado para várias políticas de distribuição de tarefas, inclusive com a antecipação da refrigeração requerida em função da carga, sempre fugindo do horário de pico. Consideráveis reduções de custo foram alcançadas neste desenvolvimento, demonstrando que escalonar tarefas geograficamente pode custar menos, e energia foi economizada ao se levar em conta a diminuição de demanda por refrigeração.

Outro interessante sistema, o GreenNebula [Berral et al. 2014], foi desenvolvido como uma extensão do OpenNebula para gerenciar máquinas virtuais que “seguem as energias renováveis”. Tomando como recurso a migração entre *datacenters* que dispõem de energia verde, máquinas virtuais são deslocadas para onde há energia disponível, com comprovado baixo custo de migração. Para decidir a melhor alocação da máquina virtual, o sistema e suas variáveis são modelados usando programação linear.

Trabalhos desta natureza reforçam o fato de que escolher um escalonamento ade-

quando à tarefa é um bom caminho para o processamento de dados massivos visando economizar energia. As cargas deste tipo muitas vezes são cargas adiáveis [Goiri et al. 2013], ou seja podem ter seu processamento protelado (até certo ponto) para serem executadas em momentos mais adequados. Isso favorece ajustá-las ao consumo de energia renovável, ou pelo menos, usar energia de menor custo.

Caminhando mais na direção de processamento de dados massivos, no trabalho de [Leverich and Kozyrakis 2010], um *cluster* Hadoop é submetido a um experimento em que conjuntos de nodos vão sendo desabilitados para que se consiga realizar uma determinada tarefa e seu desempenho seja avaliado assim como seu consumo. Hadoop tipicamente utiliza um sistema de arquivos distribuído, o HDFS (Hadoop File System) que distribui dados através de discos em um *cluster* tirando proveito de processar dados em função de sua localidade. Oferece uma estrutura robusta ao permitir a replicação de seu conjunto de dados, visando contornar falhas comuns ao processar *big data*. HDFS garante que ao armazenar réplicas nunca duas cópias são alocados em um mesmo nó e no mínimo são mantidas duas cópias em *racks* diferentes. Com uma modificação na estrutura original do HDFS, os autores desse trabalho estabeleceram um subconjunto de cobertura que permitiu ir desabilitando nós de processamento sem comprometer a conclusão da tarefa, pela exclusão de nós que estivessem fora do conjunto de cobertura. Chama a atenção nos resultados que a medida que se suprimem nós, o tempo de computação aumenta, mas a quantidade economizada de energia é significativa, variando entre 9 e até 50 %.

Há experimentos que visam melhorar o desempenho do sistema sem necessariamente aumentar na mesma proporção o consumo de energia. No estudo feito por [Wirtz and Ge 2011] a adoção de políticas de localização de dados adequada é combinada à variação de frequência de trabalho dos processadores servidores para gastar menos energia.

Outras formas já experimentadas para economizar energia em *big data* também passam por escalonar recursos. No desenvolvimento realizado por [Mashayekhy et al. 2014], um escalonador de “consumo consciente” para tarefas MapReduce é modelado como um problema de otimização por programação inteira, em que uma função objetivo busca minimizar o consumo de energia ao alocar tarefas *Map* e tarefas *Reduce* em *slots* dentro de um *cluster*, seguindo restrições determinadas pelo tempo de resposta da tarefa assim como obedecendo a característica de não iniciar a fase de *Reduce* antes que todas as tarefas de *Map* estejam concluídas. Como solução para o problema dois algoritmos gulosos que implementam heurísticas escalonaram os recursos e conseguiram resultados de até 40 % de economia, quando comparados com o escalonamento tradicional que minimiza o *makespan*, ficando próximo de uma solução ótima.

3. Metodologia de trabalho

Para o desenvolvimento desse trabalho foi escolhido o ambiente de processamento de dados massivos Apache Spark versão 1.5.2 e o sistema de arquivos Hadoop/HDFS v2.6.0 sobre Yarn. Com a finalidade de realizar as medições, foi adquirido, instalado e configurado um PDU (*Power Distribution Unit*) Raritan PX-2 da série 5000, com exatidão de 1% (norma ISO/IEC 62053-21), que é uma unidade de distribuição de energia capaz de monitorar variados parâmetros elétricos. Em nosso trabalho medimos a potência ativa por tomada, que corresponde ao consumo de cada servidor.

A orquestração dos servidores virtualizados foi feita pelo OpenStack v2.3.1 com sua estrutura montada em uma máquina servidora core-i7 2600 3,4GHz com 16 GB de memória RAM. Para os nós de computação foram usadas 6 máquinas servidoras Intel Xeon E5-2620v4 2,1GHz com 32 GB de RAM, duas interfaces de rede Gigabit Ethernet e disco sata de 2 TB sobre os quais foi possível estabelecer variadas configurações de HDFS. Todos os dados produzidos de consumo de energia, uso de CPU, disco, ocupação de memória, carga da máquina e tráfego de rede, além dos *logs* do processamento do Spark e da utilização do HDFS foram monitorados e registrados em um outro *cluster* de monitoração *Seshtat*, que contou com mais três servidores também core-i7 2600 3,4GHz com 16 GB de memória RAM. O objetivo é que a partir da coleta sistemática de dados de experimentos, fosse possível avaliar o comportamento de diferentes políticas para o escalonamento das tarefas de processamento de dados massivos, principalmente visando o consumo de energia elétrica mais eficiente.

Como carga foram processados 8 GB de dados contendo um conjunto de *posts* do Twitter, contendo *tweets* coletados usando a API da plataforma, em busca de padrões frequentes via algoritmos Twidd e Eclat em implementação paralela que foi distribuído em servidores virtualizados sobre KVM. A aplicação Twidd é uma implementação do algoritmo FPGrowth sobre a abstração de RDD's (Resilient Distributed Datasets) e resolve o problema de mineração de padrões frequentes, no qual dada uma série de transações, cada uma composta por um conjunto de itens, e um limiar, ele encontra todos os subconjuntos de itens que ocorrem no mínimo a quantidade de vezes especificada por esse limiar. A motivação para o uso desta aplicação no trabalho reside no grande custo computacional inerente ao processo de geração dos subconjuntos de itens para cada transação, o que conseqüentemente requer uso intenso do processador e se reflete em um maior uso de energia [Dias et al. 2016]. O Eclat é também um algoritmo para mineração de padrões frequentes, mas que trabalha sobre uma representação vertical da base de dados. Ao invés de um conjunto de itens para cada transação, temos conjuntos de identificadores de transações para cada item. Eclat é iterativo pela sua característica hierárquica durante a mineração e irregular pela combinatória envolvida na distribuição dos *itemsets* frequentes [Dias et al. 2016]. Dessa forma, os procedimentos realizados a cada iteração são os mesmos, mas os dados de entrada/saída variam e conseqüentemente seus padrões de comunicação também são variáveis, tornando interessante seu estudo sobre o ponto de vista de consumo energético. Para ambos os algoritmos e em função da base de dados adotada, foram definidas 128 partições, com suporte mínimo de 0,02.

3.1. Arquitetura de monitoração

Para monitorar os experimentos foi construída uma estrutura de coleta de métricas e *logs*. Como o foco desse trabalho é voltado a medir o consumo de energia sua representação esquemática pode ser vista na Figura 1, em que se destaca o PDU. A partir de suas tomadas, o PDU distribui e mede individualmente a energia elétrica que é entregue para as máquinas físicas do *cluster* que estabelece um ambiente de nuvem de processamento. Este PDU é capaz de medir tensão, corrente, fator de potência, individualmente para cada tomada, mas a grandeza de maior relevância para esse estudo é a potência ativa consumida pelos servidores e ativos de rede, uma vez que representa a energia que efetivamente realiza trabalho.

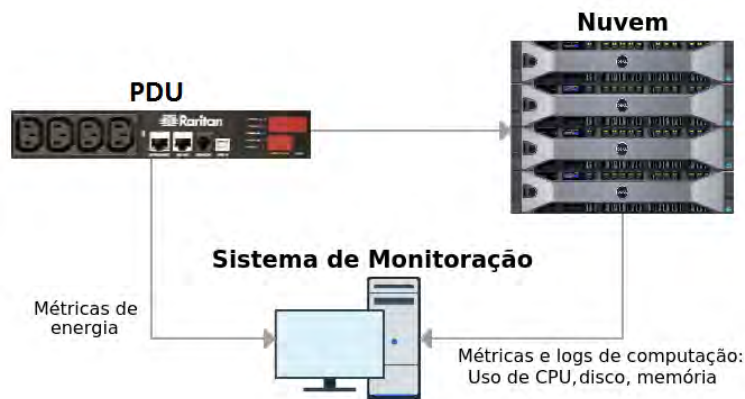


Figura 1. Esquema físico de coleta.

Nessa nuvem as aplicações que são executadas fornecem ao sistema de monitoração diversas métricas como tempo de execução, uso de CPU, utilização de memória, utilização de discos, tráfego de rede, entre outras. O registro de *logs* é capaz de coletar e registrar mensagens de execução de tarefas Spark, assim como do Yarn e do HDFS.

Para a coleta das informações providas tanto pelos servidores físicos e virtualizados assim como pelo PDU, um *cluster* constituído de três máquinas, armazenam e processam para tornar adequadas à visualização as métricas e os *logs*, gerados na nuvem. A arquitetura adotada para monitorar escolhida foi a *Seshat* que integra vários *softwares open-source*. Para coletar métricas extraídas dos servidores o *framework* Sensu foi instalado, seguindo a arquitetura e combinado ao *daemon* Collectd que é mais adequado para recuperar medições feitas pelo PDU que se comunica via SNMP. Para o armazenamento de métricas a ferramenta Influxdb foi escolhida por ser um banco de dados otimizado para o tratamento de séries temporais. Compõe ainda a arquitetura uma ferramenta que permite a visualização dos dados, o Grafana. Com ela é possível visualizar métricas de aplicações que estejam em execução na nuvem e que são registradas em um banco de dados. Esta ferramenta permite a criação de *dashboards* de maneira bastante flexível e conta com elevado poder de configuração de formatos de exibição. No caso deste sistema o Grafana é usado para plotar o comportamento das variáveis medidas no PDU e extraídas simultaneamente do Sensu, enquanto as aplicações Spark estão em execução na nuvem.

Ainda nessa arquitetura há agentes instalados nos servidores que são capazes de monitorar *logs* de execução, que ficam armazenados no Elastic Search que é um banco de dados mais adequado para *logs* e que trabalha associado ao Kibana, uma poderosa ferramenta de visualização e tratamento de *logs*.

Ao executar aplicações, é possível também rastrear variados detalhes da execução da tarefa, tais como tempo gasto em cada etapa, resultados intermediários de processamento, tudo por meio do *History Server* disponibilizado pelo próprio Spark. Desta forma cada vez que uma aplicação é disparada no *cluster* é possível a partir de seu *timestamp* buscar no banco de dados a energia que foi consumida durante sua execução. Uma vez que o PDU mede a potência ativa, e via monitoração a armazena no Influxdb, podemos determinar a quantidade de energia consumida para cada execução de uma determinada

aplicação.

3.2. Abordagem do problema

Ao executar aplicações no ambiente Spark, um considerável conjunto de parâmetros pode ser variado. Como é uma aplicação distribuída, Spark nos permite definir parâmetros de configuração para o mestre (também chamado de *driver*) assim como para seus escravos (normalmente conhecidos como executores), além de ser possível escolher a quantidade de partições que serão feitas sobre o conjunto de dados. Em nossos experimentos, um primeiro conjunto de testes nos permitiu definir a configuração de valores de *setup* para adequar o ambiente Spark ao processar dados massivos em duas tarefas de mineração de padrões, Twidd e Eclat quando executado sobre máquinas virtuais (VMs). Feito isso, nossos testes avaliaram sistematicamente o impacto de variar a disponibilidade de núcleos (*cores*) de processamento para diferentes quantidades de executores em duas políticas de ocupação física da nuvem. Os passos para isso:

- Foi estruturado um ambiente de testes Spark contando com o Yarn como escalonador de recursos para executar Twidd e Eclat sobre uma base de dados de 8 GB distribuídas em um sistema de arquivos HDFS nos nós de computação, sempre com fator de replicação de dados 3 formando a nuvem para experimentação. O suporte mínimo dos algoritmos foi o mesmo e definido em 0,02 para garantir um efeito de carga significativa sobre a nuvem.
- Foram testadas duas formas de ocupar progressivamente os nós de computação. O HDFS foi montado múltiplas vezes, acrescentando VM's na correspondência que agregasse 16 *cores* para cada novo experimento. Cada experimento citado executou cada uma das aplicações com replicação 10, o que é considerado um valor adequado a experimentos com pouca variação.
- Três configurações de VM's diferentes foram escolhidas para ocupar progressivamente o *cluster*, com objetivo de avaliar o impacto da granularidade da virtualização utilizada, tanto para a efetiva conclusão das tarefas quanto para seu consumo. Na primeira configuração de experimentos, foram disparadas V.M.'s de 4 *cores*. Na segunda, foram usadas VM's de 8 *cores* e na terceira e última VM's de 16 *cores* ocupavam completamente cada um dos servidores físicos.
- Duas políticas de ocupação dos recursos, tendo como foco o uso de *cores* de processamento disponíveis, foram avaliadas. A primeira usou uma estratégia gulosa, onde cada servidor de computação foi ocupado por um HDFS montado e os experimentos executados. A cada passo guloso seguinte outro servidor físico foi incluído e novamente os experimentos foram executados. Desse modo, sempre foram acrescidos 16 *cores* físicos a cada passo. Na segunda estratégia, de forma similar a um escalonamento Round-Robin, os acréscimos de *cores* para montagem e testes de experimentos foram feitos exatamente de forma ortogonal à estratégia anterior, já que acréscimos também de 16 *cores* físicos eram feitos só que sempre ocupando servidores físicos da forma mais distribuída.

Os resultados encontrados estão apresentados na próxima seção.

4. Testes e discussão de resultados

Em nosso trabalho, a avaliação de dois parâmetros foram foco: **consumo de energia** (medido em watthora) e **desempenho** ao executar as aplicações na forma de tempo para

concluir a tarefa (medido em segundos). Reforçamos que o consumo de energia possui uma relação atrelada ao tempo de execução da tarefa. Uma expectativa natural é que tarefas que sejam concluídas mais rapidamente apresentem tendência de consumir menos, uma vez que o consumo da estrutura sem carga é bem significativo. Seguindo a metodologia que foi apresentada na seção anterior obtivemos resultados que estão apresentados nos gráficos que compõem as Figuras 2 e 3.

Os gráficos mostram a execução dos dois algoritmos, Twidd e Eclat, avaliados em tempo de conclusão da tarefa em função da disponibilidade de *cores* e consumo ao executar a tarefa em função da mesma disponibilidade de *cores*. Como a quantidade de *cores* foi avaliada em saltos de 16 *cores* ocupando fisicamente os seis servidores de forma gulosa ou em Round Robin, temos em cada gráfico 5 séries que são executadas em 6 configurações diferentes, definindo os pontos apresentados. São 5 séries porque na terceira configuração de V.M.'s, com 16 *cores*, a ocupação da nuvem na política de Round Robin acaba se tornando a mesma da forma gulosa. Também com V.M.'s de 16 *cores* não permite que exista o primeiro ponto, uma vez que o *driver* do Spark necessitará de ao menos um executor, o que acaba ocupando pelo menos 32 *cores*. As linhas presentes em cada série do gráfico tem o objetivo apenas de facilitar a interpretação do mesmo. Todos os valores apresentados nos experimentos contam com uma confiança de 95% sobre as médias. Os valores absolutos das confianças encontrados nunca foram maiores que 4,7% de suas respectivas médias, sendo tipicamente bem inferiores. Para evitar excesso de informações nos gráficos, as confianças foram suprimidas evitando poluição visual.

De início, nos chama atenção que uma tarefa executada na mesma base de dados pode apresentar resultados de desempenho e consumo muito distintos ainda que utilizando da mesma estrutura física. Em valores, podemos afirmar a partir dos gráficos que executar Twidd distribuído pode demorar entre 600 segundos (16 *cores* em V.M.'s de 4 *cores* distribuídas de forma gulosa) e 110 segundos (96 *cores*, nuvem totalmente ocupada), conforme a quantidade de *cores* disponível. Para executar o Eclat distribuído, os tempos estão entre 1200 segundos (16 *cores* em V.M.'s de 4 *cores* distribuídas de forma gulosa) para o pior resultado e 123 segundos (48 *cores* em V.M.'s de 8 *cores* cada distribuídas de forma gulosa) no melhor caso, que são diferenças significativas, mas esperadas uma vez que se disponibilizou recursos de forma crescente para cada ensaio.

Por outro lado, no consumo de energia para executar as mesmas tarefas, podemos ter variações que vão de 14 wh (48 *cores* em V.M.'s de 8 *cores* distribuídas de forma gulosa) a 36 wh (32 *cores* em V.M.'s de 4 *cores* distribuídas de forma gulosa) para o Twidd. No Eclat as variações ficam entre 16 wh (16 *cores* em V.M.'s de 8 *cores* distribuídas em Round Robin) e 49 wh (48 *cores* em V.M.'s de 8 *cores* distribuídas de forma gulosa). Isso nos permite afirmar que não é uma tarefa trivial a decisão sobre qual forma de escalonar recursos em um ambiente virtualizado para processar dados massivos que apresente bom compromisso entre o consumo de energia e tempo de execução.

A observação dos gráficos mostra que o aumento de *cores* em muitos casos não traz ganhos significativos, então seria interessante achar soluções que encontrassem as configurações que representem maior economia em tempo ou consumo sem exigir muitos recursos, para não elevar muito o custo. De qualquer forma há relações entre o tempo e consumo que precisam ser avaliados posteriormente.

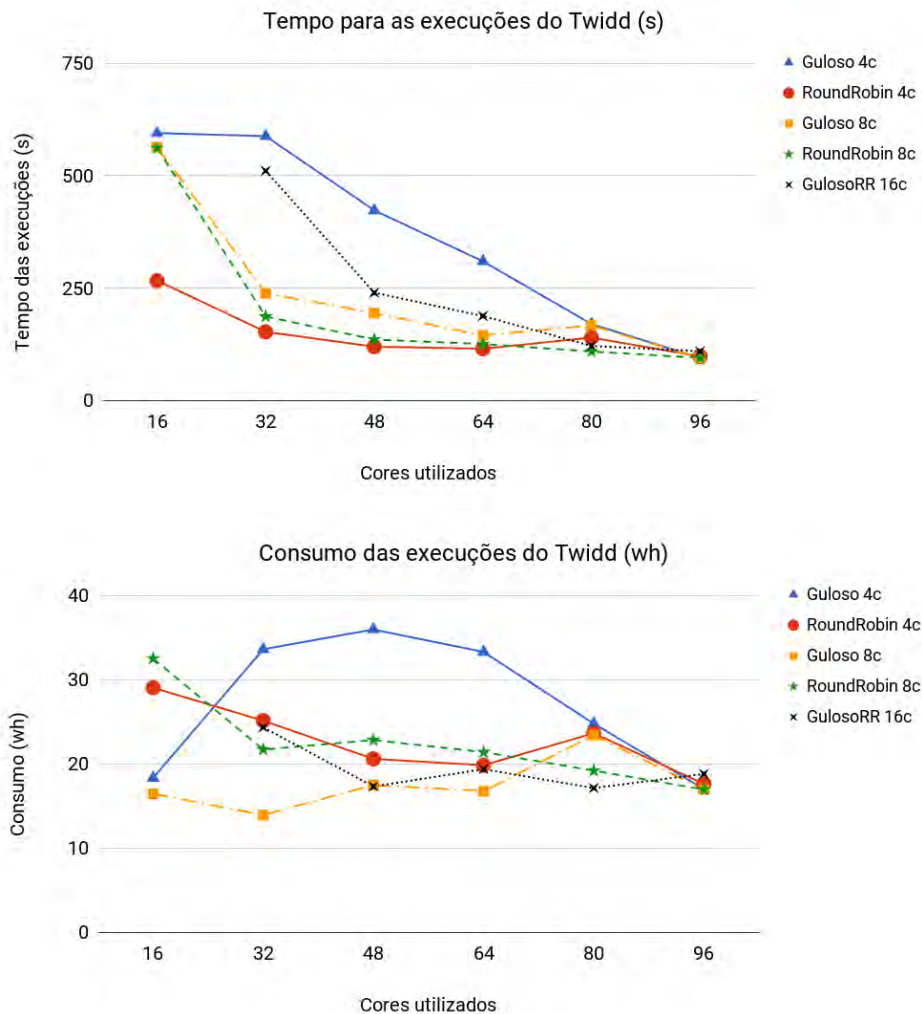


Figura 2. Gráficos do Twidd em função do tempo de execução e em função do consumo para variadas disponibilidades de *cores* em duas políticas de escalonamento: Gulosa e Round Robin

Do ponto de vista do desempenho, à medida que se oferece mais *cores* para a conclusão da tarefa, era esperado que o tempo de execução diminuísse. Nas curvas de tempo de execução para o Twidd e para o Eclat, na maioria das vezes o tempo diminui à medida que mais *cores* são oferecidos para a tarefa. Porém a forma como esse tempo diminui apresenta pouca inclinação e por vezes o tempo de execução volta a aumentar conforme pode ser observado para o Twidd entre os pontos de 64 e 80 *cores* para para V.M.'s com 8 *cores* na ocupação gulosa e também para V.M.'s de 4 *cores* em Round Robin. Para o Eclat o desempenho piora quando se oferece mais recurso apenas para V.M.'s de 4 *cores* em Round Robin entre os pontos de 64 para 80 *cores*. Essas alterações acabam por apresentar um reflexo também no consumo de energia para a tarefa, se observarmos a correspondência com os mesmos pontos nos gráficos de consumo.

Nuvens tipicamente seguem a política gulosa de ocupação das V.M.'s. Nos chama a atenção o fato de que V.M.'s de 4 *cores* que ocupam a nuvem na política Round Robin apresentam desempenho significativamente melhor, em ambos os algoritmos. Para o

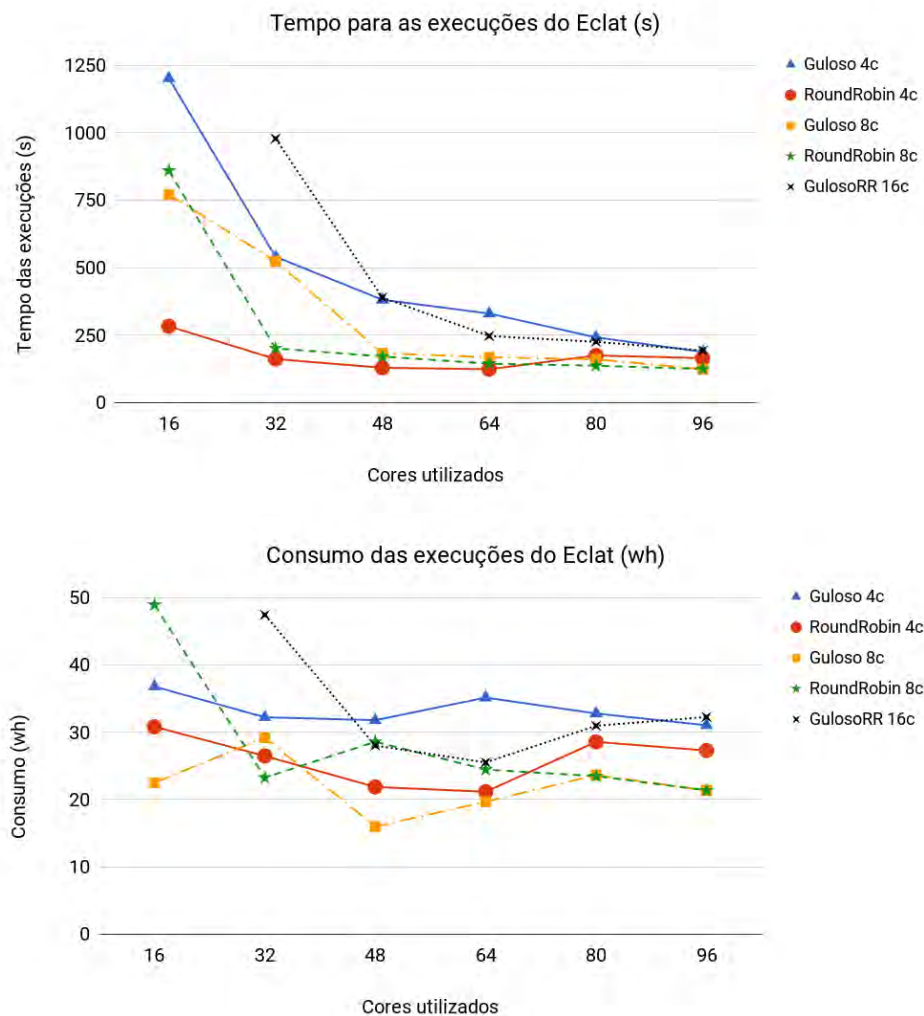


Figura 3. Gráficos do Eclat em função do tempo de execução e em função do consumo para variadas disponibilidades de cores em duas políticas de escalonamento: Gulosa e Round Robin

Twidd, o impacto disso é significativo mas para o Eclat que é um algoritmo mais simples, praticamente leva o consumo para valores bem inferiores, só não consumindo menos que alguns dos pontos em que a nuvem foi ocupada de forma gulosa por V.M.'s de 8 cores. Se considerarmos apenas V.M.'s de 4 cores para o Twidd, a diferença de consumo é inegavelmente favorável à ocupação em Round Robin. O uso de Round Robin é normalmente melhor em termos de tempo, pois espalha as máquinas distribuindo mais a carga.

Entretanto, é interessante notar também que a ocupação da nuvem por V.M.'s de 8 cores na estratégia gulosa alcançou os melhores resultados em consumo de energia, em praticamente todas as situações. Mesmo nas situações onde não foi a melhor opção, ainda esteve próxima dos melhores resultados.

Uma avaliação do início de todas as curvas também nos permite perceber que trabalhar com poucos recursos (cores), faz com que tenhamos um desempenho ruim seja para tempo de execução ou o consumo de energia. Configurações com poucos cores e

consequentemente menos memória, geram os piores tempos, pois é mais difícil conter o problema todo em memória. Isso é ainda pior para V.M.'s maiores porque perde-se mais *cores* na máquina *master* e porque um nó que pare por conta de *garbage collection* tem mais impacto, pois irá parar mais *cores* de uma vez. Justamente esse fato pode ser uma explicação para o fato de a política de ocupação com V.M.'s em Round Robin ser tão melhor: há menos contenção de recursos e quando algum nó para no *garbage collector*, o volume de recursos congelado é menor.

5. Conclusão e Trabalhos Futuros

Visando avaliar o consumo de energia ao processar dados massivos em ambientes virtualizados, estabelecemos um ambiente de nuvem amplamente monitorado para avaliarmos o impacto do escalonamento de recursos em função do tamanho, quantidade e políticas de ocupação física da estrutura. Nele foram realizados testes com dois algoritmos paralelos de execução distribuída que nos permitiram perceber que tanto o desempenho mas principalmente o consumo não possuem um comportamento simples para que se defina a melhor maneira de distribuir tarefas Spark gastando menos energia. Os experimentos realizados evidenciaram que há muito que ser estudado uma vez que o mesmo problema executado em configurações diferentes de recursos pode apresentar diferenças no consumo de energia que sejam mais que o dobro entre elas. Também foi possível perceber que a forma típica de ocupar um *datacenter* que é Gulosa, justamente para permitir o desligamento de servidores para economizar energia em cargas menores, nem sempre representa a melhor realidade, quando se trata de processar dados massivos em ambientes virtualizados.

Como trabalhos futuros, os testes realizados nos sugerem experimentar novos algoritmos de processamento de dados massivos com características que se diferenciem dos aqui utilizados, para que seja possível avaliar um comportamento mais completo das cargas típicas para esses ambientes. Outros virtualizadores além do KVM devem ser considerados para testes, para verificar se a validade do que encontramos aqui tem a mesma correspondência.

Dentro do objetivo de estudar o ambiente, também pode-se considerar variar a forma como se aproveita da localidade de dados via novas maneiras de estabelecer o HDFS, ou mesmo o uso de ambientes de armazenamento diferentes para verificar se é possível economizar energia. Também acreditamos ser interessante avaliar se é possível tirar maior proveito dos nós de computação pelo uso do Driver do Spark fora do HDFS, uma vez que pode desocupar mais uma V.M. para dedicá-la ao processamento da tarefa efetivamente, o que consequentemente deve ter impacto no consumo de energia.

Agradecimentos

Este trabalho foi parcialmente financiado por Fapemig, CAPES, CNPq, e pelos projetos MCT/CNPq-InWeb (573871/2008-6), FAPEMIG-PRONEX-MASWeb (APQ-01400-14), H2020-EUBR-2015 EUBra-BIGSEA e H2020-EUBR-2017 Atmosphere.

Referências

Abts, D., Marty, M. R., Wells, P. M., Klausler, P., and Liu, H. (2010). Energy proportional datacenter networks. *SIGARCH Comput. Archit. News*, 38(3):338–347.

- Andersen, D. G., Franklin, J., Kaminsky, M., Phanishayee, A., Tan, L., and Vasudevan, V. (2009). Fawn: A fast array of wimpy nodes. In *Proceedings of the ACM SIGOPS 22nd symposium on Operating systems principles*, pages 1–14. ACM.
- Berral, J. L., Goiri, Í., Nguyen, T. D., Gavalda, R., Torres, J., and Bianchini, R. (2014). Building green cloud services at low cost. In *Distributed Computing Systems (ICDCS), 2014 IEEE 34th International Conference on*, pages 449–460. IEEE.
- Conceição, V. S., Volpini, N. D. O., and Guedes, D. (2018). *Seshat: uma arquitetura de monitoração escalável para ambientes em nuvem*. In *Anais do XVII Workshop em Desempenho de Sistemas Computacionais e de Comunicação, Natal-RN. Sociedade Brasileira de Computação (SBC)*.
- Dias, V., Meira, W., and Guedes, D. (2016). Dynamic reconfiguration of data parallel programs. In *2016 28th International Symposium on Computer Architecture and High Performance Computing (SBAC-PAD)*, pages 190–197.
- Dupont, C., Giuliani, G., Hermenier, F., Schulze, T., and Somov, A. (2012). An energy aware framework for virtual machine placement in cloud federated data centres. In *Future Energy Systems: Where Energy, Computing and Communication Meet (e-Energy), 2012 Third International Conference on*, pages 1–10. IEEE.
- Ferro, M., Yokoyama, A., Klôh, V., Silva, G., Gandra, R., Bragança, R., Bulcao, A., Schulze, B., and SA-PETROBRAS, P. B. (2017). Analysis of gpu power consumption using internal sensors. In *Anais do XVI Workshop em Desempenho de Sistemas Computacionais e de Comunicação, Sao Paulo-SP. Sociedade Brasileira de Computação (SBC)*.
- Goiri, I. n., Katsak, W., Le, K., Nguyen, T. D., and Bianchini, R. (2013). Parasol and greenswitch: Managing datacenters powered by renewable energy. In *Proceedings of the Eighteenth International Conference on Architectural Support for Programming Languages and Operating Systems, ASPLOS '13*, pages 51–64, New York, NY, USA. ACM.
- Gu, X., Hou, R., Zhang, K., Zhang, L., and Wang, W. (2011). Application-driven energy-efficient architecture explorations for big data. In *Proceedings of the 1st Workshop on Architectures and Systems for Big Data*, pages 34–40. ACM.
- Hashem, I. A. T., Yaqoob, I., Anuar, N. B., Mokhtar, S., Gani, A., and Khan, S. U. (2015). The rise of “big data” on cloud computing: Review and open research issues. *Information Systems*, 47:98–115.
- Kansal, A., Zhao, F., Liu, J., Kothari, N., and Bhattacharya, A. A. (2010). Virtual machine power metering and provisioning. In *Proceedings of the 1st ACM symposium on Cloud computing*, pages 39–50. ACM.
- Kgil, T., D’Souza, S., Saidi, A., Binkert, N., Dreslinski, R., Mudge, T., Reinhardt, S., and Flautner, K. (2006). Picoserver: using 3d stacking technology to enable a compact energy efficient chip multiprocessor. *ACM SIGARCH Computer Architecture News*, 34(5):117–128.
- Kim, K. H., Beloglazov, A., and Buyya, R. (2009). Power-aware provisioning of cloud resources for real-time services. In *Proceedings of the 7th International Workshop on*

- Middleware for Grids, Clouds and e-Science*, MGC '09, pages 1:1–1:6, New York, NY, USA. ACM.
- Koomey, J. (2011). Growth in data center electricity use 2005 to 2010. *A report by Analytical Press, completed at the request of The New York Times*, page 9. Disponível em <http://www.analyticpress.com/datacenters.html>.
- Le, K., Bianchini, R., Zhang, J., Jaluria, Y., Meng, J., and Nguyen, T. D. (2011). Reducing electricity cost through virtual machine placement in high performance computing clouds. In *Proceedings of 2011 International Conference for High Performance Computing, Networking, Storage and Analysis*, page 22. ACM.
- Leverich, J. and Kozyrakis, C. (2010). On the energy (in) efficiency of hadoop clusters. *ACM SIGOPS Operating Systems Review*, 44(1):61–65.
- Mashayekhy, L., Nejad, M. M., Grosu, D., Lu, D., and Shi, W. (2014). Energy-aware scheduling of mapreduce jobs. In *Big Data (BigData Congress), 2014 IEEE International Congress on*, pages 32–39. IEEE.
- Ramanathan, R. M. (2006). Intel® multi-core processors. <http://www.intel.com/technology/archtecture/downloads/quad-core-06.pdf>.
- Sahoo, J., Mohapatra, S., and Lath, R. (2010). Virtualization: A survey on concepts, taxonomy and associated security issues. In *Computer and Network Technology (ICCNT), 2010 Second International Conference on*, pages 222–226. IEEE.
- Shehabi, A., Smith, S. J., Sartor, D. A., Brown, R. E., Herrlin, M., Koomey, J. G., Masanet, E. R., Horner, N., Azevedo, I. L., and Lintner, W. (2016). United states data center energy usage report.
- Whitney, J. and Delforge, P. (2014). Scaling up energy efficiency across the data center industry: Evaluating key drivers and barriers. <https://www.nrdc.org/energy/files/data-center-efficiency-assessment-IP.pdf>. Acessado em maio de 2015.
- Wirtz, T. and Ge, R. (2011). Improving mapreduce energy efficiency for computation intensive workloads. In *Green Computing Conference and Workshops (IGCC), 2011 International*, pages 1–8. IEEE.
- Ye, K., Huang, D., Jiang, X., Chen, H., and Wu, S. (2010). Virtual machine based energy-efficient data center architecture for cloud computing: a performance perspective. In *Proceedings of the 2010 IEEE/ACM Int'l Conference on Green Computing and Communications & Int'l Conference on Cyber, Physical and Social Computing*, pages 171–178. IEEE Computer Society.
- Zhang, Q., Cheng, L., and Boutaba, R. (2010). Cloud computing: state-of-the-art and research challenges. *Journal of Internet Services and Applications*, 1(1):7–18.

Uma Análise Competitiva entre as Tecnologias *Blockchain* e *Tangle* para o Projeto de Aplicações IoT

Rafael Zerbini Alves da Mata¹, Carlo Kleber da Silva Rodrigues²

¹Faculdade de Tecnologia – Departamento de Engenharia Elétrica – Universidade de Brasília (UnB) – Brasília – DF – Brasil.

²Centro de Matemática, Computação e Cognição – Universidade Federal do ABC (UFABC) – Santo André – SP – Brasil.

{rafael.zerbinia, carlokleber}@gmail.com

Abstract. *This paper aims at performing a competitive analysis between the Blockchain and Tangle technologies, mainly focusing on the design of IoT applications. To this end, we firstly carry out a theoretical comparative study of the data structures and algorithms used to manipulate information under each technology. Then, through simulations, we analyze different scenarios to assess the system security level in terms of data inviolability. Final results allow us to conjecture that Tangle is a more promising choice, besides constituting worthy insights for real designs of IoT applications. General conclusions and future work conclude this article.*

Resumo. *Este artigo tem por objetivo realizar uma análise competitiva entre as tecnologias Blockchain e Tangle, visando principalmente ao projeto de aplicações IoT. Para tanto, inicialmente é feito um estudo teórico comparativo das estruturas de dados e algoritmos usados para manipular informações sob cada tecnologia. Em seguida, por meio de simulação, diferentes cenários são analisados para aferir o nível de segurança do sistema em termos da inviolabilidade das informações, considerando o emprego de cada tecnologia. Os resultados finais permitem conjecturar que Tangle é uma alternativa mais promissora, além de se constituírem em subsídios para possíveis projetos de aplicações IoT. Conclusões gerais e trabalhos futuros encerram este artigo.*

1. Introdução

A Internet das Coisas (do inglês, *Internet of Things* – IoT) possui diversas aplicações, incluindo redes inteligentes, cidades inteligentes [Gaur et al. 2015], contratos inteligentes e gestão de saúde [Hassanalieragh et al. 2015]. Neste cenário, a rede de comunicação subjacente deve oferecer processamento e disseminação de dados de maneira transparente, intensa e pervasiva, o que traz preocupações relacionadas a aspectos de segurança, privacidade e escalabilidade [Dorri; Kanhere; e Jurdak 2017].

Considere, por exemplo, uma operação de crédito entre dispositivos IoT regulada por um contrato inteligente em uma rede de comunicação *peer-to-peer* (P2P). Os nós dos dispositivos IoT armazenam cópias consistentes do contrato inteligente. Um dos dispositivos pode solicitar a atualização do estado do contrato para atribuir crédito a um outro dispositivo para prestação de um serviço. Os nós constituintes da rede são

informados dessa atualização quando recebem individualmente o novo estado do contrato. Eventualmente, o dispositivo IoT destinatário do crédito verifica esse estado em seu nó e, em seguida, libera o serviço para o dispositivo solicitante.

Esse processo permite a troca de crédito na ausência de confiança mútua, uma vez que a transação é certificada por um terceiro, ou seja, a rede de nós, e é dificilmente reversível. O objetivo desse processo é, sobretudo, garantir que cada dispositivo observe o mesmo estado do contrato inteligente. A questão crucial de projeto é então fazer com que cada dispositivo permaneça sincronizado com a versão mais recente do contrato, além de garantir segurança, privacidade e escalabilidade.

Independentemente da natureza da aplicação IoT, a solução da questão ilustrada acima reside na escolha da tecnologia para manipular as informações das transações entre os dispositivos comunicantes. Nesse sentido, *Blockchain* [Nakamoto 2008] e *Tangle* [Popov 2017] são duas importantes tecnologias merecedoras de atenção na literatura [Dorri; Kanhere; e Jurdak 2017; Danzi et. al. 2017].

Blockchain se baseia na implementação de uma *lista encadeada* de blocos de transações (do inglês, *blockchain*). Esses blocos são adicionados à lista conforme são validados por meio de um processo matemático. Por sua vez, *Tangle* se baseia na implementação de um *Grafo Acíclico Direcionado* (do inglês, *Directed Acyclic Graph – DAG*), onde cada nó é uma transação. Para que a transação seja adicionada ao DAG, há um processo matemático, como na *Blockchain*, e ainda ocorre a validação de duas outras transações pré-existentes no DAG.

Este contexto é a motivação para este artigo, cujo objetivo é realizar uma análise competitiva entre as tecnologias *Blockchain* e *Tangle*, visando ao desenvolvimento de possíveis projetos de aplicações IoT. Para este fim, é feito inicialmente um estudo teórico das estruturas de dados e algoritmos usados para manipular informações sob cada tecnologia, buscando avaliar as complexidades de tempo com respeito a operações de *consulta* e *adição* de transações. Em seguida, por meio de simulação, diferentes cenários de operação são analisados para mensurar o nível de segurança sistêmica, em termos da inviolabilidade das informações registradas.

Ante o objetivo anunciado, este artigo tem, como principal contribuição, a disponibilização de um único texto que provê um entendimento comparativo das tecnologias *Blockchain* e *Tangle* e, ainda, oferece uma análise competitiva entre as mesmas. Essa contribuição se constitui valiosa por prover subsídios relevantes para o desenvolvimento de possíveis projetos de aplicações IoT.

A organização do restante deste artigo é descrita a seguir. A Seção 2 apresenta as tecnologias *Blockchain* e *Tangle*. Os trabalhos relacionados são comentados na Seção 3. Na Seção 4, é realizado o estudo teórico das estruturas de dados e algoritmos utilizados nas tecnologias supracitadas. A Seção 5 analisa a segurança do sistema, com foco na garantia da inviolabilidade das informações. Por fim, conclusões gerais e trabalhos futuros aparecem na Seção 6.

2. Fundamentos

2.1 Tecnologia *Blockchain*

Como mencionado, as transações dos clientes são agrupadas em blocos interligados formando uma *lista encadeada*, i.e., a *blockchain*. Cada bloco está ligado a apenas um bloco anterior a ele. *Mineração* é o processo matemático que precisa ser executado para que o bloco seja adicionado à *blockchain*. Esse processo é executado por um *peer*, denominado de *minerador*, ou por um grupo de *peers*, denominado de *mining pool* [Garay; Kiayias; e Leonardos 2015; Silva e Rodrigues 2016].

Os *mineradores* ou *mining pools* são também responsáveis por coletar as transações transmitidas em *broadcast* na rede e agrupá-las em blocos. Cada vez que um bloco é *minerado*, o correspondente *minerador* ou *mining pool* recebe uma recompensa, que é um incentivo financeiro para que a *mineração* seja continuamente realizada.

Na prática, a *mineração* consiste em tentativas sucessivas para determinar um valor de *nonce* que resolve o *hash* criptográfico do bloco de transações, atendendo a um certo critério. A dificuldade desse critério é ajustada com base na frequência com que os blocos são adicionados à *blockchain*. A determinação do *nonce* é a chamada *prova de trabalho* (do inglês, *proof of work*) de que o bloco foi verificado e pode ser adicionado. O algoritmo de *hash* criptográfico utilizado é o SHA256 [Gilbert e Handschuh 2004].

Além das transações, cada bloco possui um cabeçalho com *metadados* [Antonopoulos 2017]. A identificação de cada bloco é feita pelo *hash* criptográfico de seu cabeçalho. Como informado, cada bloco faz referência a apenas um bloco anterior, chamado de bloco *pai*. Essa referência é feita com o *hash* do cabeçalho do bloco *pai*. Ou seja, cada bloco contém o *hash* do cabeçalho de seu *pai* dentro de seu próprio cabeçalho.

A sequência de *hashes* que liga cada bloco ao seu *pai* cria uma cadeia de blocos que faz o caminho de volta até o primeiro bloco do sistema, denominado *bloco gênese*. A mudança da identidade de um bloco *B* geraria um efeito cascata da mudança da identidade de todos os blocos subsequentes a ele. Esse efeito cascata constitui a base da segurança do sistema com relação à inviolabilidade das informações registradas, conforme explorado mais adiante na Seção 5.

A *Blockchain* teve sua origem em 2008, junto com a proposta da criptomoeda *Bitcoin* [Nakamoto 2008]. Apesar da concepção original para pagamentos eletrônicos, a *Blockchain* foi rapidamente identificada como de aplicabilidade mais abrangente [Dorri; Kanhere e Jurdak 2017; Huckle et al. 2016]. Por exemplo, na implementação de aplicações de controle de votos, de recursos em gestão orçamentária, de direitos de propriedades e, ainda, de comunicação em sistemas distribuídos, incluindo especialmente IoT [Dorri; Kanhere; e Jurdak 2017; Li e Liao 2017].

2.2 Tecnologia *Tangle*

A tecnologia *Tangle* surgiu para implementação da criptomoeda *iota* [IOTA 2017], então projetada em 2014 para a indústria de IoT. Sua aplicabilidade, no entanto, não se restringe a pagamentos eletrônicos, mas envolve um escopo bem mais abrangente, tal como ocorre com a tecnologia *Blockchain*.

Como já mencionado, a tecnologia *Tangle* utiliza a estrutura DAG para armazenamento das transações. A função de *hash* criptográfico originalmente utilizada nessa tecnologia foi a Curl, tendo sido substituída mais tarde pela Keccak-384 [Sonstebo 2017], também conhecida como SHA-3 (*Secure Hash Algorithm 3*) [Chandrana e Manuelb 2016]. Os defensores dessa tecnologia professam que ela é a evolução da *Blockchain*, levando-se em conta os três aspectos comentados a seguir.

O primeiro aspecto diz respeito a não existência de recompensas na *Tangle*. Na *Blockchain*, a recompensa pode inviabilizar a execução de transações típicas de IoT. Isso porque essas transações tendem a ser de pequenos valores. Daí, a existência de uma recompensa de valor maior que o montante que está sendo transferido na transação não seria lógico. Simplesmente eliminar a recompensa na *Blockchain* tampouco seria aceitável, pois ela é o incentivo para a contínua *mineração* de blocos.

O segundo aspecto relaciona-se à homogeneidade funcional dos dispositivos IoT na tecnologia *Tangle*. Isto é, todos os participantes desempenham a mesma função. Isso não é observado na tecnologia *Blockchain*, pois há dois tipos distintos de participantes: aqueles que emitem transações e aqueles que aprovam transações.

Essa discriminação entre dispositivos pode levar a conflitos na *mineração* de blocos em virtude das recompensas previstas. Nesse caso, uma disputa entre *mineradores* ou *mining pools* pode transformar um paradigma de validação concebido originalmente descentralizado em um paradigma centralizado, constituído por *mineradores* ou *mining pools* de maior capacidade de processamento.

O terceiro aspecto refere-se à maior simplicidade da *Tangle* para *adição* de novas transações ao sistema. Essa *adição* envolve a resolução de uma *prova de trabalho* e a validação de duas transações pré-existentes. Ocorre que a *prova de trabalho* da *Tangle*, por considerar apenas uma transação, demanda um esforço computacional bem menor que àquele da *mineração* na *Blockchain*, que envolve múltiplas transações. Por sua vez, a validação de uma transação pré-existente é simples e pode ser realizada em tempo constante. A *adição* de transações é detalhadamente analisada na Subseção 4.2.

Para terminar esta subseção, cabe mencionar que as validações são representadas por arestas direcionadas no DAG, que partem da nova transação em direção às transações pré-existentes. Se não existe uma aresta direcionada entre uma transação *A* e uma transação *B*, mas há um caminho direcionado entre elas, diz-se então que *A* indiretamente aprova *B*. Quanto maior é o número de validações que uma transação possui, mais confiável é a validade dessa transação. Por definição, há ainda a transação denominada *gênese*, que representa a primeira transação do sistema e é validada direta ou indiretamente por todas as outras transações do DAG.

3. Trabalhos Relacionados

A tecnologia *Blockchain* possui um considerável número de trabalhos na literatura. O mesmo não pode ser afirmado para tecnologia *Tangle*, a qual é mais alvo de *blogs* e de outros veículos informais. Estudos comparativos entre essas duas tecnologias tampouco são fáceis de encontrar. Neste contexto, esta seção discorre sobre alguns dos mais recentes e relevantes trabalhos da literatura que contribuem para o objetivo deste artigo, buscando ofertar uma visão do estado da arte desta área de pesquisa.

Opara e Soluade (2015) e Singh (2014) analisam ataques cibernéticos a plataformas baseadas na tecnologia *Blockchain*. Busca-se comprovar a inadequação das infraestruturas de redes legadas, identificar e caracterizar os atores das ameaças, suas capacidades, motivações e propósitos. São também apresentadas as melhores práticas para enfrentar essa realidade e informados cenários de aplicabilidade dessa tecnologia.

Eyal e Sirer (2013) e Courtois e Bahack (2014) estudam a vulnerabilidade da *Blockchain* sob a condição de ataques de *mineradores desonestos*, os quais conseguem receber recompensas maiores que aquelas de *mineradores honestos*. Usando modelagem analítica e simulação, são alcançadas evidências de que, sem regulações da *mineração* por *mining pools*, o sistema pode deixar de ser verificável de maneira descentralizada.

Karame, Androulaki e Capkun (2012), Rosenfeld (2014) e Gervais et al. (2016) estudam ataques de *double-spending* em sistemas baseados na *Blockchain*. Esses ataques objetivam permitir que um mesmo recurso seja usado em diferentes transações. A partir de modelagem analítica e simulação, são obtidos resultados que permitem inferir a probabilidade de ocorrência desses ataques e propor contramedidas de defesa.

Sompolinsky e Zohar (2015) e Karame (2016) discutem sobre a escalabilidade da *Blockchain*, considerando a quantidade de informações, o tempo de verificação de transações, a otimização das estruturas de dados e dos algoritmos, o tamanho dos blocos da cadeia, a diversidade de aplicações, dentre outros aspectos. Os resultados permitem conjecturar sobre uma nova geração de aplicações alicerçadas pela *Blockchain*.

Dorri, Kanhere e Jurdak (2017) propõem uma versão mais otimizada da *Blockchain* para o desenvolvimento de aplicações IoT. Essa versão tenciona diminuir o custo de processamento criptográfico, otimizar a banda e diminuir as latências de transmissão. Emprega-se um paradigma de camadas funcionais, permitindo que bases de dados locais centralizadas se comuniquem com bases públicas distribuídas. Os blocos são *minerados* localmente e adicionados à *blockchain* de sua camada, que posteriormente se torna consistente com a *blockchain* de camadas públicas superiores.

Danzi et al. (2017) descrevem arquiteturas gerais e protocolos de sincronização para nós de clientes IoT usando tecnologia *Blockchain*. Consideram-se diferentes capacidades de transmissão da rede, níveis de segurança e características intrínsecas de ambientes *wireless*. Nesse trabalho são modelados e caracterizados analiticamente os tráfegos gerados pelos protocolos de sincronização e, usando simulações, são avaliados o consumo de energia e o custo de banda para sincronização da *blockchain*.

Rodrigues (2017b) analisa a eficiência e a segurança da tecnologia *Blockchain*. Usando modelagem analítica e simulação, são realizados experimentos em diferentes cenários de operação. Os resultados apontam para uma promissora eficiência na manipulação de dados e uma atrativa segurança quanto à inviolabilidade da informação.

Li e Liao (2017) apresentam o protocolo GHOST, o qual propõe a utilização de uma estrutura de dados baseada em árvores em substituição à *blockchain*. Utilizando principalmente modelagem analítica, os autores verificam a possibilidade da redução do tempo de confirmação das transações e a otimização do nível de segurança sistêmica. Esse trabalho pode ser visto como um dos precursores da tecnologia *Tangle*.

Além do protocolo GHOST, há ainda algumas outras propostas que podem ser também consideradas precursoras da tecnologia *Tangle* como, por exemplo, os trabalhos

de Lerner (2015), Sompolinsky e Zohar (2013) e Lewenberg, Sompolinsky e Zohar (2015). Este último, em especial, propõe um modelo de criptomoeda também baseada em DAG. No entanto, o modelo difere da *Tangle* nos seguintes dois pontos principais: as unidades de informação são blocos de dados e não transações individuais e, ainda, o conceito de recompensa para validação de transações é aplicado.

Por fim, Popov (2017) analisa os fundamentos teóricos matemáticos da criptomoeda *iota*. Esses fundamentos constituem a descrição e a explicação da tecnologia *Tangle*, incluindo discussões sobre o número de validações que as transações devem possuir para sua aceitação, a estabilidade do sistema, as possibilidades de ataques e vulnerabilidades, e os algoritmos de seleção de transações pré-existentes. Em que pese a existência apenas de abordagem teórica, esse trabalho constitui uma das mais importantes referências da literatura sobre a análise da tecnologia *Tangle*.

Ante o exposto, a contribuição e a diferenciação deste artigo se revelam pelo relativo ineditismo de uma análise competitiva entre as tecnologias *Blockchain* e *Tangle* com foco em IoT, contemplando a eficiência das estruturas de dados e algoritmos, sob o viés teórico da complexidade de tempo, além da avaliação da segurança quanto à inviolabilidade das informações, sob o viés da simulação.

4. Eficiência: Complexidade de Tempo

Esta seção avalia a eficiência das tecnologias *Blockchain* e *Tangle*, considerando a seguinte questão de pesquisa: quais as complexidades de tempo para a execução de operações de *adição* de transações (ou blocos) e *consulta* de transações?

4.1. Tecnologia *Blockchain*

Cada bloco *minerado* é propagado em *broadcast* na rede. Em seguida, aguarda-se que *nós completos*, *peers* da rede que mantêm uma cópia local de toda a cadeia de blocos, realizem a sua *adição*. A cópia da cadeia de blocos local de cada *nó completo* é atualizada à medida que novos blocos são *minerados* e propagados. A convergência das cópias locais dos *nós completos* ocorre conforme as informações são disseminadas.

Para *adição* de um novo bloco, B_{novo} , um *nó completo*, P , inicialmente examina o cabeçalho desse bloco para conhecer o valor do campo *hash do bloco anterior*, que é a referência para o bloco *pai*. Seja $h(B_{anterior})$ este valor. Admita que P possui m blocos na sua cadeia local, sendo B_m o último bloco adicionado, e $h(B_m)$ o *hash* do seu cabeçalho.

O nó P então adiciona B_{novo} à sua cadeia somente se $h(B_m) = h(B_{anterior})$ e se a *prova de trabalho* tiver sido realizada, ou seja, se o valor do *nonce* tiver sido determinado. Embora a computação do valor de *nonce* seja intrincada, a sua verificação é de complexidade constante, pois basta que o valor de *nonce* seja testado. Assim, a complexidade de tempo de *adição* de um bloco é de apenas $O(1)$. No entanto, é preciso ressaltar que, para que a *adição* ocorra, é necessário que o bloco já tenha sido *minerado*. Ou seja, existe um custo computacional anterior que, na prática, não pode ser ignorado.

Para *consultar* sobre a existência de uma transação em um bloco no sistema são usados nós especiais da rede, chamados de nós de Verificação Simplificada de Transação (VST). Esses nós utilizam o chamado *caminho de Merkle* ou *caminho de autenticação* [Antonopoulos 2017], conforme explicado a seguir.

Seja Q um nó VST que deseja consultar a existência da transação s , tendo como uma das partes envolvidas um cliente de endereço lógico E . Esse nó Q deve então criar um filtro em suas conexões com os demais *peers* da rede para restringir as transações a ele informadas, ou seja, apenas blocos com transações que se relacionam ao endereço E devem ser informados a ele. Assim, quando um *peer* da rede vê um bloco com uma transação de endereço E , ele então informa sobre aquele bloco ao nó Q . Para tanto, ele envia uma mensagem especial, denominada *mensagem de bloco de Merkle* (MBM). Essa mensagem contém o *cabeçalho do bloco* e o *caminho de Merkle* que liga a transação s à *raiz da árvore de Merkle* do bloco.

A *árvore de Merkle* [Berman; Karpinski; e Nekrich 2007], conhecida como árvore binária de *hash*, é construída a partir dos seus vértices *folhas*. Cada *folha* guarda o *hash* de uma das transações existentes no bloco. São então computados recursivamente os *hashes* de pares de vértices de mesmo nível (i.e., vértices *irmãos*), a partir dos vértices *folhas* em direção à raiz, até que se obtenha um único *hash*, o qual constitui o *resumo* de todas as transações existentes no bloco. Esse *resumo* é guardado em um campo do cabeçalho do bloco, denominado *raiz da árvore de Merkle*. No caso de o número de transações ser ímpar, a última transação é duplicada, obtendo sempre uma *árvore binária cheia* [Cormen; Leiserson; e Rivest, 2009].

Seja então a sequência de vértices $(v_1, v_2, \dots, v_l)$ o *caminho de Merkle* informado ao nó Q . Cada vértice da sequência está em um nível imediatamente inferior ao anterior na *árvore de Merkle*. Mais especificamente, o vértice v_1 é uma *folha* e o vértice v_l está no nível imediatamente superior ao nível da raiz da árvore, que está no nível 1. Além disso, o vértice v_1 é *irmão* do vértice *folha* que armazena o *hash* da transação s .

A *consulta* da transação s consiste então em computar recursivamente o *hash* de vértices *irmãos* correspondentes na árvore em diferentes níveis sucessivos, a partir do vértice *folha* v_1 em direção à raiz. Ao atingir-se o segundo nível, é então verificado se o *hash* computado é igual ao valor da raiz, obtido do exame do *cabeçalho do bloco*, que é também informado na MBM. Se ocorre a identidade, então a transação s de fato existe.

Essa operação de *consulta* tem complexidade de tempo $O(\lceil \log_2(2n - 1) \rceil)$, onde n é o número total de transações existentes no bloco. Esse resultado é bastante eficiente por ser logarítmico em função da quantidade de dados. Isso ocorre porque a *árvore de Merkle* construída é sempre uma *árvore binária cheia* e, portanto, goza da propriedade de altura logarítmica [Cormen; Leiserson; e Rivest 2009].

4.2. Tecnologia Tangle

Como mencionado, cada nó do DAG armazena uma única transação. Considerando nós VST, a *consulta* de uma transação s pode então ser realizada em tempo constante $O(1)$, dado que se saiba qual o nó que armazena a transação s . Em termos de complexidade de tempo, tem-se então que *Tangle* é mais eficiente que *Blockchain* para realizar *consultas*, pois a complexidade desta última é maior, sendo igual a $O(\lceil \log_2(2n - 1) \rceil)$, onde n é o número total de transações existentes no bloco (vide Subseção 4.1).

Com relação à *adição* de uma transação s na *Tangle*, tem-se a análise baseada nos dois pontos a seguir. Primeiro, é preciso realizar um processamento matemático para resolução de um *hash* criptográfico, i.e., obtenção do valor *nonce*. Esse *hash*

criptográfico envolve metadados relacionados à identificação da transação s , além da transação s propriamente dita. Seja C_T a complexidade dessa operação.

Segundo, é preciso validar duas transações pré-existentes no sistema. O processo de validação é simples, consistindo apenas em verificar se há conflitos entre as transações e se a *prova de trabalho* foi realizada pelo respectivo nó da transação a ser validada. Essa verificação pode ser feita em $O(1)$. Resulta então que a complexidade final de *adição* é $C_T + O(1) = C_T$, ou seja, é determinada pela complexidade da realização da *prova de trabalho*.

Para fins de comparação, tem-se que a *adição* de um bloco é feita em $O(1)$ na tecnologia *Blockchain*. Na prática há, contudo, a necessidade da *mineração* prévia do bloco a ser adicionado (vide Subseção 4.1). Seja o custo dessa *mineração* igual a C_B . O custo final de *adição* na *Blockchain* é, portanto, $C_B + O(1) = C_B$.

Tendo em vista que a *mineração* de um bloco na *Blockchain* envolve um *hash* criptográfico com metadados relacionados à identificação de todo o bloco, além de todas as transações nele existentes, tem-se que $C_B > C_T$. Ou seja, a tecnologia *Tangle* resulta mais eficiente que a tecnologia *Blockchain* quanto à complexidade de tempo da *adição*. Essa conclusão é comparativa e independe dos valores absolutos de C_B e C_T .

Como observação final, deve-se mencionar que cada *adição* na *Blockchain* considera n transações de uma só vez, e que cada *adição* na *Tangle* se refere a apenas uma única transação. Mesmo assim, é possível conjecturar que $C_B/n > C_T$ em virtude da natureza do cálculo de *hash* envolvido [Cormen; Leiserson; e Rivest 2009].

5. Segurança: Inviolabilidade da Informação

Esta seção avalia a segurança das tecnologias *Blockchain* e *Tangle*, considerando a seguinte questão de pesquisa: qual o tempo de registro de uma transação no sistema para garantir um nível de segurança aceitável, significando inviolabilidade da informação?

5.1. Descrição dos Cenários de Investigação

Para entender a questão de pesquisa na *Blockchain*, considere o cenário descrito a seguir [Nakamoto 2008]. Quando dois *mineradores* distintos enviam diferentes versões daquele que seria o próximo bloco da *blockchain*, os nós *completos* vão receber uma ou outra versão primeiramente. Cada nó *completo* considera a primeira versão recebida, mas vai também criar uma ramificação em sua *blockchain* local para adicionar a outra versão do bloco recebida posteriormente. Passam a existir duas ramificações: uma para a primeira versão e a outra para a segunda versão.

A seleção entre as duas ramificações ocorre quando as próximas *provas de trabalho* são encontradas, fazendo com que uma das ramificações se torne mais longa que a outra. Neste instante, a ramificação mais curta é desprezada. Para ser considerada válida, uma transação deve então pertencer à ramificação mais longa.

A competição entre duas ramificações pode ser modelada como uma *corrida* para *adição* de blocos à *blockchain*, como explicado a seguir [Nakamoto 2008]. Um *minerador desonesto* deseja realizar uma fraude, substituindo um bloco B legítimo adicionado por um *minerador honesto*. Para isso, o *minerador desonesto* espera até que z blocos subsequentes a B sejam adicionados à *blockchain*. Neste instante, o *minerador*

honesto está *minerando* o bloco $z+1$. Explica-se que essa espera de z blocos é o tempo que o cliente da aplicação tolera esperar para ter sua transação confirmada no sistema.

O *minerador desonesto* então inicia a *mineração* explícita de blocos e tenta ser mais rápido que o *minerador honesto* para compensar a desvantagem de z blocos, fazendo sua ramificação se tornar mais longa que a ramificação contendo o bloco legítimo B . Um detalhe importante é que o *minerador desonesto* *minera* blocos secretamente em taxa compatível com sua capacidade para que, ao iniciar a competição explícita com o *minerador honesto*, já possua blocos *minerados* para inserir no sistema.

Iniciada a competição, tem-se o seguinte modelo parametrizado [Nakamoto 2008]: com probabilidade p , o próximo bloco *minerado* da *blockchain* é do *minerador honesto*, e com probabilidade $q = (1 - p)$, o próximo bloco *minerado* é do *minerador desonesto*. A chegada de blocos *minerados* à *blockchain* segue um processo de Poisson com parâmetro μ . Esse modelo permite estimar a probabilidade de fraude para um dado valor de z . Quanto maior é z , menor é a probabilidade de fraude e maior é o tempo de espera (i.e., tempo de registro da transação). Ou seja, há um compromisso entre a segurança e o tempo de espera. Esse modelo é resolvido por simulação na Subseção 5.2.

O cenário de investigação da *Tangle* é análogo ao da *Blockchain* [Popov 2017]. A competição, porém, é pelo número de validações da transação. O nó fraudador deseja substituir uma transação legítima T do DAG. Ele então espera até que a transação T atinja v validações, que reflete o tempo que o cliente da aplicação tolera esperar para ter a transação confirmada. Em seguida, ele insere uma transação fraudulenta F no DAG para tentar substituir a transação T . Para isso, ele passa a criar novas transações que só validam a transação F . O objetivo é que, eventualmente, a transação F passe a ter mais validações que a transação T . Como na *Blockchain*, o fraudador na *Tangle* trabalha secretamente em taxa compatível com sua capacidade para que, ao iniciar a competição explícita, ele já possua transações prontas para inserir no DAG.

O modelo parametrizado a seguir é usado para análise da *Tangle* [Popov 2017]: a chegada de transações ao DAG segue um processo de Poisson com parâmetro β ; a fração de transações que valida a transação legítima T é $k\%$ do total de transações, e a fração de transações que valida a transação fraudulenta F é $r\%$ do total de transações, sendo $k + r = 100$. Esse modelo permite calcular a probabilidade de fraude para um dado valor de v . Quanto maior é v , menor é a probabilidade de fraude e maior é o tempo de espera. Ou seja, como na *Blockchain*, há um compromisso entre a segurança e o tempo de espera. Esse modelo é resolvido por simulação na Subseção 5.2.

5.2. Ambiente de Simulação e Modelos

A simulação usa o Tangram-II [De Souza e Silva; Figueiredo; e Leão 2009]. Tangram-II é um ambiente de modelagem de sistemas computacionais concebido na Universidade Federal do Rio de Janeiro, com a participação da Universidade da Califórnia em Los Angeles nos EUA. Os modelos são definidos em termos de objetos que interagem entre si por mensagens, podendo ser resolvidos analiticamente ou via simulação.

O modelo de simulação da *Blockchain* é explicado a seguir. São criados dois objetos: Obj_1 e Obj_2 . O objeto Obj_1 emula os *mineradores* (*honestos* e *desonestos*) do sistema, os quais têm individualmente a mesma capacidade de *mineração*. O processo de Poisson tem seu parâmetro μ igual a um bloco *minerado* a cada 10 minutos,

conforme definição da *Blockchain* [Rodrigues 2017a]. O objeto *Obj₂* emula a *blockchain* do sistema, permitindo monitorar a diferença de blocos entre as ramificações *honestas* e *desonestas*. O sistema é considerado no estado estacionário.

O modelo de simulação da tecnologia *Tangle* é explicado a seguir. São criados três objetos: *Obj₁*, *Obj₂* e *Obj₃*. Os dois primeiros objetos emulam os nós *honestos* e *desonestos*, respectivamente, os quais têm individualmente a mesma capacidade de processamento. Os nós *honestos* coletivamente contribuem com $k\%$ das transações totais do sistema, e os *desonestos* com $r\%$ das transações totais, sendo $k + r = 100$. O processo de Poisson tem seu parâmetro β igual a 500 transações a cada 10 minutos. Esse valor permite uma comparação mais equânime com a tecnologia *Blockchain*, pois o número médio de transações por bloco é 500 [Antonopoulos 2017]. O objeto *Obj₃* emula o DAG, permitindo monitorar a diferença de validações entre as transações *honestas* e *desonestas*. O sistema é considerado no estado estacionário.

Cabe destacar que os resultados de simulação têm intervalos de confiança de 95% que estão dentro do limite de 5% dos valores estimados, tendo sido consideradas 30 execuções (rodadas) com um tempo de simulação de 21.000 minutos cada. Estes resultados e as correspondentes análises estão na subseção seguinte. Por restrição de espaço, menciona-se que apenas os resultados mais relevantes para suporte às conclusões atingidas são apresentados no que se segue.

5.3. Resultados e Análises

Os resultados obtidos para *Blockchain* estão nas Figuras 1 e 2. A Figura 1 traz resultados de seis cenários de análise: S_1 , S_2 , S_3 , S_4 , S_5 e S_6 . Nos três primeiros cenários, considera-se $p = 0,6$ e varia-se z para determinar a probabilidade de a diferença de tamanho entre as ramificações concorrentes exceder o valor de X blocos. Nos três cenários seguintes, repete-se a análise para $p = 0,8$.

A partir dos resultados dessa figura, conclui-se que maiores valores de p e z tendem a fornecer maior segurança, pois esses valores tendem a aumentar a diferença de tamanho entre as ramificações. Por exemplo, a maior probabilidade associada a $X = 0$ é obtida para $p = 0,8$ e $z = 6$ (Cenário S_6). Isso significa que a diferença de tamanho neste cenário tem mais chance de ser maior do que zero do que nos demais cenários. Essa superioridade se mantém destacada para todos os valores de X em que a curva está definida. Assim, a fraude tem menor chance de ocorrer, pois apenas existiria se as ramificações se igualassem, compensando a desvantagem inicial de z blocos.

Por outro lado, aumentos arbitrários de p e z , sem prévia avaliação, podem ser inócuos. Por exemplo, os resultados obtidos para os Cenários S_1 e S_2 , respectivamente, são praticamente idênticos, ou seja, o aumento de z , de 1 para 3, pouco contribui para segurança quando p é mantido em 0,6. Diferentemente, quando $p = 0,8$, o aumento de z , também de 1 para 3, aumenta a segurança do sistema, pois a diferença entre as ramificações se torna maior, como visto nos resultados dos Cenários S_4 e S_5 .

A Figura 2, por sua vez, apresenta resultados para os mesmos seis cenários anteriores. Porém, agora são destacados valores médios das diferenças de tamanhos entre as ramificações. Confirma-se novamente que maiores valores de p e z tendem a ocasionar uma maior segurança. Por exemplo, a diferença registrada no Cenário S_6 é quase cinco vezes maior que aquela do Cenário S_1 . Porém, como antes, também se

confirma que aumentos arbitrários de p e z podem resultar inócuos. Por exemplo, nos Cenários S_5 e S_6 , dobrar o valor de z , de 3 para 6, não faz a diferença média ser dobrada. Enfim, como diretriz de projeto, sugere-se admitir valores de p acima de 0,5 e valores de z que devem observar a tolerância do cliente da aplicação para espera de confirmação.

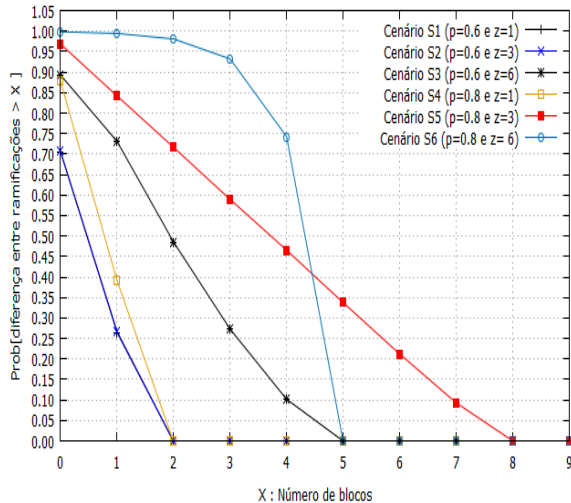


Figura 1. Diferença entre ramificações.

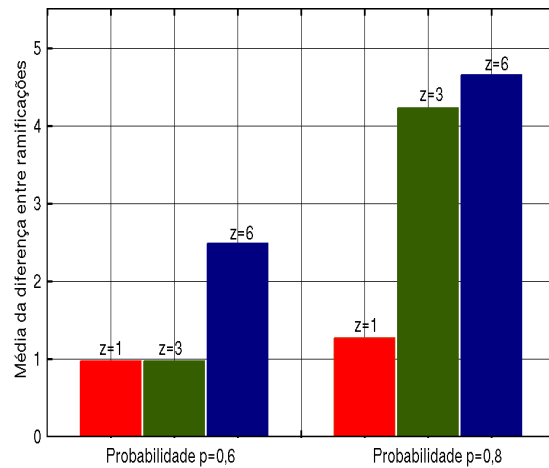


Figura 2. Média da diferença.

As Figuras 3 e 4, por sua vez, trazem os resultados para *Tangle*. São considerados seis cenários de análise: S_7 , S_8 , S_9 , S_{10} , S_{11} e S_{12} . Considere inicialmente a Figura 3. Nos três primeiros cenários, tem-se $k = 60\%$ e v sendo variado para determinar a probabilidade de a diferença de validações entre as transações T e F exceder um valor Y . Nos três cenários seguintes, faz-se a mesma análise para $k = 80\%$. Considere agora a Figura 4. Nesta figura são destacados os valores médios das diferenças de validações entre as transações T e F para os mesmos seis cenários anteriores.

A partir dos resultados da Figura 3, observa-se que maiores valores de k e v redundam em maior segurança. Isso também ocorre na *Blockchain* ao aumentar-se os valores de p e z . Entretanto, na *Tangle* esse aumento de segurança é mais contundente. Por exemplo, vê-se que as curvas de probabilidade têm valor inicial mais próximo ao valor unitário que aquelas da Figura 1. Além disso, mesmo aumentos independentes de v ou de k , sem avaliação prévia, são capazes de aumentar as probabilidades. Essa independência entre parâmetros não foi verificada na *Blockchain*. Ou seja, o nível de segurança é mais facilmente otimizado na *Tangle*.

Na Figura 4, por sua vez, tem-se que, para $k = 60\%$ ou 80% , ao dobrar-se ou triplicar-se o valor de v , há um aumento de mesma magnitude nos correspondentes valores médios das diferenças de validações entre as transações T e F , implicando maior segurança. Essa correspondência é diferente daquela observada na *Blockchain*. Em síntese, a tecnologia *Tangle* é uma solução mais atrativa que *Blockchain*. Por fim, como diretriz de projeto, sugere-se admitir valores de k acima de 50% e valores de v que devem observar a tolerância do cliente da aplicação para espera de confirmação.

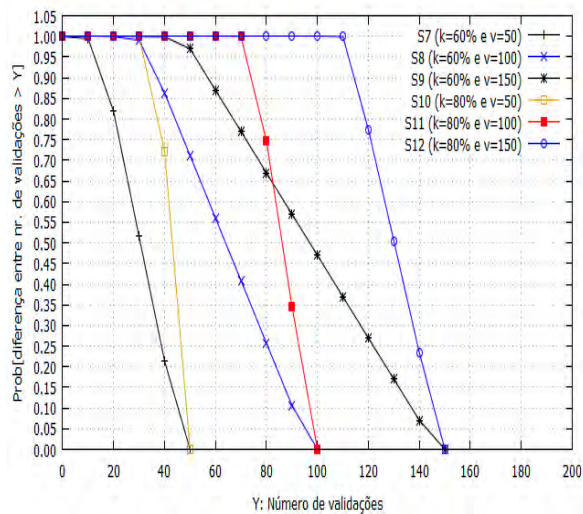


Figura 3. Diferença entre validações.

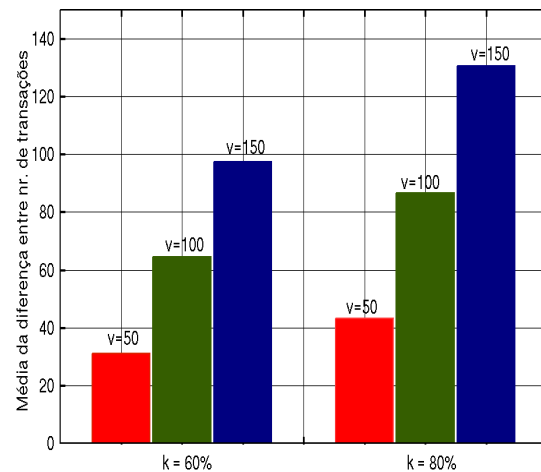


Figura 4. Média da diferença.

6. Conclusões e Trabalhos Futuros

Este artigo realizou uma análise competitiva entre as tecnologias *Blockchain* e *Tangle*. Para tanto, inicialmente foi feito um estudo teórico das estruturas de dados e dos algoritmos para manipulação das informações. Na sequência, por meio de simulação, foram examinados diferentes cenários para mensurar o nível de segurança do sistema.

O estudo teórico mostrou principalmente que o uso da *árvore de Merkle* para *Blockchain* é uma solução teórica efetiva, pois garante a otimização da complexidade de tempo para execução de *consultas* e *adições* de transações. Essa efetividade se mostra promissora na prática conforme sejam maiores os blocos de dados. O contraponto é que o processamento matemático da *prova de trabalho* demanda significativa capacidade de processamento, justamente por lidar com o bloco inteiro de informações.

Por sua vez, o uso de DAG para *Tangle* teve seu destaque pela sua simplicidade de implementação: uma transação por vértice. Isso possibilita uma execução prática otimizada de *consultas* e *adições* de transações. Além disso, a realização da *prova de trabalho* é menos onerosa que aquela da *Blockchain* por lidar com menos informações. Dado que o processamento típico em cenários de IoT envolve poucos dados, a *Tangle* termina sendo então uma tecnologia mais atrativa na prática que a *Blockchain*.

Em relação à segurança, tem-se precipuamente o que se segue. Os experimentos permitiram ver que as tecnologias *Blockchain* e *Tangle* apresentam satisfatório nível de segurança no que se refere à inviolabilidade das informações, podendo ser consideradas para projetos reais de aplicações IoT. No entanto, sob o viés da competição, a *Tangle* foi entendida aqui como uma solução mais atrativa. Essa conclusão adveio do fato de ela ser uma solução que mais objetivamente e facilmente responde à mudança de configuração de parâmetros para alcançar o nível de segurança desejado.

Como trabalhos futuros, tem-se: (i) análise de cenários mais complexos, com diferentes valores de validação de transações na *Tangle* e com disputas entre múltiplas ramificações na *Blockchain*; (ii) desenvolvimento de modelos analíticos e novos modelos de simulação com distribuições diferentes de Poisson; e (iii) análise competitiva dos algoritmos de consenso *proof of work* e *proof of stake* [Li e Liao 2017].

Referências

- Antonopoulos, M. (2017). *Mastering Bitcoin: Programming the Open Blockchain*. 2nd Edition. Sebastopol, California: O'Reilly Media.
- Berman, P.; Karpinski, M.; and Nekrich, Y. (2007). Optimal trade-off for Merkle tree traversal. *Theoretical Computer Science*, v. 372, n. 1, pp. 26-36.
- Chandran, N. R. and Manuelb, E. M. (2016). Performance Analysis of Modified SHA-3. *Procedia Technology*, v. 24, pp. 904-910.
- Cormen, T. H; Leiserson, C. E.; and Rivest, R. L. (2009). *Introduction to Algorithms*. 3rd Edition. Cambridge, Massachussets: MIT Press.
- Courtois, N. T. and Bahack, L. (2014). On subversive miner strategies and block withholding attacks in bitcoin digital currency. CoRR, abs/1402.1718. Disponível em: <https://arxiv.org/abs/1402.1718>. Acessado em: 8 de dezembro de 2017.
- Danzi, P. et al. (2017). Analysis of the Communication Traffic for Blockchain Synchronization of IoT Devices. CoRR, abs/1711.00540. Disponível em: <https://arxiv.org/abs/1711.00540v1>. Acessado em: 8 de dezembro, 2017.
- De Souza e Silva, E.; Figueiredo, R.; and Leão, R. (2009). The TANGRAM-II integrated modeling environment for computer systems and networks. *ACM SIGMETRICS Performance Evaluation Review*, v. 36, n. 4, pp. 64-69.
- Dorri, A.; Kanhere, S.; and Jurdak, R. (2017). Towards an Optimized Blockchain for IoT. In: *International Conference on Internet-of-Things Design and Implementation (IoTDI'17)*, Pittsburgh, PA, USA.
- Eyal, I. and Sirer, E. G. (2014). Majority is not enough: bitcoin mining is vulnerable. In: *International conference on financial cryptography and data security*, pp. 436-452. Springer, Berlin, Heidelberg.
- Garay, J.; Kiayias, A.; Leonardos, N. (2015). *The bitcoin backbone protocol: analysis and applications*. LNCS, Springer, Berlin, Heidelberg, v. 9057, pp. 281-310.
- Gaur, A. et al. (2015). Smart city architecture and its applications based on IoT. *Procedia Computer Science*, v. 52, p. 1089-1094.
- Gervais, A. et al. (2016). On the security and performance of proof of work blockchains. In: *ACM Conference on Computer and Communications Security*, Vienna, Austria.
- Gilbert, H. and Handschuh, H. (2004). Security analysis of SHA-256 and sisters. LNCS, Springer, Berlin, Heidelberg, v. 3006, pp. 175-193.
- Hassanalieragh, M. et al. (2015). Health monitoring and management using Internet-of-Things (IoT) sensing with cloud-based processing: Opportunities and challenges. In: *IEEE International Conference on Services Computing*, p. 285-292.
- Huckle, S. et al. (2016). Internet of Things, blockchain and shared economy applications. *Procedia Computer Science*, v. 98, pp. 461-466.
- IOTA. (2017). What is IOTA? Disponível em: <https://iota.readme.io/docs>. Acessado em: 8 de dezembro de 2017.

- Karame, G. O. (2016). On the security and scalability of bitcoin's blockchain. In: ACM Conference on Computer and Communications Security (CCS'16), Vienna, Austria.
- Karame, G. O.; Androulaki, E.; Capkun, S. (2012). Two bitcoins at the price of one? Double-spending on fast payments in bitcoin. In: ACM Conference on Computer and Communications Security (CCS'12), Raleigh, NC, USA.
- Lerner, S. D. (2015). DagCoin: a cryptocurrency without blocks. Disponível em: <https://bitslog.files.wordpress.com/2015/09/dagcoin-v41.pdf>. Acessado em: 6 de dezembro de 2017.
- Lewenberg, Y.; Sompolinsky, Y.; and Zohar, A. (2015). Inclusive Block Chain Protocols. Disponível em: http://fc15.ifca.ai/preproceedings/paper_101.pdf. Acessado em: 8 de dezembro de 2017.
- Li, I-C and Liao, T-C. (2017). A Survey of blockchain security issues and challenges. International Journal of Network Security, v. 19, n. 5, pp.653-659, 2017.
- Nakamoto, S. (2008). Bitcoin: A peer-to-peer electronic cash system. Disponível em: <https://bitcoin.org/bitcoin.pdf>. Acessado em: 8 de dezembro de 2017.
- Opara, E. U. and Soluade, O. A. (2015). Straddling the next cyber frontier: the empirical analysis on network security, exploits, and vulnerabilities. International Journal of Electronics and Information Engineering, v. 3, n. 1, pp. 10-18.
- Popov, S. (2017). The Tangle – Version 1.3. Disponível em: https://iota.org/IOTA_Whitepaper.pdf. Acessado em: 6 de dezembro, 2017.
- Rodrigues, C. K. S. (2017a). Sistema Bitcoin: uma análise da segurança das transações. Revista Brasileira de Sistemas de Informação, v. 10, n. 3, pp. 5-23.
- Rodrigues, C. K. S. (2017b). Uma análise simples de eficiência e segurança da tecnologia Blockchain. Revista de Sistemas e Computação, v. 7, n. 2, pp. 147-162.
- Rosenfeld, M. (2014). Analysis of hasrate-based double spending. CoRR, abs/1402.2009. Disponível em: <https://arxiv.org/abs/1402.2009>. Acessado em: 8 de dezembro de 2017.
- Silva, G. A. e Rodrigues, C. K. S. (2016). Mineração individual de bitcoins e litecoins no mundo. In: Simpósio Brasileiro em Segurança da Informação e de Sistemas Computacionais (SBSeg 2016), Niterói, Rio de Janeiro, Brasil.
- Singh, J. (2014). Cyber attacks in cloud computing: a case study. International Journal of Electronics and Information Engineering, v. 1, n. 2, pp. 78-87.
- Sompolinsky, Y. and Zohar, A. (2013). Accelerating Bitcoin's Transaction Processing – Fast Money Grows on Trees, Not Chains. Cryptology ePrint Archive, Report 2013/881. Disponível em: <https://eprint.iacr.org/2013/881>. Acessado em: 8 de dezembro de 2017.
- Sompolinsky, Y. and Zohar, A. (2015). Secure high-rate transaction processing in bitcoin. LNCS, Springer, Berlin, Heidelberg, v. 8975, pp. 507-527.
- Sonstebo, D. (2017). Curl disclosure, beyond the headline. Disponível em: <https://blog.iota.org/curl-disclosure-beyond-the-headline-1814048d08ef>. Acessado em: 6 de dezembro de 2017.

Contaminação Epidêmica em Redes: Imunidade Coletiva e Suas Implicações Frente a Atacantes Estratégicos

Vilc Rufino^{1,2}, Daniel Menasché², Italo Cunha³, Cabral Lima², Leandro P. de Aguiar⁴

¹DCTIM - Marinha do Brasil, Centro, Rio de Janeiro - RJ

²PPGI - Universidade Federal do Rio de Janeiro, Ilha do Fundão, Rio de Janeiro - RJ

³DCC - Universidade Federal de Minas Gerais, Pampulha, Belo Horizonte, MG

⁴Siemens Corporate Research, Princeton, NJ

Abstract. *Herd immunity, one of the most fundamental concepts in network epidemics, occurs when a large fraction of the population is immune against a virus or malware. The few individuals who have not taken countermeasures against the threat are assumed to have very low chances of infection, as they are indirectly protected by the rest of the population. Although very fundamental, herd immunity does not account for strategic attackers scanning the network for vulnerable nodes. In face of such attackers, extant nodes who linger vulnerable in the network become easy targets. The greater the proportion of individuals in a community who are immune, the higher the probability that the attacker will choose those few who are not immune. In this paper, we propose an analytical model which allows us to capture the impact of countermeasures against attackers when both in-network as well as exogenous infections coexist. Using the proposed model, we show that a diverse set of potential attacks produces non-trivial equilibria, some of which go counter to herd immunity, where nodes are more prone to adopt countermeasures when the remainder of the nodes has decided to do so.*

Resumo. *Imunidade coletiva, um dos conceitos fundamentais associados à contaminações em redes, ocorre quando grande parte da população está imune a um ataque oriundo de uma classe de códigos maliciosos. A menor parcela de indivíduos que permanecem sem adotar contramedidas tende a estar mais protegida de ataques, por efeito da redução da contaminação epidêmica. Contudo, esta condição não leva em consideração a capacidade do atacante de estrategicamente buscar por indivíduos vulneráveis, se tornando alvos fáceis. Neste artigo, propomos um modelo analítico que nos permite capturar o impacto das contramedidas face a epidemias e atacantes estratégicos. Usando o modelo propostos, podemos verificar a existência de um equilíbrio não trivial entre o custo de uma imunização e a probabilidade de estar infectado em função do número de suscetíveis.*

1. Introdução

Motivação Modelos epidêmicos são largamente estudados há décadas. Provenientes da biologia, e posteriormente também usados no estudo da propagação de vírus em redes de computadores [Murray 1988, Kephart and White 1991], tais modelos visam capturar

como um vírus se espalha em uma rede, seus impactos e o efeito das contramedidas para mitigar ameaças. As contramedidas geralmente passam por imunizar indivíduos vulneráveis. Quando a parcela da população imunizada atinge um limiar percentual significativo dentro da rede, evita-se a contaminação de indivíduos que ainda não foram imunizados [Libster 2017]. Tal fenômeno é conhecido como imunidade coletiva (do inglês, *herd immunity*), e está relacionado a externalidades positivas decorrentes da imunização.

Desafios e limitações do estado da arte A análise do custo-benefício de um programa de vacinação leva em conta o número de agentes (nós) contaminados e imunes e consequentemente o risco de contaminação de agentes suscetíveis. Numa população onde poucos indivíduos são suscetíveis, estes possuem menor incentivo em investir na vacinação. Esta trata-se de uma decisão racional, que leva em conta indivíduos que seguem um comportamento de *evitar a multidão* (*avoid the crowd*), e que podem seguramente ignorar a vacina (evitando seus custos e efeitos colaterais). Contudo, esta análise ignora a possibilidade de ocorrer uma contaminação exógena, na qual sistemas computacionais podem ser contaminados por adversários estratégicos. Atualmente, é possível que um adversário estratégico consiga localizar sistemas suscetíveis (vulneráveis) em poucas horas, por meio de eficientes varreduras de todo o espaço de endereços IPv4 da Internet [Durumeric et al. 2013]. Ainda que hajam poucos indivíduos suscetíveis, neste caso o risco de contaminação é alto. O risco aumenta para os poucos indivíduos ainda não vacinados. Há um incentivo para que estes apliquem as vacinas, e a decisão racional é de *seguir a multidão* (*follow the crowd*). Não é de nosso conhecimento nenhum trabalho na literatura de epidemias que tenha considerado o efeito de atacantes estratégicos com poder para varrer todo o endereço de IPs da Internet, na busca por nós vulneráveis. O presente trabalho visa preencher esta lacuna.

Objetivos e metodologia Neste trabalho, estabelecemos políticas ótimas de aplicação de contramedidas, levando em conta seus custos e benefícios. Ao considerarmos os benefícios da aplicação de uma contramedida para um determinado nó, levamos em consideração o fato de que estes dependem das estratégias adotadas pelos demais participantes. Por meio de um modelo analítico, determinamos condições para existência de equilíbrios, nos quais os nós não possuem incentivos para modificar suas estratégias de contramedidas.

Nossos achados também são embasados por resultados empíricos sobre a aplicação de *patches*, observados experimentalmente na rede Shodan. Avaliando o comportamento da população de usuários que expõem dispositivos de controle industrial na Internet, podemos observar que para certos dispositivos a população se comporta num padrão condizente com o *follow the crowd*, enquanto que para outros dispositivos o comportamento assemelha-se a *avoid the crowd*.

Contribuições Podemos resumir nossas contribuições conforme a seguir:

Identificação do comportamento de usuários reais Avaliando dados advindos da plataforma Shodan, identificamos que em cenários reais, para inúmeros dispositivos, conjuntos de usuários se comportam de forma similar ao *follow the crowd*. Em

outras palavras, para tais dispositivos, quanto mais usuários aplicam uma contramedida (*patch*), maior a chance de um novo usuário também aplicar tal contramedida. Esta observação serve como motivador adicional para estudar-se tal regime, que vai ao encontro da literatura padrão de epidemiologia, na qual presume-se que o comportamento padrão é do tipo *avoid the crowd*.

Modelo analítico Propomos um modelo analítico capaz de capturar fatores positivos e negativos associados as contramedidas de segurança computacional em um jogo de defesa e ataque. O modelo proposto é simples, tratável, enquanto ainda tem poder expressivo para produzir *tradeoffs* relacionados à vacinação de nós em rede.

Equilíbrio Apresentamos uma investigação numérica de equilíbrio no sistema, indicando que para dois valores extremos de regime a probabilidade de infecção cresce monotonicamente como função do tamanho da população suscetível (vulnerável), correspondendo ao comportamento de seguir a multidão. Com o sistema sob alguns ajustes, a probabilidade de infecção primeiro diminui e depois cresce em função do tamanho da população suscetível.

Organização O restante deste artigo é organizado da seguinte forma. Na Seção 2 apresentamos uma descrição dos aspectos relevantes do sistema sobre o qual se propagam as epidemias em consideração. Avaliamos o comportamento de usuários reais com relação a aplicação de contramedidas na Seção 3. Tais comportamentos motivam aspectos do modelo analítico apresentado na Seção 4, seguido por sua análise na Seção 5. Trabalhos relacionados e a conclusão seguem nas Seções 6 e 7.

2. Descrição do sistema

2.1. *Malware*: infecções exógenas e endógenas no mundo real

WannaCry é um código malicioso que ficou conhecido em 12 de maio de 2017 por ser um *ransomware* (sequestra arquivos de usuários e exige resgate) que em apenas um dia havia atingido 230.000 usuários infectados em mais de 150 países [Lee 2017]. As vulnerabilidades exploradas no protocolo SMBv2 haviam sido divulgadas e corrigidas pela Microsoft ainda em março. Em [Lee 2017] questiona-se como é possível que uma ameaça explorando um protocolo típico de redes locais tenha se espalhado tão rapidamente. O *WannaCry* é um código malicioso que possui uma alta taxa de contaminação endógena, mas bastante limitada a capacidade de contaminação entre redes. Em geral, o código entrava nas redes locais por meio de anexos em emails falsos (contaminações exógenas). Neste trabalho, propomos um modelo analítico que visa capturar o impacto de infecções exógenas na propagação de epidemias em sistemas computacionais.

Mirai é outro código malicioso que também se espalhou rapidamente pela Internet. Porém, diferente do *WannaCry*, seu alvo eram dispositivos que pudessem estar com configurações inadequadas ou mesmo de fábrica. Estes foram usados como fonte de ataques de Negação de Serviço Distribuídos (DDoS), por meio de um controle centralizado (*botmaster*). A análise de [Antonakakis et al. 2017] revela que a estrutura do código fonte possui uma parte do código, executado nas vítimas, que busca por novos alvos e realiza comandos do *botmaster*; ao encontrar uma vítima, são testados combinações conhecidas de usuário e senhas, e caso haja sucesso, a informação é passada para um servidor, que de

forma assíncrona usa o login e senha para carregar o código apropriado de acordo com a arquitetura do dispositivo. Como o código não é residente, uma simples reinicialização do dispositivo pode fazer com que o código executado na vítima seja descarregado, e o servidor que serve ao atacante deve recontaminar as vítimas novamente. *Este comportamento é similar ao do modelo SIS, considerado neste trabalho.*

Após ter conhecimento de quais dispositivos estão vulneráveis, a capacidade do atacante se limitará a capacidade do servidor de carregar o código nas vítimas e de manter a rede operacional. Como não há contaminação de vítimas para vítimas, podemos supor que a taxa de contaminação endógena é pequena, mas não nula pois a descoberta de novas vítimas ainda se dá pelas vítimas. A taxa de contaminação exógena, por outro lado, é um elemento chave do sistemas, e é limitada pela capacidade do carregador do código malicioso injetá-lo nas vítimas identificadas. *O impacto de tal taxa de contaminação exógena é um dos objetos de estudo deste trabalho.*

2.2. Poder do atacante

Consideramos um adversário ao sistema (atacante que possui controle de um *malware*) que possui um poder computacional limitado. Sua capacidade média de contaminação por unidade de tempo é denotada por Λ . No caso mais simples, esta capacidade de ataque é voltada de forma uniforme entre os nós suscetíveis existentes no sistema.

2.3. Contramedidas

Contramedidas moderadas A vacinação é uma contramedida importante para evitar a disseminação epidêmica. Em sistemas computacionais existem formas de vacinações de efeito moderado e de efeito total. Vacinação de efeitos moderados são aquelas realizadas por meio de atualizações de sistemas e anti-vírus baseados em assinatura, aos quais possuem a necessidade de constantes renovações, diárias ou semanais (“jogo do gato e rato”). Tão logo as atualizações são disponibilizadas, *hackers* usam técnicas de despistamento, modificando o código e o comportamento dos programas maliciosos. Assim, promovem-se várias gerações de um mesmo programa malicioso. As novas versões de sistemas de proteção (anti-vírus) precisam lidar com as evoluções de códigos maliciosos, caracterizando-se assim um processo de contaminação epidêmica tipicamente caracterizado como SIS (Susceptible Infected Susceptible). *Segundo este modelo, adotado neste trabalho, os nós alternam entre os estados de suscetível e infectado.*

Contramedidas rigorosas Algumas contramedidas contra códigos maliciosos envolvem tratamentos mais rigorosos, tais como a desconexão de nós da rede, instalação de sistemas operacionais mais modernos e sistemas de anti-vírus de efeito total. Estes últimos detectam mais eficientemente os códigos maliciosos, porém possuem um custo de manutenção, poder computacional e capacidade de memória maiores que os anti-vírus de efeito moderado. *Para todos os propósitos práticos, neste trabalho os nós que aplicam algum tipo de contramedida rigorosa são considerados imunes, e são removidos da população de interesse. O tamanho da população, portanto, é igual ao número de nós que não aplicaram contramedidas ou que aplicaram contramedidas moderadas.*

2.4. Dilema da atualização (*patch management*)

Toda aplicação de um *patch* de correção ou a atualização de um sistema computacional possui um custo. Muitas vezes a simples paralisação do sistema computacional pode gerar perdas consideráveis. Além disso, uma atualização pode representar uma mudança de tecnologia, que pode representar a substituição de todo um parque instalado e investimentos elevados. Os modelos de epidemias podem auxiliar essa decisão, fornecendo a probabilidade e o valor esperado de um agente ou rede ser contaminada. Neste trabalho, focamos no impacto da decisão de um agente sobre a decisão dos demais.

Evitando a multidão Essa é a abordagem clássica, e se assemelha ao modelo biológico. Se todos os indivíduos foram vacinados, o risco da epidemia diminui, pois há poucos indivíduos que podem ser contaminados, diminuindo o contágio. Portanto no caso da maioria dos indivíduos aplicarem a atualização (*patch*), a decisão de fazer a atualização é desincentivada.

Seguindo a multidão Outra abordagem é semelhante a brincadeira infantil de *pique-esconde*, no qual quanto mais indivíduos se salvaram, maiores as chances dos indivíduos que ainda não foram salvos serem descobertos e perderem o jogo. Este é o caso onde existe um adversário inserido no sistema em busca de sistemas vulneráveis para contaminação. Portanto, se o poder do atacante é finito e a maioria dos indivíduos aplicaram a atualização, a decisão de também fazer a atualização é incentivada.

3. Avaliação do comportamento real de usuários: identificando *follow the crowd* e *avoid the crowd* na Internet

A seguir, visamos identificar na Internet comportamentos que se assemelhem a padrões do tipo *seguir* ou *evitar* a multidão. Para tal, usamos dados reportados em [Wang et al. 2017] sobre a aplicação de *patches* por parte dos usuários. De acordo com [Wang et al. 2017], em uma análise de mais de 64 mil amostras de dispositivos de sistemas de controle industrial, menos de 30% desses sistemas são atualizados para versões imunes a ataques, dentro de um prazo de 60 dias desde a descoberta da vulnerabilidade.

Seleção dos dados Adotamos os seguintes critérios para selecionar os sistemas cujos dados são relevantes para nossas análises: 1) selecionamos sistemas que possuem pelo menos duas versões amostradas; 2) dado que os sistemas podem ter várias versões, para cada sistema escolhemos as versões mais populares, ou seja, as que estavam presentes em mais dispositivos durante o histórico de medições; 3) excluimos os sistemas que não tiveram ao menos dez dispositivos ativos em uma única medição em qualquer uma das versões mais populares; 4) para facilitar e padronizar a caracterização dos sistemas selecionados, numeramos sequencialmente o número das versões, de modo que os valores menores são os mais antigos e valores maiores correspondem aos mais modernos.

Caracterizando o comportamento Os típicos comportamentos de *seguir* ou *evitar* a multidão foram caracterizados conforme a adoção de versões mais modernas ou mais antigas. O comportamento de *seguir* (resp., *evitar*) a multidão ocorre quando o pico de usuários da versão mais moderna (resp., mais antiga) **ocorre após** do pico de usuários da

versão mais antiga (resp., mais moderna), mostrando que há um típico comportamento para atualizar (resp., ignorar) as versões mais recentes do sistema, que se propaga pela população.

A Tabela 1 apresenta o resultado da observação do comportamento de *seguir* a multidão (S), *evitar* a multidão (E) e um comportamento indefinido (I), onde não podemos caracterizar nenhum comportamento típico. As duas primeiras colunas se referem respectivamente ao índice da versão mais popular (1ºpop) e ao dia no qual ocorreu a medição com maior número de sistemas utilizando a respectiva versão (topo 1º) ¹ e as duas próximas colunas referem-se a segunda versão mais popular. A coluna intitulada S/E/I indica o comportamento típico da população. Para fazer a classificação do comportamento, definimos o índice $\iota = (\text{topo } 1^\circ - \text{topo } 2^\circ) / (\text{topo } 1^\circ + \text{topo } 2^\circ)$. O caso $\iota > 0,01$ (resp., $\iota < -0,01$) corresponde ao comportamento de *seguir* (resp., *evitar*) a multidão. Valores entre $-0,01$ e $0,01$ são associados a comportamentos indefinidos.

1ºpop	topo 1º	2ºpop	topo 2º	S/E/I	Nome do sistema (arquivo)
11	847	6	268	S	Allegro RomPager
2	857	1	854	I	AMX NetLinx A
35	833	23	850	E	Apache httpd
25	450	21	191	S	AVM FRITZ!Box Fon WLAN 7170 SIP
5	831	2	847	I	Boa HTTPd
11	156	4	857	E	Dropbear sshd
6	757	4	755	I	Lantronix MSS100 serial interface fingerd
13	448	8	311	S	lighttpd
7	849	5	848	I	Microsoft IIS httpd
12	850	6	833	S	Microsoft SQL Server
2	558	1	567	I	MoxaHttp
45	852	36	833	S	MySQL
36	837	8	848	I	nginx
33	852	28	852	I	OpenSSH
17	857	6	164	S	ProFTPD
7	513	5	521	I	Schneider BMX NOE 0100
5	532	2	525	I	Schneider BMX P34 2020
8	533	7	533	I	Schneider Electric SAS TSXETY4103
8	569	6	572	I	Siemens BACnet Field Panel
3	564	1	547	S	Siemens PXG3
9	1105	8	1096	I	Siemens SIMATIC IM151
5	1119	4	1117	I	Siemens SIMATIC S7 1200
39	2097	38	2097	I	Siemens SIMATIC S7 300
18	838	1	840	I	Tridium Niagara httpd
4	845	3	847	I	Virata-EmWeb
5	841	4	839	I	vxTarget ftpd
7	858	5	858	I	VxWorks ftpd
4	856	3	845	I	WindWeb

Tabela 1. Comportamento *seguir* ou *evitar* a multidão, observado em populações de usuários de sistemas reais de controle industrial conectados à Internet. A maioria dos produtos possui comportamento típico de *seguir* a multidão, o que vai ao encontro de *evitar* a multidão, típico de sistemas biológicos.

A Figura 1 ilustra o número de dispositivos adotando cada uma das versões de um determinado produto, ao longo do tempo, para dois produtos distintos (Allegro RomPage e Dropbear sshd). A população do Allegro RomPage possui comportamento típico de *seguir* a multidão (a versão mais nova substitui a versão mais antiga). Já o Dropbear sshd possui população com comportamento compatível com *evitar* a multidão. O número de dispositivos com versão antiga cresce em conjunto com o número de dispositivos adotando a versão mais nova do produto. Este último comportamento deve-se, por exemplo,

¹Os dias são medidos com relação à data da primeira coleta.

a novas instalações do produto, que muitas vezes podem vir embarcados com versões antigas do firmware. Mesmo que o comportamento de *evitar* a multidão não esteja sendo tomado de forma consciente e estratégica, os seus impactos são os mesmos que aqueles observados numa população em que indivíduos param de aplicar uma vacina (contramedida), por considerar que a ameaça é desprezível. Eventualmente, a população pode se ver em face a uma epidemia de um vírus que se pensava erradicado.

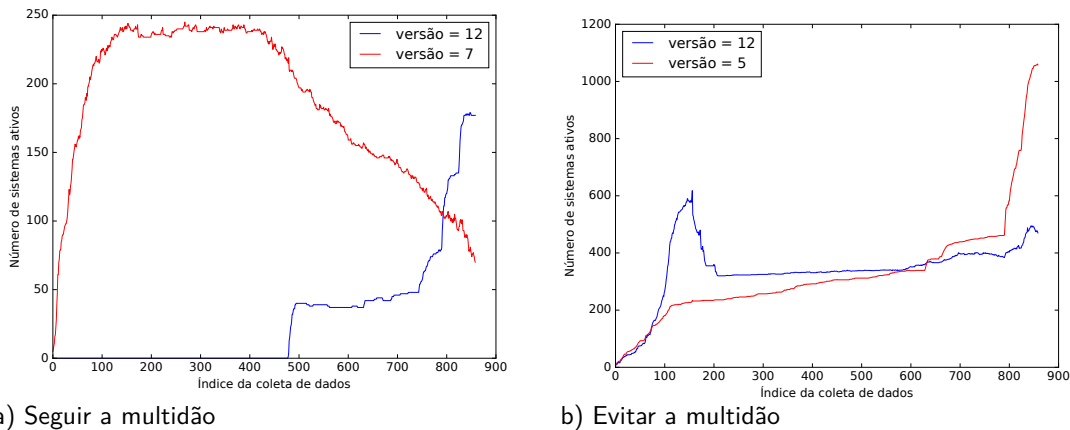


Figura 1. a) Comportamento de seguir a multidão : sistema Allegro RomPage. b) Comportamento de evitar a multidão: sistema Dropbear sshd

4. Modelo epidêmico

Nesta seção apresentamos o modelo a ser usado no decorrer do presente artigo.

4.1. Visão geral

Na presente seção apresentamos um exemplo para ilustrar algumas propriedades importantes do modelo proposto. A seção serve para apreciar intuitivamente algumas das características do modelo. Os parâmetros apresentados possuem fins ilustrativos, de modo a capturar e evidenciar as propriedades de interesse. O modelo será apresentado na próxima seção em maiores detalhes.

Assumimos que o atacante pode identificar nós vulneráveis, mas não pode distingui-los entre suscetíveis e infectados. Além disso, quando existem N nós vulneráveis na rede, a taxa de ataque por nó é $\lambda = \Lambda/N$.

A Figura 2 ilustra graficamente como a probabilidade de infecção varia em função do tamanho da população de nós vulneráveis em diferentes regimes. Quando há um domínio da infecção endógena, o sistema se comporta como a curva mais acima do gráfico, onde a probabilidade de infecção é elevada, independente da quantidade de nós vulneráveis. Todos os incentivos são para que cada indivíduo aplique a vacina (contramedida mais custosa); quando a taxa de infecção endógena atinge valores moderados (próximo do nulo multiplicativo), a probabilidade de infecção passa a depender do número de nós vulneráveis. Nesse caso, na medida em que o número de indivíduos na rede aumenta, verificamos inicialmente um domínio da infecção exógena, e posteriormente da endógena. Para atacantes de capacidade limitada, o aumento do número de nós

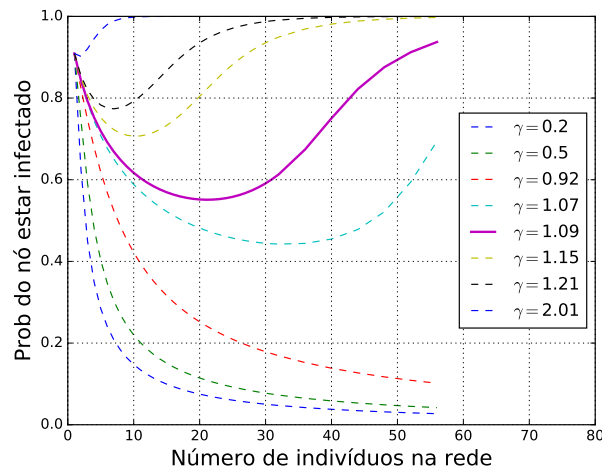


Figura 2. Probabilidade de um nó estar infectado em função do tamanho da população de vulneráveis.

vulneráveis faz com que diminua-se a probabilidade de um determinado nó estar infectado. Os indivíduos são incentivados a aplicar a vacina quando o tamanho da população é extremamente pequeno, a não aplicá-la quando o tamanho da população é um pouco maior, e novamente a aplicá-la quando a população é grande.

Quando a infecção endógena é $\gamma = 0, 2$, a probabilidade de infecção apenas diminui com o aumento de nós vulneráveis, como mostrado nas curvas mais abaixo; quando a taxa de contaminação endógena aumenta, na medida em que o número de nós vulneráveis aumenta eles novamente passam a ter incentivo para aplicar a vacina. Assim, há uma transição do comportamento de seguir para evitar a multidão.

variável	descrição	valor de referência
N	população pronta para infecção	-
M	tamanho total da população	56
γ	taxa de infecção endógena (por aresta)	1.09
λ	taxa de infecção exógena (por nó)	Λ/N
Λ	taxa de infecção exógena total	10
A	matriz de adjacência (conexões)	completa
d	número de nós vizinhos infectados	-
$\lambda\gamma^d$	taxa de infecção (por nó)	-
μ	taxa de recuperação	1
$\pi(x)$	probabilidade do estado x	-
i	número de nós infectados na rede	-
ρ	probabilidade de um nó escolhido ao acaso estar contaminado	-

Tabela 2. Tabela de notação

Estendendo um pouco mais a análise do gráfico da Figura 2, destacamos a curva que possui a contaminação endógena $\gamma = 1,09$, linha contínua do gráfico; considerando como um custo limite para aplicar a vacina uma probabilidade de contaminação igual a 0,6 podemos verificar na Figura 3 dois pontos de equilíbrio: o primeiro instável entre 11 e 12 nós vulneráveis, e outro estável entre 30 e 31 nós vulneráveis. A probabilidade de contaminação é mínima quando o número de nós vulneráveis é igual a 21, atingindo

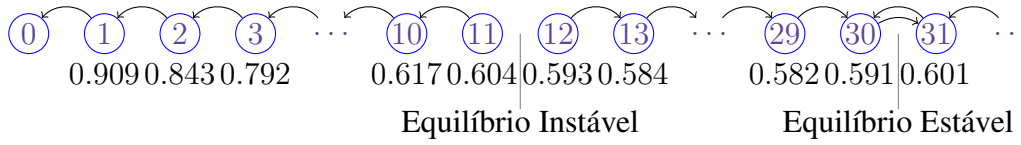


Figura 3. Dinâmica do número de nós na vulneráveis (suscetíveis ou infectados). Considerando uma taxa de contaminação endógena $\gamma = 1.09$ e custo da contramedida igual a 0.6 obtemos dois pontos de equilíbrio. O equilíbrio dinâmico instável (resp., estável) conta com um número de nós na rede que varia entre 11 e 12 (resp., 30 e 31).

o valor de 0,5510. Segundo o modelo proposto, para alcançar-se este ótimo global são necessários incentivos adicionais por parte de entidades regulatórias e/ou empresas de seguros [Grossklags et al. 2008].

4.2. O modelo multiplicativo de contaminação (*scaled* SIS)

Consideramos uma população finita contendo M nós, dos quais, um total de N nós não aplicaram nenhuma forma de vacina, e portanto possuem uma vulnerabilidade que pode ser explorada por um atacante. Os $M - N$ nós, aplicaram alguma vacina e consideramo-os imunes.

Cada um desses N nós podem assumir os estados de suscetível (S ou 0) ou infectado (I ou 1). Um nó infectado pode ser recuperado para o estado suscetível após um período médio $1/\mu$ distribuído exponencialmente.

Um nó suscetível pode ser infectado por um atacante externo (infecção exógena) ou por um ataque interno (infecção endógena) de um vizinho na rede. Seja d o número de vizinhos infectados de um determinado nó, podemos assumir que a taxa de infecção é exponencialmente dependente de d , isto é, a contaminação endógena ocorre a uma taxa γ^d . O efeito da contaminação exógena é assumido como multiplicativo, o qual ocorre a uma taxa λ por nó. Logo, a taxa total de infecção é $\lambda\gamma^d$, assumiremos ainda que o tempo entre as infecções é dado por uma distribuição exponencial.

Seja $\mathbf{x}$ um estado possível da rede, entre todos os estados possíveis $\mathcal{X}$. O estado é um vetor N dimensional, $\mathbf{x} \in \{0, 1\}^N$, $\mathbf{x} = (x_1, x_2, \dots, x_k, \dots, x_{N-1}, x_N)$, onde $x_k \in \{0, 1\}$. A dinâmica do sistema é caracterizada por um processo Markoviano contínuo, homogêneo temporal, irreduzível e de estados finitos. Cada estado da rede corresponde a um estado no processo Markoviano. Além disso, o nosso processo Markoviano é reversível, conforme [Kelly 1979].

4.3. Função de utilidade e topologia da rede

Cada nó decide se deve investir na aplicação de uma vacina. O investimento na aplicação de uma vacina possui um custo C , enquanto que a contaminação possui um custo H (usando a terminologia consistente com [Hayel et al. 2014]). Se um determinado nó decide não investir em uma vacina, ele passa estar sujeito ao processo de contaminação epidêmica, podendo ser contaminado. Contudo, após ser contaminado o nó pode se recuperar, sem necessariamente ser imunizado, tal como retorno ao estado inicial, formatação e reinicialização. Após a remoção dos nós que decidiram investir na vacina e estão imunes, a topologia restante da rede determina o processo de dispersão da epidemia.

A topologia da rede é dada por uma matriz de adjacência A_{NN} , onde cada entrada a_{kl} é igual a 1 se o nó x_k e x_L estão contaminados, e zero caso contrário. Como tratamos de conexões não direcionadas, então $a_{kl} = a_{lk}$. Os elementos na diagonal de A são todos iguais a 0.

4.4. Matriz geradora infinitesimal

Seja Q a *matriz infinitesimal* associada ao processo Markoviano. Seja $d_k^{(i)}$ o número de vizinhos infectados ao nó k no estado i , e seja $x_k^{(i)}$ a k -ésima entrada do vetor de estados, no qual o vetor esteja no estado i do processo Markoviano.

A entrada na linha i e coluna j de Q , $q_{i,j}$, é dada conforme a seguir:

$$q_{i,j} = \begin{cases} -\sum_{\substack{j \neq i \\ j=1}}^{2^N} q_{i,j}, & \text{se } i = j \\ \lambda \gamma^{d_k^{(i)}}, & \text{se } x_k^{(i)} = 0, x_k^{(j)} = 1, x_l^{(i)} = x_l^{(j)}, \text{ para } l \neq k \\ \mu, & \text{se } x_k^{(i)} = 1, x_k^{(j)} = 0, x_l^{(i)} = x_l^{(j)}, \text{ para } l \neq k \\ 0, & \text{caso contrário} \end{cases} \quad (1)$$

4.5. Modelo de ameaça

Consideramos um atacante com uma capacidade de ataque igual a Λ infecções por unidade de tempo. Em um ajuste simplista, tal qual a capacidade total dividido pelo número total de nós suscetíveis, obtemos uma taxa de infecção por nó de $\lambda = \Lambda/N$.

Mais genericamente, podemos permitir que Λ seja escrita como uma função de N , e λ assume uma forma mais generica de função dada por Λ e N , temos que $\lambda(\Lambda, N)$ denota a taxa de infecção exógena por nó.

4.6. Distribuição de probabilidades dos estados em regime estacionário

A distribuição de probabilidades dos estados em regime estacionário é dada por [Zhang and Moura 2014, Zhang et al. 2017].

$$\pi(\mathbf{x}) = \frac{\tilde{\pi}(\mathbf{x})}{Z} \quad (2)$$

onde

$$\tilde{\pi}(\mathbf{x}) = \left(\frac{\lambda}{\mu}\right)^{1^T \mathbf{x}} \gamma^{\mathbf{x}^T A \mathbf{x} / 2}, \mathbf{x} \in \mathcal{X} \quad (3)$$

e $Z = \sum_{\mathbf{x} \in \mathcal{X}} \tilde{\pi}(\mathbf{x})$.

O número de nós infectados no estado $\mathbf{x}$ é dado por $1^T \mathbf{x} = \sum_{k=1}^N x_k$. O número de ligações (arestas) onde os dois nós vizinhos estão contaminados (arestas contaminadas) é dado por $\frac{1}{2} \mathbf{x}^T A \mathbf{x} = \frac{1}{2} \sum_{k=1}^N \sum_{\substack{l=1 \\ l \neq k}}^N x_k x_l a_{kl}$.

5. Análise do modelo

A seguir, focamos em uma topologia totalmente conectada. Neste caso, por simetria temos que

$$\tilde{\pi}(i) = \binom{N}{i} \left(\frac{\lambda(N)}{\mu} \right)^i \gamma^{i(i-1)/2}, \quad i = 0, \dots, N \quad (4)$$

A probabilidade de infecção de um nó escolhido de forma aleatoria e uniforme é

$$\rho(N) = \frac{1}{N} \sum_{i=0}^N i \frac{\tilde{\pi}(i)}{Z} \quad (5)$$

A análise direta das equações acima é complexa, por envolver um termo quadrático no expoente de γ . Para simplificar a análise, consideramos uma solução aproximada para o modelo acima. Para tal, definimos $\hat{\rho}(N) \approx \rho(N)$ e $\hat{\pi}(i) \approx \tilde{\pi}(i)$,

$$\hat{\rho}(N) = \frac{1}{N} \sum_{i=0}^N i \frac{\hat{\pi}(i)}{\hat{Z}} \quad (6)$$

$$\hat{\pi}(i) = \binom{N}{i} \left(\frac{\lambda(N)}{\mu} \gamma^{N^*/2} \right)^i \quad (7)$$

$$\hat{Z} = \sum_{i=0}^N \hat{\pi}(i) \quad (8)$$

onde $N^*(N)$ é uma função crescente de N , que denotamos simplesmente por N^* para simplificar a notação. Nos referimos a o modelo proposto para aproximar a solução do modelo original como *modelo binomial*, por fazermos uso do binômio de Newton na demonstração do resultado a seguir.

Lema 5.1. *No modelo binomial, temos que*

$$\hat{\rho}(N) = \frac{1}{1 + \mu/(\lambda(N)\gamma^{N^*/2})} \quad (9)$$

Demonstração. O resultado é fruto de manipulações algébricas,

$$\hat{Z}\hat{\rho}(N) = \frac{1}{N} \sum_{i=0}^N i \binom{N}{i} \left(\frac{\lambda(N)}{\mu} \gamma^{N^*/2} \right)^i = \sum_{i=1}^N \binom{N-1}{i-1} \left(\frac{\lambda(N)}{\mu} \gamma^{N^*/2} \right)^i \quad (10)$$

$$= \left(\frac{\lambda(N)}{\mu} \gamma^{N^*/2} \right) \left(1 + \frac{\lambda(N)}{\mu} \gamma^{N^*/2} \right)^{N-1} \quad (11)$$

O resultado segue a partir da obtenção, de forma similar, da expressão de $\hat{Z}$. $\square$

O resultado acima pode ser usado, por exemplo, para caracterizarmos os pontos de equilíbrio do sistema.

Teorema 5.1. *O modelo binomial admite no máximo dois equilíbrios interiores ao considerarmos um custo de vacinação constante, desde que $\gamma > 1$, $\partial N^*/\partial N$ seja positivo e não decrescente e $\lambda(N)$ decrescente.*

Demonstração. Seja $\tau(N) = (\lambda(N)/\mu)\gamma^{N^*/2}$. Então, pelo lema acima, temos que $\partial\hat{\rho}(N)/\partial N = (\partial\tau/\partial N)/(\tau^2(1 + 1/\tau)^2)$. Claramente, todos os termos de $\partial\hat{\rho}(N)/\partial N$ são positivos, com exceção de $\partial\tau/\partial N$. Temos que

$$\frac{\partial\tau}{\partial N} = \lambda(N)\gamma^{N^*/2} \left(\frac{1}{2} \log \gamma \frac{\partial N^*}{\partial N} + \frac{\lambda'(N)}{\lambda(N)} \right) \quad (12)$$

Se $\frac{\partial N^*}{\partial N}$ for positivo e não decrescente, e $\lambda(N)$ for decrescente, a expressão acima admite um único zero. Assim, a função $\hat{\rho}(N)$ possui no máximo um ponto de mínimo interno, e por isso cruza qualquer linha horizontal em no máximo dois pontos. $\square$

O resultado acima está de acordo com a ilustração apresentada nas Figuras 2 e 3. Segundo a Figura 2, em todos os casos em que $\gamma > 1$, a probabilidade de um nó estar infectado primeiro diminui e depois aumenta, ou simplesmente sempre aumenta. Conforme discutido na Figura 3, a um ponto de mínimo correspondem dois equilíbrios, um estável e um instável.

Caso especial: $\lambda(N) = \Lambda/N$ O modelo proposto é factível de análise em fórmula fechada para vários casos especiais da função $\lambda(N)$. Para fins de ilustração, consideramos o caso especial em que $\lambda(N) = \Lambda/N$. Este caso corresponde a um atacante que tem poder de ataque (*budget*) constante igual a Λ infecções por segundo, e divide esse poder entre os N nós da rede. Nesse caso, $\lambda'(N) = -\Lambda/N^2$. Assumindo para fins de simplificação que $N^* = N$, temos

$$\frac{\partial}{\partial N} \hat{\rho}(N) = \kappa \left(\frac{1}{2} \log \gamma - \frac{1}{N} \right) \quad (13)$$

onde κ é uma constante positiva. Podemos verificar que $\frac{\partial}{\partial N} \hat{\rho}(N) = 0$ quando $N = (2/\log \gamma)$. No caso de $\gamma = 1.09$ encontramos o valor crítico quando $N \approx 23$, que está de acordo com aquele apresentado na Figura 2.

Note também que podemos obter fórmulas fechadas para os pontos de equilíbrio. Para tal, seja C o custo de aplicação da contramedida. Então, os pontos de equilíbrio interno são pontos tais que $\hat{\rho}(N) - C = 0$. Assumindo para fins de simplificação que $N^* = N$, temos que os valores de N que satisfazem a equação de equilíbrio são $N = \frac{-2}{\log \gamma} W \left(\frac{\Lambda(C-1) \log \gamma}{2C\gamma^{1/2}} \right)$ onde $W(x)$ é a função de Lambert, que admite dois valores reais, correspondentes aos ramos -1 e 0. No caso de $\Lambda = 10$, $\gamma = 1.09$ e $C = 0.6$, por exemplo, temos os valores de N correspondentes aos ramos -1 e 0 dados por 45,6 e 9,7. O segundo equilíbrio condiz com o resultado da Figura 3, enquanto que o primeiro foi superestimado. Tal fato deve-se à simplificação de que $N^* = N$, conforme ilustrado na Figura 4, podendo o resultado ser melhor aproximado em refinamentos sucessivos do modelo. A Figura 4(a) mostra a probabilidade de um nó estar infectado, em função do número de nós na rede. A aproximação $N^* = N$ captura bem o comportamento do sistema antes de $\rho(N)$ atingir o seu ponto de mínimo. Depois deste ponto, é necessário ajustar N^* para o seu valor ótimo, que é crescente e sempre maior que N como ilustrado na Figura 4(b) (satisfazendo os critérios do Teorema 5.1). De forma mais geral, cabe destacar que o fato de a função de Lambert possuir dois ramos reais está de acordo com a constatação de que o sistema admite no máximo dois equilíbrios internos (vide Teorema 5.1).

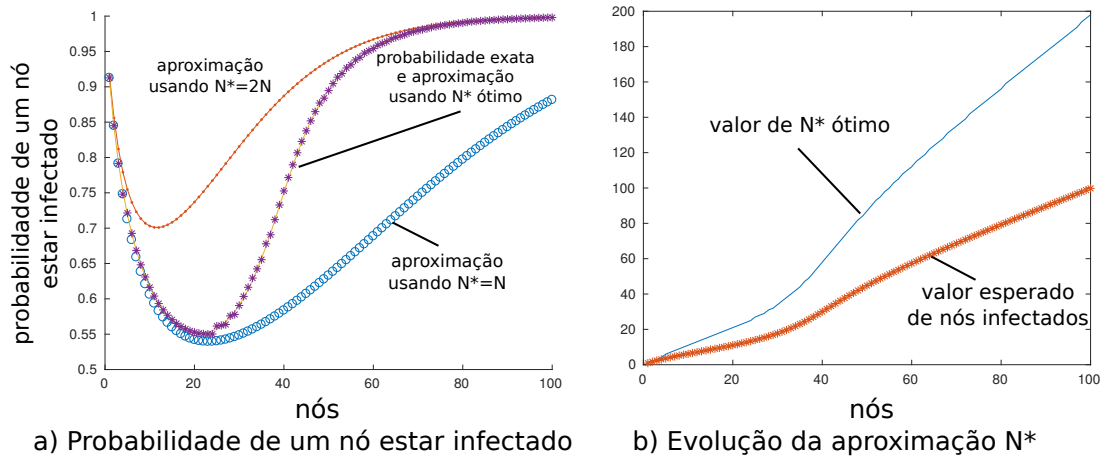


Figura 4. Validação da solução aproximada pelo modelo binomial ($\gamma = 1,09$).

6. Trabalhos relacionados

A literatura sobre epidemias é vasta. Entretanto, não é de nosso conhecimento nenhum trabalho anterior que tenha avaliado externalidades negativas resultantes da aplicação de contramedidas, usando um modelo de disseminação de epidemias como o SIS.

Em [Maillé et al. 2011], por exemplo, os autores discutem as motivações financeiras e econômicas por trás dos interesses dos hackers na elaboração de *malware*. Os autores partem do princípio de que quanto mais difundida uma vulnerabilidade, maior o estímulo para tomar proveito da mesma. Neste trabalho, entretanto, focamos no cenário oposto, que consiste em um atacante que busca promover ataques direcionados, se aproveitando de vulnerabilidades que foram sanadas pela maioria da população mas que ainda contem com vítimas fáceis.

Embora os modelos clássicos epidemiológicos sejam usados para explicar comportamentos biológicos observáveis na natureza, a propagação de programas maliciosos (*malwares*) em rede de computadores precisa capturar as contaminações intencionais e direcionadas. Neste artigo, uma solução usada para capturar essa característica foi adicionar ao modelo biológico mais uma forma de contaminação, que chamamos de contaminação exógena, tal como em [Altman et al. 2014, Zhang and Moura 2015, Van Mieghem and Cator 2012].

Em [Grossklags et al. 2008], os autores consideram externalidades positivas, por exemplo, advindas da instalação de *honeypots* em uma rede. Tal externalidade é similar em essência a ideia de que quanto mais nós suscetíveis, menores as chances de um nó escolhido de forma uniforme e aleatória ser infectado. Embora o presente trabalho tenha semelhanças em termos de suas conclusões com [Maillé et al. 2011], os autores deste último negligenciaram aspectos epidemiológicos, que são o foco do presente estudo.

7. Conclusões

O campo da epidemiologia demonstra que quanto maior o número de indivíduos imunes, menor é a chance de que uma contaminação se espalhe na rede. Quando há muitos indivíduos que aplicaram uma contramedida, os indivíduos restantes têm menos incentivo

para aplicar a contramedida (evitando a multidão). Contudo, com a caracterização de um atacante externo com capacidade limitada, argumentamos que é possível que o número menor de indivíduos que não aplicaram a contramedida torne-os alvos preferidos dos atacantes externos. Isso faz com que os poucos indivíduos que não aplicaram algum tipo de contramedida sejam incentivados a fazê-lo (seguindo a multidão). Para capturar tais efeitos, propusemos um modelo analítico, e analisamos os seus pontos de equilíbrio. Numericamente, indicamos os regimes que caracterizam situações de evitar e seguir a multidão, e evidenciamos que tais regimes de fato ocorrem na prática, usando dados reais providos pela plataforma Shodan sobre o comportamento de *patching* de usuários na Internet.

Referências

- Altman, E., Avritzer, A., El-Azouzi, R., Menasche, D. S., and de Aguiar, L. P. (2014). Rejuvenation and the spread of epidemics in general topologies. In *ISSREW*, pages 414–419. IEEE.
- Antonakakis, M., April, T., Bailey, M., Bernhard, M., Bursztein, E., Cochran, J., Durumeric, Z., Halderman, J. A., Invernizzi, L., Kallitsis, M., et al. (2017). Understanding the mirai botnet. In *USENIX Security Symposium*.
- Durumeric, Z., Wustrow, E., and Halderman, J. A. (2013). Zmap: Fast internet-wide scanning and its security applications. In *Usenix Security*, volume 2013.
- Grossklags, J., Christin, N., and Chuang, J. (2008). Secure or insure?: a game-theoretic analysis of information security games. In *WWW*, pages 209–218. ACM.
- Hayel, Y., Trajanovski, S., Altman, E., Wang, H., and Van Mieghem, P. (2014). Complete game-theoretic characterization of sis epidemics protection strategies. In *Decision and Control (CDC), 2014 IEEE 53rd Annual Conference on*, pages 1179–1184. IEEE.
- Kelly, F. P. (1979). *Reversibility and stochastic networks*. John Wiley Sons, New York, NY, USA.
- Kephart, J. O. and White, S. R. (1991). Directed-graph epidemiological models of computer viruses. In *Research in Security and Privacy*, pages 343–359. IEEE.
- Lee, B. (2017). Ransomware: Unlocking the lucrative criminal business model. Technical report, Palo Alto Networks. <https://www.paloaltonetworks.com/>.
- Libster, R. (2017). The power of herd immunity. TED Talk.
- Maillé, P., Reichl, P., and Tuffin, B. (2011). Interplay between security providers, consumers, and attackers: A weighted congestion game approach. In *GameSec*, pages 67–86. Springer.
- Murray, W. H. (1988). The application of epidemiology to computer viruses. *Computers & Security*, 7(2):139–145.
- Van Mieghem, P. and Cator, E. (2012). Epidemics in networks with nodal self-infection and the epidemic threshold. *Physical Review E*, 86(1):016116.
- Wang, B., Li, X., de Aguiar, L. P., Menasche, D. S., and Shafiq, Z. (2017). Characterizing and modeling patching practices of industrial control systems. *POMACS*, 1(1):18.
- Zhang, J. and Moura, J. M. (2014). Diffusion in social networks as sis epidemics: Beyond full mixing and complete graphs. *IEEE Journal of Selected Topics in Signal Processing*, 8(4):537–551.
- Zhang, J. and Moura, J. M. (2015). Contact process with exogenous infection and the scaled sis process. *arXiv preprint arXiv:1507.00396*.
- Zhang, J., Moura, J. M., and Zhang, J. (2017). Contact process with exogenous infection and the scaled sis process. *Journal of Complex Networks*.

Detecção de Anomalias de Desempenho em Aplicações de Alto Desempenho baseadas em Tarefas em Clusters Híbridos

Vinicius Garcia Pinto^{1,2}, Lucas Mello Schnorr¹, Arnaud Legrand²,
Samuel Thibault³, Luka Stanisic⁴, Vincent Danjean²

¹ Instituto de Informática, Universidade Federal do Rio Grande do Sul
Porto Alegre, Brasil

² Univ. Grenoble Alpes, Inria, CNRS, Grenoble INP, LIG,
Grenoble, France

³ Inria Bordeaux Sud-Ouest
Bordeaux, France

⁴ Max Planck Computing and Data Facility
Garching, Germany

{vgpinto, schnorr}@inf.ufrgs.br
{arnaud.legrand, samuel.thibault, vincent.danjean}@inria.fr
luka.stanisic@mpcdf.mpg.de

Abstract. *Programming paradigms in High-Performance Computing have been shifting towards task-based models which are capable to more readily adapt to heterogeneous and scalable supercomputers. Detecting performance anomalies in such environments is particularly difficult since it must consider architecture heterogeneity, variability, and the capability to obtain trusted measurements. This work presents a case-study about the detection of anomalies in the execution of the well-known tiled dense Cholesky factorization developed with StarPU. Our experiments have been conducted in a variety of hybrid multi-node platforms to demonstrate how we are capable to detect and highlight performance anomalies.*

Resumo. *Os paradigmas de programação em Computação de Alto Desempenho estão mudando para modelos baseados em tarefas que são capazes de se adaptar a supercomputadores com arquiteturas heterogêneas e escaláveis. A detecção de anomalias de desempenho em tal cenário é particularmente difícil uma vez que ela deve considerar a heterogeneidade da arquitetura, a variabilidade e a capacidade de obter medições confiáveis. Este trabalho apresenta um estudo de caso sobre a detecção de anomalias na execução da conhecida fatoração de Cholesky por blocos desenvolvida com StarPU. Os experimentos foram conduzidos em uma variedade de plataformas com múltiplos nós híbridos para demonstrar a capacidade de detectar e destacar anomalias de desempenho.*

1. Introdução

As plataformas de Processamento de Alto Desempenho (PAD) têm sido construídas com nós computacionais híbridos visando atender a crescente demanda por poder computacional. Este fato tem provocado uma mudança nos paradigmas de programação para PAD em

direção a modelos baseados em tarefas que executam sobre ambientes de execução, uma vez que estes são capazes de se adaptar mais facilmente a plataformas com arquiteturas heterogêneas e escaláveis. Ambientes de execução automatizam etapas que, tradicionalmente, seriam feitas manualmente pelo programador como o escalonamento das tarefas, balanceamento de carga e as transferências de dados.

Entretanto, uma vez que a execução de aplicações neste modelo de hardware e software é inerentemente estocástica, detectar anomalias de desempenho em tal cenário é particularmente difícil. As ferramentas e técnicas existentes para detecção de anomalias em PAD foram desenvolvidas para análise de aplicações implementadas com o modelo BSP (*bulk-synchronous parallel*). Neste contexto, a identificação de anomalias é relativamente trivial, uma vez que a aplicação é bem estruturada em termos de fases de computação e comunicação e o *hardware* é homogêneo. Por outro lado, em plataformas híbridas e com aplicações baseadas em tarefas, esta detecção é desafiadora pois deve considerar a heterogeneidade da arquitetura, a variabilidade do desempenho e a capacidade de se obter medições confiáveis.

Neste trabalho, nós apresentamos um estudo de caso sobre a detecção de anomalias na execução de uma fatoração de Cholesky implementado com paralelismo de tarefas. Neste contexto, é esperado que a duração das tarefas de um mesmo tipo seja dependente apenas do tipo de recurso de processamento no qual elas são executadas. Assim, uma tarefa anômala é aquela cuja duração é excessivamente longa em comparação com as outras tarefas do mesmo tipo que se executam sobre um recurso de mesma natureza (CPU ou GPU).

Este estudo de caso utiliza nosso *framework* construído sobre ferramentas de análise de dados para visualizar rastros de execução. Nossos experimentos foram executados com o ambiente de execução StarPU [Augonnet et al. 2011] em uma variedade de plataformas híbridas (mesclando CPUs homogêneas e GPUs homogêneas) e os resultados estão divididos em duas contribuições: a detecção de anomalias em núcleos CPU e uma investigação sobre as anomalias na duração das tarefas que executam em GPUs. Embora em trabalhos anteriores [Pinto et al. 2016, Pinto et al. 2018] nós já tenhamos utilizado nosso *framework* para analisar execuções com StarPU, este trabalho foca em um aspecto que não foi discutido previamente. A investigação descrita neste trabalho permite enumerar hipóteses para origem das anomalias identificadas na duração das tarefas assim como a correção destas.

O restante deste artigo está organizado da seguinte forma. A Seção 2 discute aspectos relacionados ao contexto deste trabalho como as arquiteturas híbridas CPU/GPU (2.1), a programação baseada em tarefas (2.2), o ambiente de execução StarPU (2.3) e a fatoração de Cholesky (2.4) bem como os trabalhos relacionados (2.5). A Seção 3 apresenta o nosso *framework* StarVZ (3.1) utilizado para a análise de execuções de aplicações baseadas em tarefas que executam em arquiteturas híbridas com suporte do ambiente de execução StarPU, bem como o modelo utilizado para identificar anomalias na duração das tarefas (3.2). A Seção 4 apresenta os resultados da detecção de anomalias em núcleos CPU (4.2) e em dispositivos GPU (4.3). Por fim, a Seção 5 apresenta as considerações finais. Os dados brutos dos experimentos, o código fonte para processamento dos rastros e geração das figuras estão disponíveis na versão reproduzível deste trabalho em <https://gitlab.in2p3.fr/vgarciap/wperformance2018>.

2. Contexto e Trabalhos Relacionados

2.1. Arquiteturas Híbridas CPU/GPU

A estagnação da frequência dos processadores tem incentivado a adoção de outros meios para atender a demanda de desempenho das aplicações atuais. Um destes meios tem sido a construção de arquiteturas híbridas, combinando recursos de processamento distintos em um mesmo nó. Muitas plataformas de computação de alto desempenho atuais são constituídas de nós computacionais híbridos, onde vários dispositivos aceleradores são acoplados a processadores tradicionais para aumentar a capacidade computacional do sistema. Esta tendência pode ser confirmada por meio da análise da edição de Novembro de 2017 da lista Top500, onde 102 dentre os 500 sistemas classificados utilizam algum tipo de dispositivo acelerador como GPUs e Xeon Phi [Meuer et al. 2014].

2.2. Programação Paralela baseada em Tarefas

A programação baseada em tarefas tem sido adotada como uma alternativa para tratar a heterogeneidade das plataformas atuais de computação de alto desempenho. Tradicionalmente, as aplicações de alto desempenho são implementadas seguindo o modelo BSP utilizando ferramentas clássicas e explícitas como MPI, OpenMP, *threads* e CUDA. Entretanto, em plataformas híbridas, este modelo leva a códigos monolíticos, que são difíceis de desenvolver, otimizar e manter, embora ainda possibilite que se atinja um desempenho próximo do ideal.

Por outro lado, o modelo baseado em tarefas se baseia em ambientes de execução dinâmicos visando abstrair a complexidade da plataforma e explorar de maneira eficiente os múltiplos níveis de paralelismo do *hardware*. Neste modelo, o programador descreve a aplicação em termos de tarefas abstratas e suas dependências. O ambiente de execução é responsável por atividades relacionadas à plataforma como escalonamento das tarefas, transferências de dados e gerenciamento das dependências. Para executar estas atividades, o ambiente de execução analisa características da plataforma e do código, tais como as implementações fornecidas para uma tarefa, os recursos de processamento disponíveis (CPUs, GPUs, etc), a duração estimada das tarefas, a localidade dos dados e a largura de banda da interconexão. A partir das informações coletadas, o ambiente de execução pode usar heurísticas de escalonamento adequadas e implementar otimizações, como por exemplo, antecipar as transferências de dados, visando atingir melhor desempenho. Como exemplo de ferramentas com suporte ao modelo de tarefas em arquiteturas híbridas, pode-se citar: OmpSs [DURAN et al. 2011], XKaapi [Gautier et al. 2013] e StarPU [Augonnet et al. 2011].

2.3. StarPU

O StarPU [Augonnet et al. 2011] é um ambiente de execução para programação baseada em tarefas em arquiteturas híbridas. O ambiente foi inicialmente projetado para gerenciar plataformas híbridas compostas por um único nó de processamento equipado com processadores *multicore* (CPUs) e aceleradores (GPUs, Intel Xeon Phi). De forma a explorar de maneira eficiente o paralelismo da plataforma, o StarPU utiliza múltiplas implementações para uma mesma tarefa, por exemplo, com versões equivalentes para CPUs e/ou GPUs.

O escalonador utilizado pelo StarPU decide em tempo de execução onde executar as tarefas levando em conta os recursos de processamento disponíveis, o tipo destes, a localização atual dos dados a serem acessados pela tarefa e as implementações disponíveis

para a mesma. Este escalonador oferece várias políticas de escalonamento. As políticas DMDA e DMDAS têm base em modelos de desempenho pré-calibrados, enquanto as políticas WS e LWS utilizam uma estratégia baseada em roubo de tarefas, roubando tarefas do recurso mais carregado (WS) ou do vizinho mais carregado (versão LWS). A política PRIO se baseia apenas em anotações de prioridade especificadas pelo programador da aplicação.

Os algoritmos DMDA (*Deque Model Data Aware*) e DMDAS (*Deque Model Data Aware Sorted*) são membros de uma família de escalonadores do StarPU que utilizam a duração prevista da tarefa e a duração prevista das transferências de dados para executar o escalonamento das tarefas. Ambas estratégias têm base na técnica de escalonamento de listas (*list scheduling*). Os algoritmos DMDA e DMDAS são heurísticas gulosas que escalonam tarefas na ordem em que elas se tornam disponíveis. Quando da tomada das decisões de escalonamento, estas técnicas levam em conta o tempo previsto para execução da tarefa, a duração estimada das transferências de dados entre CPUs e GPUs e o desempenho relativo dos recursos de processamento para cada *kernel* computacional. O algoritmo DMDAS melhora as decisões do DMDA por meio da ordenação das tarefas em listas em cada recurso, utilizando como critério de ordenação o número de fatias de dados já transferidas e a prioridade, o que pode ser custoso quando o número de tarefas é elevado. Este algoritmo, é portanto, mais próximo da proposta original do algoritmo HEFT (*Heterogeneous Earliest Finish Time*) que respeita as prioridades e leva em conta as decisões passadas de escalonamento.

Os algoritmos WS (*Work Stealing*) e LWS (*Locality Work Stealing*) utilizam uma lista de tarefas por recurso de processamento; novas tarefas são armazenadas localmente por padrão. Quando um recurso se torna ocioso, na política WS, ele rouba tarefas do recurso mais sobrecarregado. A política LWS, por outro lado, impõe que o recurso ocioso deve, em um primeiro momento, tentar roubar do recurso mais sobrecarregado dentre os seus vizinhos. Esta escolha da vítima difere do algoritmo clássico de roubo de tarefas [Blumofe and Leiserson 1999] onde a escolha da vítima é feita por meio de uma estratégia aleatória.

O algoritmo PRIO utiliza uma simples lista centralizada que é compartilhada entre todos os recursos de processamento. Esta lista mantém as tarefas ordenadas respeitando as prioridades especificadas pelo programador da aplicação.

A extensão StarPU-MPI [Augonnet et al. 2012, Agullo et al. 2017] fornece suporte adicional para gerenciar arquiteturas com múltiplos nós, adotando a especificação MPI [Gropp et al. 1999] como meio de acesso à rede. A principal característica desta extensão é que ela utiliza um escalonador StarPU independente para cada nó. Por exemplo, se um experimento é configurado com a política DMDA, este escalonador será usado em cada um dos nós para processar uma parte do grafo acíclico direcionado (DAG, abreviação em inglês de *Directed Acyclic Graph*) da aplicação. As dependências existentes entre os nós são resolvidas através de operações MPI *send/receive* ponto-a-ponto, e gerenciadas como qualquer outra tarefa escalonada pelo ambiente de execução. Uma vez que a decomposição de domínio é estática, estas operações *send/receive* são submetidas automaticamente como tarefas do DAG. StarPU-MPI utiliza uma *thread* especificamente para gerenciar as comunicações MPI do nó. Ao concluir uma tarefa cuja saída é requerida por uma tarefa em outro nó MPI, a *thread* MPI submete a operação *send* correspondente. No

outro nó, quando da recepção dos dados por MPI, a *thread* MPI habilita a execução da tarefa correspondente.

A distribuição bidimensional clássica *Block-Cyclic Distribution* [Blackford et al. 1997] foi previamente modificada para suportar execuções com decomposição estática em múltiplos nós com apoio do StarPU-MPI. A decomposição depende dos parâmetros $P \times Q$ e do número de nós MPI. Em função do número de nós MPI, os parâmetros P e Q controlam como a matriz de entrada é particionada entre os nós com base nos blocos. O valor de P pode variar de um até o número de nós. Tomando como exemplo uma matriz de 16×16 blocos, uma configuração de $P=1 \times Q=8$ leva a uma distribuição por linhas, enquanto uma configuração de $P=8 \times Q=1$ leva a uma distribuição por colunas. Combinações de valores intermediários levam a distribuições intercaladas. Em um cenário ideal, para um dado número de nós, o valor de P deveria ser definido de forma a minimizar o perímetro de comunicação de cada nó uma vez que ele é relativo ao volume total de comunicações.

2.4. Cholesky

Neste trabalho, utilizamos a fatoração de matrizes pelo método de Cholesky como aplicação de estudo de caso. Este método é uma das operações mais comuns de álgebra linear e é usado por várias aplicações científicas. Com objetivo de melhorar a reprodutibilidade e a estabilidade de nossos experimentos, nós adotamos a implementação da fatoração de Cholesky por blocos (*tiled*) fornecida pelo pacote de álgebra linear Chameleon/MORSE [Agullo et al. 2012]. Esta implementação é construída sobre o ambiente de execução StarPU [Augonnet et al. 2011] e compilada com bibliotecas BLAS (CPU) e CUBLAS (GPUs) padrão.

A Figura 1 ilustra uma versão simplificada desta aplicação. As linhas com as chamadas `dpotrf`, `dtrsm`, `dsyrk` e `dgemm` representam a criação de tarefas StarPU com implementações de precisão dupla para CPUs e GPUs (exceto o `dpotrf` que possui apenas a versão CPU). Os parâmetros destacados **RW** e **R** indicam o modo de acesso do bloco da matriz em questão. A partir destas informações sobre os modos de acesso, o ambiente de execução pode inferir as dependências entre as tarefas e então construir um grafo direcionado acíclico (DAG) de tarefas. A cada iteração k do laço mais externo, uma tarefa `dpotrf` habilita a execução de $N - k - 1$ `dtrsm`, e então $N - k - 1$ `dsyrk`, sucedida por $\approx (N - k)^2/2$ tarefas `dgemm`. A partir das dependências, podemos observar que várias iterações podem ser executadas simultaneamente e que o número de repetições nos laços mais internos diminui quando k aumenta. Por fim, o tempo de execução de uma tarefa depende altamente do seu tipo (`dpotrf`, `dtrsm`, `dsyrk` e `dgemm`) e do tipo de recurso onde ela é executada (CPU ou GPU).

Embora neste trabalho nós tenhamos utilizado a fatoração de Cholesky como estudo de caso, existem outras aplicações que apresentam características similares e que portanto poderiam ser analisadas com os mesmos métodos e ferramentas aqui descritos. Podemos citar os algoritmos por blocos para fatoração de matrizes LU e QR que, assim como o método de Cholesky, utilizam como base chamadas às operações do padrão BLAS (*Basic Linear Algebra Subprograms*).

```

for (k = 0; k < N; k++) {
    DPOTRF(RW, A[k][k]);
    for (i = k+1; i < N; i++)
        DTRSM(RW, A[i][k], R, A[k][k]);
    for (i = k+1; i < N; i++) {
        DSYRK(RW, A[i][i], R, A[i][k]);
        for (j = k+1; j < i; j++)
            DGEMM(RW, A[i][j], R, A[i][k], R, A[j][k]);
    }
}

```

Figura 1. Pseudo-código da decomposição de Cholesky por blocos

2.5. Trabalhos Relacionados

Existem poucas ferramentas para análise de desempenho de aplicações baseadas em tarefas. Entre as ferramentas que não possuem foco em arquiteturas híbridas, podemos citar o Framesoc/Ocelotl [Dosimont et al. 2015], que oferece técnicas escaláveis para agregação de rastros. Estas técnicas agregam comportamentos homogêneos apresentados tanto na dimensão espacial (recursos de processamento) quando temporal (estados da aplicação). Comportamentos não homogêneos como anomalias são mantidos em seu estado bruto e são visualizados sem agregação. O controle sobre quais dados são considerados homogêneos ou não é feito por uma heurística baseada na relação entre a perda de informação e a complexidade da visualização. Dessa forma, a detecção de anomalias globais, como aquelas que ocorrem em todas as tarefas de um core, pode ser prejudicada, uma vez que a ferramenta tende a identificar este comportamento como homogêneo.

No contexto de aplicações com tarefas em arquiteturas híbridas, nós apresentamos previamente uma discussão preliminar dos experimentos aqui apresentados [Schnorr et al. 2017]. Uma análise similar, utilizando a mesma aplicação (cholesky do pacote Chameleon) e o mesmo ambiente de execução (StarPU) foi feita por [Beaumont et al. 2018]. Neste estudo, o objetivo dos autores foi avaliar o impacto da presença de tarefas com duração atípica no tempo total de execução da aplicação (*makespan*). A análise foi conduzida com o escalonador DMDAS em uma plataforma single-node com 24 cores e 4 GPUs. Os autores sugerem que as anomalias podem estar relacionadas a efeitos da arquitetura NUMA (*Non Uniform Memory Access*) da plataforma. Dessa forma a análise das anomalias não foi feita individualmente, mas por grupos de *cores* (correspondentes aos nós NUMA). Por fim, para avaliar o impacto da presença de tarefas anômalas no *makespan*, os autores fizeram uso de execuções simuladas utilizando StarPU-Simgrid [Stanisic et al. 2015] com injeção de tempos anômalos para algumas tarefas conforme as hipóteses enumeradas. Os principais diferenciais de nosso trabalho em relação aos demais aqui discutidos são: (a) o emprego de uma análise visual dos rastros de execução da aplicação nas dimensões espacial (recursos de processamento) e temporal (duração das tarefas), (b) nossa análise considera execuções em arquiteturas distribuídas com múltiplos nós híbridos, (c) nosso estudo de caso inclui testes com outros escalonadores além do DMDAS (padrão) e (d) nosso estudo permite enumerar hipóteses para origem e correção das causas das anomalias.

3. Framework para análise de aplicações com tarefas e Modelo de desempenho para tarefas de álgebra linear densa

Nesta seção, apresentamos o *framework* utilizado para analisar execuções de aplicações baseadas em tarefas que executam em arquiteturas híbridas.

3.1. StarVZ

Em trabalhos anteriores [Pinto et al. 2016, Pinto et al. 2018] nós propusemos um *framework*¹ baseado em uma estratégia modular e incremental que permite representar execuções de maneira visual utilizando rastros de execução previamente coletados. Esta solução é construída sobre ferramentas modernas para análise de dados, combinando *pj_dump* [Schnorr et al. 2016], a linguagem de programação R [R Core Team 2017], incluindo sua biblioteca *ggplot2* [Wickham 2009] e as funções para manipulação de dados fornecidas pelo meta-pacote *tidyverse* [Wickham 2016], além de *org-mode* [Schulte et al. 2012] e *plotly* [Sievert et al. 2016].

Este *framework* permite obter, a partir dos rastros de execução, um diagrama espaço/tempo representando a duração de cada uma das tarefas da aplicação nos recursos de processamento ao longo do tempo. Graças a estratégia modular e incremental, podemos enriquecer esta visualização básica adicionando informações relevantes como arestas de dependências entre as tarefas, estimativas para o *makespan*, porcentagem de tempo ocioso e tarefas com duração anômala. Além disso, por meio de gráficos auxiliares, podemos adicionar outras informações relevantes como a quantidade de tarefas disponível para execução em cada instante, a repartição de trabalho entre os recursos de processamento de diferentes tipos e a progressão da computação ao longo do tempo.

Em trabalhos anteriores mostramos como este conjunto de técnicas pode ajudar a identificar e corrigir problemas de escalonamento em nós híbridos equipados com CPUs *multicore* e múltiplas GPUs [Pinto et al. 2016] e como identificar e corrigir gargalos em execuções com nós híbridos em plataformas distribuídas [Pinto et al. 2018]. Neste estudo, focamos na detecção de anomalias de desempenho na duração das tarefas, buscando identificar a origem e possíveis soluções para as mesmas.

3.2. Modelo de desempenho para detecção de anomalias na duração das tarefas

No contexto do tipo de aplicação considerado neste trabalho, uma operação de álgebra linear densa implementada utilizando o modelo de programação paralela baseado em tarefas, é esperado que a duração das tarefas de um mesmo tipo seja dependente apenas do tipo de recurso de processamento no qual elas são executadas. Esta restrição é particularmente relevante no caso de aplicações já bastante otimizadas, como a fatoração de matrizes pelo método de Cholesky apresentada na Seção 2.4. Nesta aplicação, as tarefas executam chamadas a operações do padrão BLAS, que são comuns a muitas outras aplicações como as fatorações LU e QR.

Neste trabalho, nós utilizamos a seguinte Equação (1) para identificar as tarefas com duração anômala. A duração de uma tarefa t de um dado tipo quando executada em um certo recurso de processamento (core ou GPU) é considerada anormal se a sua duração t_d :

¹<https://github.com/schnorr/starvz>

$$t_d \geq Q_3 + 1.5 \times IQR \quad (1)$$

onde Q_3 é o terceiro quartil e IQR é a diferença ($Q_3 - Q_1$), sendo que Q_1 é o primeiro quartil. Os quartis são computados considerando globalmente todas as tarefas de um mesmo tipo, e que executam em um recurso de mesma natureza (core CPU ou GPU). Posteriormente, nós aplicamos esta Equação para marcar as tarefas com duração anômala.

O uso desta Equação, tipicamente usado para definir *outliers* em *boxplots*, é debatível e foi baseado em nossa experiência. De qualquer maneira, em nossa abordagem, esta fórmula é facilmente substituível. Entretanto, ela atende aos requisitos do nosso estudo de caso uma vez que considera tanto a heterogeneidade da aplicação (tipo de tarefa) quanto dos recursos de processamento (GPU, CPU).

4. Avaliação Experimental e Resultados

Nesta seção apresentamos a avaliação experimental de nossa abordagem para detecção de anomalias. Os experimentos estão divididos em duas partes: a primeira apresenta a análise do desempenho da aplicação fatoração de Cholesky quando executada em um *cluster* de nós híbridos enquanto a segunda apresenta a análise de *microbenchmarks* com chamadas diretas ao *kernel* `dgemm` da biblioteca `cuBLAS`², executados diretamente em uma GPU sem o uso do ambiente StarPU. As execuções da fatoração de Cholesky foram feitas com matrizes de dois tamanhos: 108×108 blocos de 1440×1440 e 35×35 blocos de 960×960.

4.1. Ambiente Experimental

Os experimentos aqui discutidos foram executados em arquiteturas híbridas compostas de CPUs *multicore* e de GPUs conforme detalhado na Tabela 1. A plataforma *chifflet* foi configurada com Debian Linux (kernel 4.9.0-3-amd64), gcc 6.3.0, Chameleon 0.9.1 (master), StarPU (developer, com revisões específicas para as correções propostas neste trabalho, OpenMPI 2.0.2, OpenBLAS 0.2.19 e CUDA 7.0 com *driver* 375.66. A plataforma *Hype5*, onde foram executados os *microbenchmarks*, foi configurada com Ubuntu Linux (kernel 4.4.0-112-generic), gcc 5.4.1 e CUDA 9.0 com driver 384.111.

Tabela 1. Plataformas Experimentais Utilizadas

Nome	Nós	Processador do Nó	GPUs	Rede
chifflet	8	2 × 14-core Intel Xeon E5-2680v4	2 × Nvidia GTX 1080Ti	Ethernet 10Gbit
Tupi	1	1 × 8-core Intel Xeon E5-2620v4	2 × Nvidia GTX 1080Ti	(não aplicável)
Hype5	1	2 × 10-core Intel Xeon E5-2650v3	2 × Nvidia Tesla K80	(não aplicável)

²<http://docs.nvidia.com/cuda/cublas/index.html>

4.2. Anomalias em Núcleos CPU

A Figura 2 apresenta um cenário representativo da visualização de rastros utilizando a detecção de anomalias discutida anteriormente. A análise desta execução permite identificar um comportamento instigante uma vez que a presença de tarefas com duração anormal é mais concentrada em alguns *cores* que em outros. Tarefas anormalmente longas ocorrem majoritariamente no *core* 0 de cada nó da plataforma, o que pode ser verificado nos gráficos da Figura 3. Devido ao fato que o problema aparece independente da quantidade de cores utilizada no experimento, preferimos ilustrar o problema de forma mais concisa através de um experimento com apenas 6 cores e 2 GPUs em cada um dos oito nós.

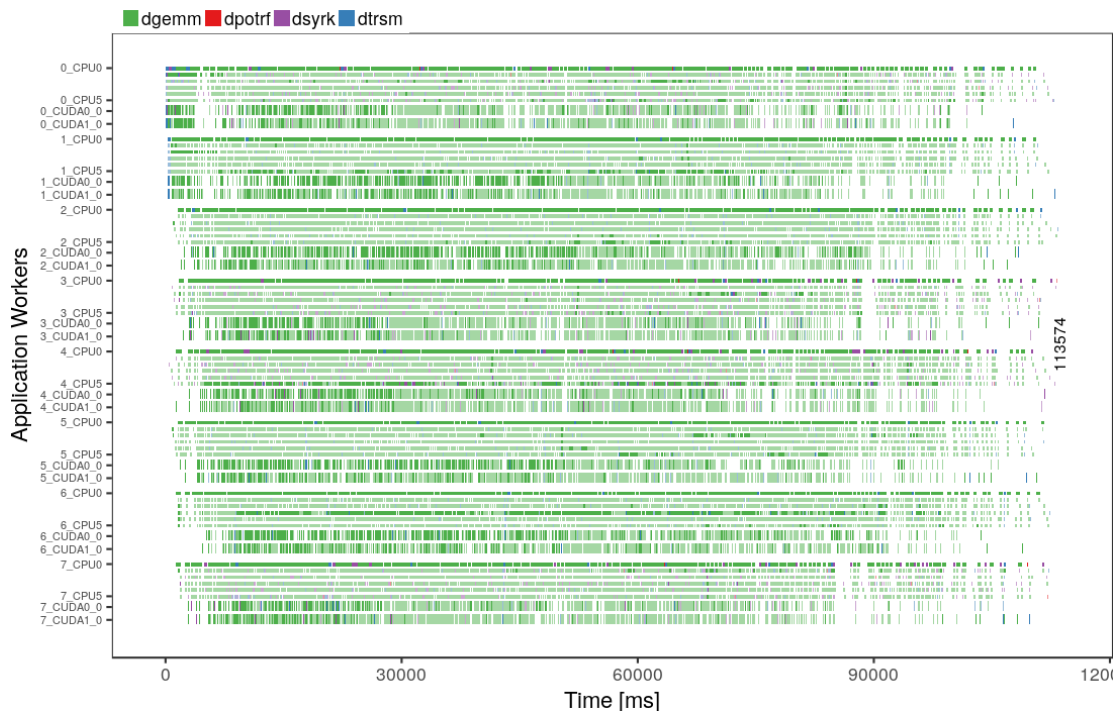


Figura 2. Anomalias (marcadas por cor mais forte) em um *core* (principalmente CPU0) em cada nó da plataforma *chifflet* durante a execução da fatoração de Cholesky em uma matriz com 108×108 blocos de 1440×1440 utilizando o escalonador LWS e P=4.

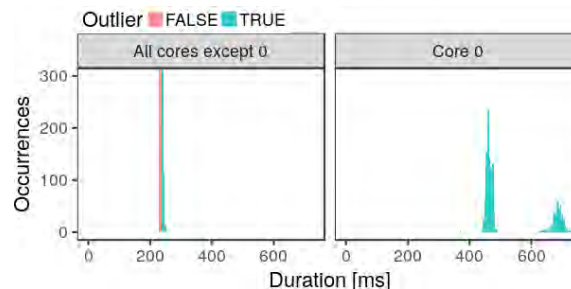


Figura 3. Comparação da ocorrência de anomalias em todos os cores exceto no Core 0 contra o cenário que considera apenas o Core 0. Em vermelho, a ocorrência de tarefas cuja duração é considerada normal pelo nosso modelo. Em azul, a ocorrência de tarefas cuja duração é avaliada como anômala.

Por padrão as *workers threads* do StarPU são fixadas aos *cores*, ou seja, *threads* que executam computação diretamente e aquelas que controlam as GPUs são associadas aos *cores* em uma distribuição 1 para 1 e a migração durante a execução não é permitida. Entretanto, além das *workers threads* existem duas outras *threads* que não são fixadas aos *cores*. A primeira é a thread principal responsável pela criação e submissão das tarefas enquanto a segunda gerencia as comunicações MPI.

A hipótese inicial para explicar as anomalias na duração das tarefas é de que as *threads* adicionais (submissão de tarefas e MPI) estejam executando no *core* 0, perturbando a *worker thread* fixada neste *core*. Para verificar esta hipótese, foram implementadas modificações no StarPU de forma que o usuário possa, manualmente, fixar cada uma das *threads* (*worker*, MPI e principal) a um *core* específico.

A Figura 4 apresenta uma execução em dois nós (marcados no eixo vertical por "0_" e "1_", cada um com 5 *cores* e 2 GPUs) utilizando dois escalonadores diferentes: DMDAS no painel da esquerda e LWS na direita. O experimento tem por objetivo verificar a efetividade da solução proposta. Podemos notar que as anomalias não estão mais presentes no *core* 0 de cada nó. Entretanto, ainda existem *cores* individualmente afetados como o *core* 2 do nó 0 da execução com DMDAS e o *core* 3 do nó 1 na execução com LWS. Experimentos adicionais indicam que as anomalias estão ou localizadas espacialmente (afetando um *core*) ou ausentes (sem correlação significativa com os *cores*).

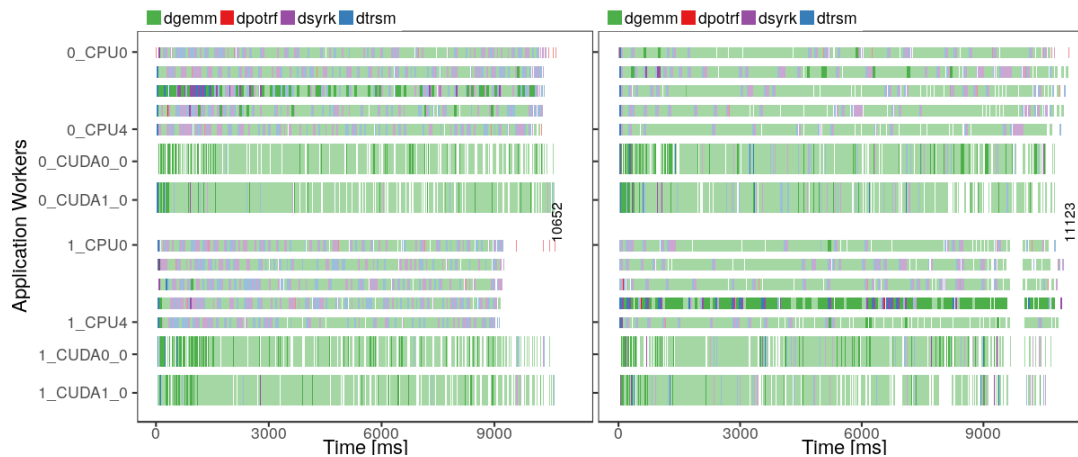


Figura 4. Execuições da fatoração de Cholesky em dois nós após as modificações no StarPU. Matriz com 35×35 blocos de 960×960 utilizando os escalonadores DMDAS (painel da esquerda) e LWS (direita) e P=1.

Uma investigação mais aprofundada, inspecionando a execução através do depurador GDB indica que o problema pode estar relacionado à implementação da biblioteca cuBLAS, cujo código é proprietário, ou às configurações de afinidade definidas pelo sistema operacional na inicialização da aplicação.

4.3. Anomalias em GPUs

As anomalias discutidas na Seção 4.2 ocorrem principalmente em *workers* executando em CPUs. Entretanto, também podemos observar na Figura 2 que existem anomalias nos *workers* que executam em GPUs. Neste caso, o comportamento parece ser comum a todas as GPUs, sem indicação de que exista uma correlação entre as anomalias e alguma GPU

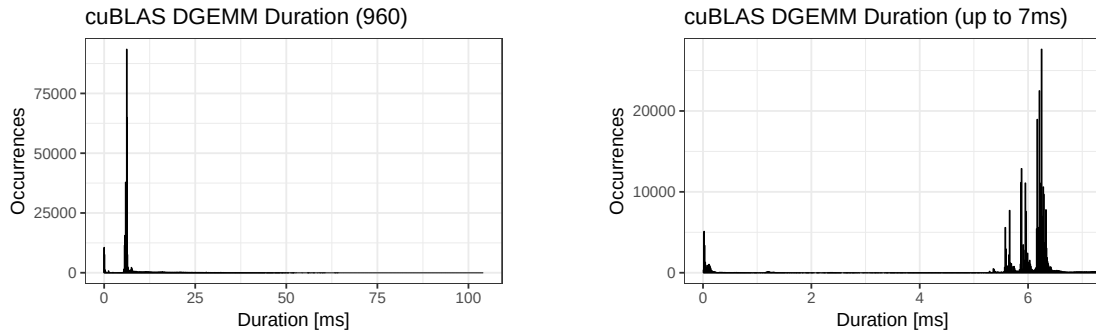


Figura 5. Histograma da duração das tarefas `dgemm` que executam em GPUs com matrizes de 960×960 . O gráfico da direita apresenta um *zoom* nas tarefas com duração de até 7ms.

específica. Uma vez que as GPUs executam somente computação, ao contrário das CPUs do caso anterior, elas não são passíveis de perturbação por *threads* de gerenciamento.

Uma hipótese para explicar a origem das tarefas com duração anômala em GPUs pode ser a imprecisão do modelo de desempenho utilizado. É possível que este modelo esteja considerando como atípicas, tarefas que estariam dentro da variabilidade esperada para execução neste *hardware*. A Figura 5 apresenta o histograma da duração das tarefas `dgemm` que executam em GPUs. A análise deste histograma mostra que a distribuição da duração destas tarefas é multimodal. O que pode ser a razão do número elevado de tarefas cuja duração é identificada como anômala pelo nosso modelo.

Uma possível explicação para a distribuição multimodal pode estar relacionada a efeitos de *warm-up* e de *cache*. A Figura 6 apresenta os resultados de um conjunto de experimentos para investigar estas hipóteses. Nestes experimentos, nós executamos diretamente chamadas ao *kernel* `dgemm` da biblioteca cuBLAS, variando o intervalo entre as execuções entre 1 e 20ms (colunas) bem como o tamanho da matriz de entrada (linhas) entre 128×128 e 1960×1960 . Pode-se observar que para que se obtenha benefício do *warm-up*, o tempo entre duas execuções não deve ser maior que a duração total do *kernel* em execução. Isto provavelmente pode estar relacionado a uma sequência de execuções do mesmo *kernel*, o que pode explicar a identificação de anomalias nas execuções em *workers* GPU com o StarPU. Além disso, quanto maior a duração do *kernel* maiores as oportunidades de se observar os resultados do *warm-up*, e a consequente aceleração.

5. Considerações Finais

Neste artigo nós apresentamos um estudo de caso sobre a detecção de anomalias na duração das tarefas de uma aplicação que executa sobre nós híbridos. A detecção deste tipo de irregularidade é desafiadora dadas as características da arquitetura híbrida e das técnicas implementadas pelo ambiente de execução. O estudo foi conduzido com o nosso *framework* baseado na linguagem de programação R e que possibilita a análise de rastros de execução obtidos com o ambiente de execução StarPU e plataformas híbridas multi-(nó, *core*, GPU).

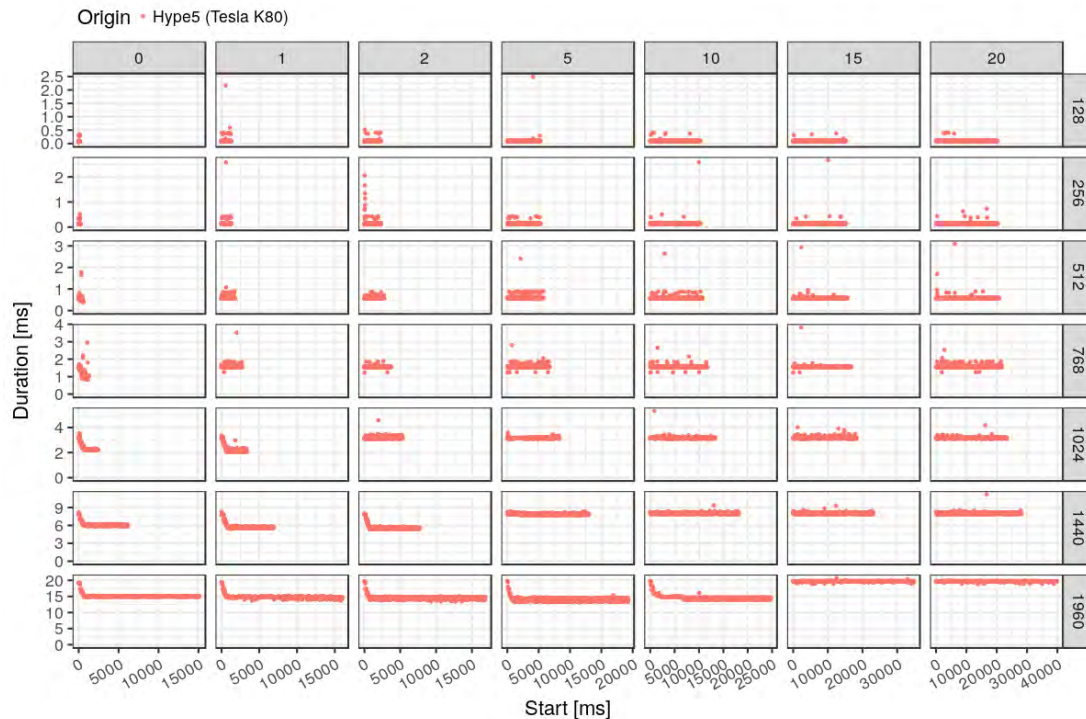


Figura 6. Análise da execução do *kernel dgemm* do cuBLAS. As colunas indicam o intervalo em mili-segundos entre duas execuções, enquanto as linhas indicam o tamanho da matriz de entrada.

Por meio dos resultados obtidos pela análise da fatoração de Cholesky fornecida pelo pacote Chameleon foi possível identificar e caracterizar anomalias na duração tanto de tarefas que executam em núcleos CPU quanto em GPUs. Em trabalhos futuros nós pretendemos estender esta análise utilizando aplicações de álgebra linear esparsa e também investigar execuções em *clusters* com processadores *many-core* como o Intel Xeon Phi.

Agradecimentos

Nós agradecemos os projetos CAPES/Cofecub 764-13, FAPERGS/Inria ExaSE, FAPERGS 16/2551-0000 354-8 (PPP 2014), FAPERGS 16/2551-0000 488-9 (PRONEX 2014), CNPq 447311/2014-0, o Laboratório Internacional CNRS/LICIA, o programa H2020 da União Europeia e o MCTI/RNP através do projeto HPC4E, identificador 689772. Alguns experimentos foram executados na plataforma Grid'5000 (<https://www.grid5000.fr>), com suporte do Inria, CNRS, RENATER e várias outras organizações francesas. O material auxiliar deste artigo³, incluindo detalhes experimentais, código fonte e trechos de código para reger as figuras é graciosamente mantido pelo CNRS-IN2P3 (<http://www.in2p3.fr/>), para o qual nós também enviamos nossos agradecimentos.

Referências

[Agullo et al. 2017] Agullo, E., Aumage, O., Faverge, M., Furmento, N., Pruvost, F., Sergeant, M., and Thibault, S. P. (2017). Achieving high performance on supercomputers

³<https://gitlab.in2p3.fr/vgarciap/wperformance2018>

- with a sequential task-based programming model. *IEEE Transactions on Parallel and Distributed Systems*, Early Access:1–14.
- [Agullo et al. 2012] Agullo, E., Bosilca, G., Bramas, B., Castagnede, C., Coulaud, O., Darve, E., Dongarra, J., Faverge, M., Furmento, N., Giraud, L., Lacoste, X., Langou, J., Ltaief, H., Messner, M., Namyst, R., Ramet, P., Takahashi, T., Thibault, S., Tomov, S., and Yamazaki, I. (2012). Poster: Matrices over runtime systems at exascale. In Wasserman, H., editor, *High Performance Computing, Networking, Storage and Analysis (SCC), 2012 SC Companion*:, pages 1332–1332.
- [Augonnet et al. 2012] Augonnet, C., Aumage, O., Furmento, N., Namyst, R., and Thibault, S. (2012). StarPU-MPI: Task Programming over Clusters of Machines Enhanced with Accelerators. In Träff, J. L., Benkner, S., and Dongarra, J. J., editors, *Recent Advances in the Message Passing Interface: 19th European MPI Users’ Group Meeting, EuroMPI 2012, Vienna, Austria, September 23-26, 2012. Proceedings*, pages 298–299. Springer Berlin Heidelberg, Berlin, Heidelberg.
- [Augonnet et al. 2011] Augonnet, C., Thibault, S., Namyst, R., and Wacrenier, P.-A. (2011). StarPU: a unified platform for task scheduling on heterogeneous multicore architectures. *Concurrency and Computation: Practice and Experience*, 23(2):187–198.
- [Beaumont et al. 2018] Beaumont, O., Eyraud-Dubois, L., and Gao, Y. (2018). Influence of Tasks Duration Variability on Task-Based Runtime Schedulers. Research report, INRIA.
- [Blackford et al. 1997] Blackford, L. S., Choi, J., Cleary, A., D’Azevedo, E., Demmel, J., Dhillon, I., Hammarling, S., Henry, G., Petitet, A., Stanley, K., Walker, D., and Whaley, R. C. (1997). *ScaLAPACK user’s guide*. Society for Industrial and Applied Mathematics.
- [Blumofe and Leiserson 1999] Blumofe, R. D. and Leiserson, C. E. (1999). Scheduling multithreaded computations by work stealing. *J. ACM*, 46(5):720–748.
- [Dosimont et al. 2015] Dosimont, D., Corre, Y., Schnorr, L. M., Huard, G., and Vincent, J.-M. (2015). Ocelotl: Large Trace Overviews Based on Multidimensional Data Aggregation. In Niethammer, C., Gracia, J., Knüpfer, A., Resch, M. M., and Nagel, W. E., editors, *Tools for High Performance Computing 2014*, pages 137–160. Springer International Publishing.
- [DURAN et al. 2011] DURAN, A., AYGUADÉ, E., BADIA, R. M., LABARTA, J., MARTINELL, L., MARTORELL, X., and PLANAS, J. (2011). Ompss: A proposal for programming heterogeneous multi-core architectures. *Parallel Processing Letters*, 21(02):173–193.
- [Gautier et al. 2013] Gautier, T., Lima, J. V., Maillard, N., and Raffin, B. (2013). Xkaapi: A runtime system for data-flow task programming on heterogeneous architectures. In *Parallel & Distributed Processing (IPDPS), 2013 IEEE 27th International Symposium on*, pages 1299–1308. IEEE.
- [Gropp et al. 1999] Gropp, W., Thakur, R., and Lusk, E. (1999). *Using MPI-2: Advanced features of the message passing interface*. MIT press.

- [Meuer et al. 2014] Meuer, H. W., Strohmaier, E., Dongarra, J., and Simon, H. D. (2014). *The TOP500: History, Trends, and Future Directions in High Performance Computing*. Chapman & Hall/CRC, 1st edition.
- [Pinto et al. 2018] Pinto, V. G., Schnorr, L. M., Stanisic, L., Legrand, A., Thibault, S., and Danjean, V. (2018). A visual performance analysis framework for task-based parallel applications running on hybrid clusters. *Concurrency and Computation: Practice and Experience*, Early Access:1–27.
- [Pinto et al. 2016] Pinto, V. G., Stanisic, L., Legrand, A., Schnorr, L. M., Thibault, S., and Danjean, V. (2016). Analyzing dynamic task-based applications on hybrid platforms: An agile scripting approach. In *Third Workshop on Visual Performance Analysis, VPA@SC 2016, Salt Lake, UT, USA, November 18, 2016*, pages 17–24.
- [R Core Team 2017] R Core Team (2017). *R: A Language and Environment for Statistical Computing*. R Foundation for Statistical Computing, Vienna, Austria.
- [Schnorr et al. 2016] Schnorr, L. M., Faverge, M., Trahay, F., de Oliveira Stein, B., and de Kergommeaux, J. C. (2016). The Paje trace file format. Technical report, UFRGS.
- [Schnorr et al. 2017] Schnorr, L. M., Legrand, A., Thibault, S., Stanisic, L., Pinto, V. G., and Danjean, V. (2017). Detecting performance outliers for task-based hpc applications in multi-[cpulgpulnode] clusters. Presentation at Workshop on Hybrid Computing 2017, Held in conjunction with SBAC-PAD 2017.
- [Schulte et al. 2012] Schulte, E., Davison, D., Dye, T., Dominik, C., et al. (2012). A multi-language computing environment for literate programming and reproducible research. *Journal of Statistical Software*, 46(3):1–24.
- [Sievert et al. 2016] Sievert, C., Parmer, C., Hocking, T., Chamberlain, S., Ram, K., Corvellec, M., and Despouy, P. (2016). *plotly: Create Interactive Web Graphics via 'plotly.js'*. R package version 4.5.6.
- [Stanisic et al. 2015] Stanisic, L., Agullo, E., Buttari, A., Guermouche, A., Legrand, A., Lopez, F., and Videau, B. (2015). Fast and Accurate Simulation of Multithreaded Sparse Linear Algebra Solvers. In *The 21st IEEE International Conference on Parallel and Distributed Systems*, pages 481–490, Melbourne, Australia.
- [Wickham 2009] Wickham, H. (2009). *ggplot2: Elegant Graphics for Data Analysis*. Springer-Verlag New York.
- [Wickham 2016] Wickham, H. (2016). *tidyverse: Easily Install and Load 'Tidyverse' Packages*. R package version 1.0.0.

Análise de Características Comportamentais de Aplicações OpenMP para Redução do Consumo de Energia

Gabriel B. Moro, Lucas Mello Schnorr

¹Instituto de Informática – Universidade Federal do Rio Grande do Sul (UFRGS)
Caixa Postal 15.064 – 91.501-970 – Porto Alegre – RS – Brazil

{gbmoro, schnorr}@inf.ufrgs.br

Resumo. *Desempenho e consumo energético são requisitos fundamentais em sistemas de computação. Um desafio comumente encontrado é conciliar esses dois aspectos, buscando manter o mesmo desempenho, consumindo cada vez menos energia. Muitas técnicas possibilitam a redução do consumo de energia em aplicações paralelas, mas na maioria das vezes elas envolvem recursos encontrados apenas em processadores modernos ou um conhecimento amplo das características da aplicação e da plataforma alvo. Nesse trabalho propomos uma biblioteca que altera dinamicamente a frequência de processador, utilizando como base de conhecimento uma análise prévia das regiões limitadas pela memória. Os resultados apresentam uma redução de 1,89% no consumo de energia para aplicação Lulesh com cerca de 0,09% de sobrecarga no tempo de execução, quando comparamos nossa técnica com a política de processador Ondemand do Sistema Operacional Linux.*

Abstract. *Performance and energy consumption are fundamental requirements in computer systems. A very frequent challenge is to combine both aspects, searching to keep the high performance computing while consuming less energy. There are a lot of techniques to reduce energy consumption, but in general, they use modern processors resources or they require specific knowledge about application and platform used. In this paper, we propose a library that dynamically changes the processor frequency according to the application's computing behavior, using a previous analysis of its Memory-Bound regions. The results show a reduction of 1,89% in energy consumption for Lulesh application with an increase of 0,09% in runtime when we compare our approach against the governor Ondemand of the Linux Operating System.*

1. Introdução

Aplicações de alto desempenho são de extrema importância para a humanidade. Cada vez mais cientistas de diversas áreas necessitam de poder de computação para processar modelos complexos com o objetivo de avançar suas pesquisas. Além do desempenho dessas aplicações, o consumo de energia é uma preocupação constante em centros de supercomputadores, os quais fornecem máquinas potentes para executar aplicações que muitas vezes levam segundos, horas ou até mesmo dias em alguns casos. Nesse sentido, muitas pesquisas estão sendo realizadas para reduzir o consumo de energia tanto em nível de processador, trabalhando com diferentes configurações de frequência para o processamento (uso de *Dynamic Voltage and Frequency Scaling* - DVFS), como também a nível

de software, utilizando técnicas de caracterização para conhecer melhor o comportamento de aplicações, a fim de realizar otimizações para redução do consumo de energia.

Cada aplicação possui um perfil comportamental de execução, o que está relacionado com o tamanho de entrada, suas estruturas de dados, a quantidade e o tipo de cálculos realizados, entre outros aspectos. Esse perfil pode ser utilizado para agrupar as aplicações de acordo com o componente computacional que limita o desempenho da aplicação. Sendo assim, podemos ter aplicações cujo desempenho é limitado pelos acessos à memória (*Memory-Bound*), outras que são limitadas pelo desempenho computacional do processador (*CPU-Bound*) e enfim aquelas limitadas pelas operações de entrada e saída (*IO-Bound*). [Jesshope and Egan 2006] definem aplicações do tipo *Memory-Bound* como aquelas em que o desempenho aumenta à medida que for reduzido a taxa de *Cache misses* para o segundo nível de *Cache*. Já nas aplicações do tipo *CPU-Bound*, uma redução da taxa de *Cache misses* não necessariamente se transformaria em um melhor desempenho, pois o limitante é a espera pela CPU ou por IO (entrada/saída) e não a memória.

Embora seja possível classificar as aplicações nessas categorias, nem sempre uma aplicação terá um comportamento totalmente *Memory-Bound* ou *CPU-Bound*. O que pode ocorrer é que diferentes regiões de código de uma mesma aplicação podem apresentar uma espera mais marcante por um determinado recurso, seja ele de CPU, Memória ou IO. Aplicações como o *benchmark* Lulesh [Karlin et al. 2013], por exemplo, possuem diferentes limitantes computacionais em suas regiões paralelas. Algumas regiões tem baixa taxa de *misses* na *Cache* L2 e um alto IPC (*Instruction Per Cycle*), da mesma forma, existem outras em que o comportamento é equilibrado entre essas duas medidas (mais detalhes em [Moro and Schnorr 2017]).

A partir de uma caracterização comportamental das regiões *Memory-Bound* de uma aplicação, seria possível escolher regiões potenciais a serem executadas em uma frequência menor, com o objetivo de reduzir o consumo de energia. Nesse cenário, o principal desafio é a coleta de informações detalhadas durante a execução da aplicação, as quais possam fornecer as características de cada região da aplicação paralela.

O objetivo desse trabalho é propor uma metodologia de análise de desempenho que permita detectar o comportamento de regiões paralelas em aplicações OpenMP, através do uso de medidas como IPC (*Instruction Per Cycle*) e taxa de *Misses* na *Cache* L2. É proposto nesse trabalho um *Workflow* de análise para identificar quais regiões são *Memory-Bound*, a partir dessa investigação, o *Workflow* atua realizando um controle dinâmico de frequência de acordo com o comportamento da aplicação.

O presente artigo está organizado da seguinte maneira. Na Seção 2 é abordado sobre os tipos de aplicações paralelas, suas características e os principais conceitos de consumo de energia. A Seção 3 é apresentado o *Workflow* de investigação de regiões *Memory-Bound*, o qual é definido em duas fases, a primeira de investigação das regiões e a segunda que realiza o controle dinâmico de frequência. A Seção 4 apresenta os resultados do experimento em que comparamos a política *Ondemand* com a biblioteca proposta. A Seção 5 apresenta uma comparação de nossos trabalhos com o estado da arte. A Seção 6 encerra o artigo com as considerações finais e trabalhos futuros.

2. Comportamento de Aplicações Paralelas e Consumo de Energia

Segundo Diamond et al. [Diamond et al. 2011] existem três padrões de aplicação em computação de alto desempenho: as aplicações regulares que possuem computações e acessos à memória, os quais são previsíveis; as irregulares que possuem acessos aleatórios a memória, como também conseguem alterar dinamicamente suas estruturas de dados; e as aplicações de grafos que realizam acessos em diferentes localidades da memória, fazendo com que a localidade espacial da *Cache* seja menor. Nessas diferentes classes de aplicações é possível encontrar regiões paralelas que possuem uma espera maior pela memória e outras que são mais limitadas pela CPU.

Ao realizar a leitura de uma determinada informação, a aplicação realiza buscas nas memórias *Cache*. Não encontrando a informação, é necessário acessar a memória principal. O acesso à memória principal gera um custo maior de desempenho, visto que, a memória principal possui uma velocidade inferior a da memória *Cache*, pois são tecnologias diferentes para fins diferentes. Nisso, a taxa de acertos dessa busca interfere diretamente no desempenho de aplicações *Memory-Bound*, pois quanto menos vezes for necessário acessar a memória principal, maior será o desempenho da aplicação. Aplicações consideradas como *Memory-Bound* possuem uma grande parte de sua execução ociosa, interferindo no desempenho da aplicação como um todo [Subramanian 2015]. Geralmente esse comportamento é encontrado em aplicações SIMD (*Single Instruction Multiple Data*), tais como processamento de imagens, vídeos e assim por diante.

Diferente das aplicações *Memory-Bound*, as aplicações *CPU-Bound* são limitadas pela espera por CPU. Essas aplicações são aquelas que possuem o desempenho totalmente afetado pela velocidade em que o processador realiza as operações aritméticas solicitadas [Hutcheson and Natoli 2011]. Um bom exemplo deste tipo de aplicação é a sequência de Fibonacci, a qual solicita à CPU, a computação de várias instruções por ciclo, devido sua recursividade e demais cálculos realizados.

Segundo [Orgerie et al. 2014], o consumo de energia de recursos computacionais pode ser medido através de sensores de energia (gasto real) ou estimados a partir de modelos (estimativa teórica). Algumas ferramentas podem ser utilizadas para acessar esses sensores. Um exemplo de ferramenta que permite obter uma relação do consumo de energia total gasto pela aplicação é a Intel PCM, fornecendo o consumo de energia gasto pela CPU (considerando as memórias *Cache*) e memória principal [Silveira et al. 2016]. Outra maneira é estimar a energia utilizando modelos, o CACTI é um exemplo de ferramenta que permite especificar o modelo de memória utilizado pela plataforma de execução, a partir disso é possível obter uma estimativa de gasto de energia por instrução [Silveira et al. 2016].

Além de conhecer o quanto de energia a aplicação consome, vários trabalhos possuem propostas de redução no consumo energético, dentre os assuntos investigados, uma técnica amplamente utilizada para economizar energia a partir da redução da frequência de processador é a técnica DVFS (*Dynamic Voltage and Frequency Scaling*). Essa técnica permite reduzir a frequência de processador em determinados trechos de execução do programa, os quais possuem um comportamento mais *Memory-Bound* [Sueur and Heiser 2010].

Dynamic Voltage and Frequency Scaling (DVFS) é uma técnica muito utilizada em sistemas embarcados, *desktops* e também em *workstations* que executam aplicações de alto desempenho [Sueur and Heiser 2010]. Geralmente, essa técnica é utilizada em

conjunto com metodologias que definem o momento exato para a troca de frequência ocorrer. Dependendo do tipo de aplicação e da metodologia empregada, a técnica DVFS pode ser vantajosa, tanto em termos de consumo de energia, quanto em desempenho. O *driver* CPUFreq é um exemplo de utilização da técnica DVFS no Sistema Operacional Linux, ele fornece uma interface para alterar a frequência dos processadores e utilizar políticas pré-implementadas [Brodowski et al. 2013]. O *framework* disponibiliza as seguintes políticas (modos): *Performance* - utiliza a maior frequência disponível; *Power-save* - utiliza a mais baixa frequência; *Ondemand* - escolhe a frequência de acordo com o uso da CPU; *Conservative* - funciona como o *Ondemand*, mas o ajuste é mais refinado; e *Userspace* - utiliza a frequência informada pelo usuário ou por algum programa executado como super usuário [Brodowski et al. 2013].

3. Workflow de Investigação de Regiões Memory-Bound

A Figura 1 apresenta uma visão geral do *Workflow* proposto. O programa paralelo escrito em OpenMP é executado com uma biblioteca externa que, permite a observação e o controle de frequências do processador. Essa biblioteca com dois comportamentos permite a execução do *Workflow* em duas fases.

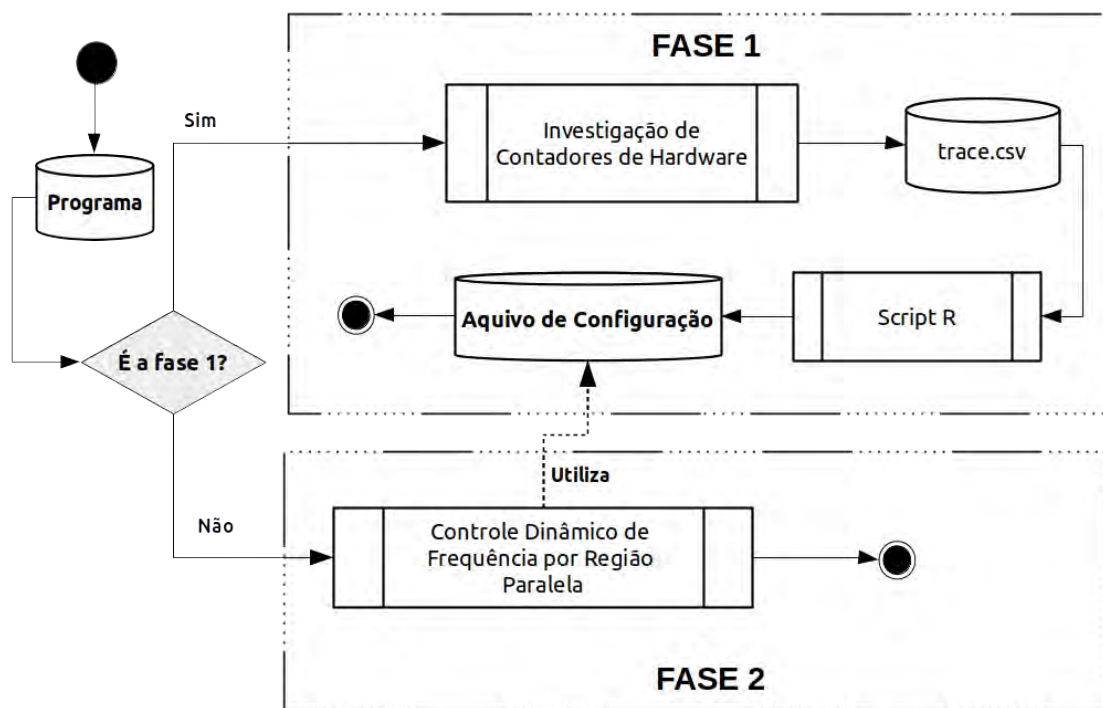


Figura 1. Visão Geral do *Workflow* de investigação e controle de frequência, com duas fases: (1) contadores de hardware são coletados e os dados interpretados a partir de scripts em R que permitem gerar um arquivo de configuração para a fase seguinte; (2) baseado em um arquivo de configuração que associa regiões paralelas a frequências de processador específicas o programa é executado com o objetivo de reduzir o consumo de energia.

Na primeira fase (parte superior na figura), o programa paralelo em OpenMP, durante sua execução, terá seu comportamento registrado. Esse comportamento contém nos

registros os contadores de hardware relevantes para identificar regiões de código limitadas pelos acessos à memória. Como resultado intermediário da primeira fase, obtém-se um arquivo de rastro que pode ser potencialmente grande dependendo da quantidade de regiões paralelas, da duração do programa e do tamanho do problema. Os rastros tem, em cada evento, o identificador da região paralela OpenMP (nome do arquivo onde aparece e número da linha), os contadores de hardware, o início e a duração da região paralela. Caso uma região paralela seja executada várias vezes, teremos no rastro várias ocorrências do seu comportamento.

Ainda na primeira fase, esse rastro é processado através de um *script* R que, faz uma análise de dados para identificar quais regiões paralelas tem características do estilo *Memory-Bound*. Como conclusão da primeira fase, obtemos um arquivo de configuração que será realimentado para a biblioteca externa que, muda seu comportamento para a fase seguinte. Esse arquivo de configuração tem um mapeamento das regiões paralelas para a frequência do processador apropriada, mediante edição manual pelo analista. Após isso na segunda fase, e de posse do arquivo de configuração, a biblioteca externa, ao invés de coletar informações, altera a frequência do processador durante a execução do programa.

O resto desta seção apresenta maiores detalhes de cada uma das fases assim como um detalhamento da implementação da biblioteca *LibNina* que implementa este *work-flow* e permite a observação/controle de programas paralelos escritos em OpenMP.

3.1. Fase 1: Obtenção da Assinatura Computacional

A partir de contadores de hardware é possível obter medidas como o total de ciclos de CPU, acessos à *Cache* ou memória principal, quantidade de instruções de tipo *load*, entre outras. Como em nossa abordagem desejamos conhecer quais são as regiões da aplicação que são limitadas pela memória, utilizamos contadores de hardware para obter o total de instruções executadas, quantidade de ciclos de CPU, total de acessos e *misses* na *Cache* L2.

De forma geral, ao executar aplicações OpenMP, o processo principal dispara várias *threads* de processamento. Portanto, precisamos conhecer o comportamento das regiões executadas por cada *thread*, desprezando a parte sequencial do programa. Na primeira fase, os contadores de hardware são coletados por *thread*, a fim de que no final, agrupando todos os contadores por *thread*, seja possível identificar as regiões mais limitadas pela memória. O IPC (*Instruction per Cycle*) permite mapear quantas instruções são realizadas por ciclo de processador, essa porcentagem é extremamente importante para reconhecer em aplicações paralelas, o comportamento *CPU-Bound*. Já a taxa de *misses* na *Cache* L2 permite mapear a porcentagem de insucesso nas requisições realizadas pela aplicação à memória.

Na execução da primeira fase não é necessário executar a aplicação com uma entrada grande, pois quanto maior a entrada, mais tempo a aplicação levará para processar e maior será o rastro a ser analisado. A partir disso, o resultado da primeira fase é um arquivo de configuração para a biblioteca. Ele é obtido por análise manual, utilizando como entrada o rastro de comportamento. As frequências de processador que deverão ser utilizadas serão definidas para cada região paralela. A partir dos valores dos contadores de hardware, estabelecemos uma relação entre as duas métricas (IPC e taxa de *Misses* na *Cache* L2). As regiões identificadas como *Memory-Bound*, por exemplo, podem receber

uma frequência de processador menor do que as demais.

3.2. Fase 2: Controle Dinâmico da Frequência por Região Paralela

O controle dinâmico da frequência é realizado de acordo com o arquivo de configuração gerado na fase anterior. As frequências a serem configuradas são armazenadas em uma estrutura de dados, ao entrar em uma região é buscado nessa estrutura de dados, a frequência de processador associada a sua respectiva região. Como a troca de frequência é custosa em termos de desempenho (devido a latência da troca da frequência uma vez que o comando é dado pelo Sistema Operacional), realizamos ela apenas quando necessário, levando em consideração a duração da região paralela. A troca de frequência ocorre sempre ao entrar na região paralela, ao sair não é realizada nenhuma configuração de frequência, visto que ao entrar em uma nova região, uma nova frequência será atribuída para a mesma.

3.3. Implementação do workflow na LibNina: DVFS para Regiões Paralelas

Uma biblioteca de código aberto¹ chamada *LibNina* foi desenvolvida para implementar o *Workflow* em duas fases. A aplicação paralela com regiões OpenMP é instrumentada automaticamente com um compilador fonte-para-fonte (*source-to-source*) chamado *Opari2*. A instrumentação consiste em chamadas a funções específicas que, são implementadas pela *LibNina* de forma que, elas possam observar/controlar a execução da aplicação OpenMP. Dentre os múltiplos pontos de instrumentação possíveis, a *LibNina* adota especificamente os pontos de código onde um *Parallel begin* e um *Parallel end* são executados. Eles representam o início e o fim de blocos de código marcados por `#pragma parallel` da especificação OpenMP.

No momento que cada região paralela começa e termina de ser executada, nos pontos de instrumentação descritos acima, funções específicas da *LibNina* são chamadas para atuarem conforme o *Workflow*. O mapeamento das regiões paralelas é possível tendo em vista que o compilador *Opari2* instrumenta o código criando identificadores únicos para cada região paralela, contendo o nome do arquivo, e a linha de código de início e fim de cada região paralela. A *LibNina* contém dois componentes principais, o primeiro destinado a utilizar a biblioteca *PAPI* para coletar os contadores de hardware (na primeira fase do *Workflow*), o segundo destinado a alterar a frequência do processador utilizando a biblioteca *CPUFreq* (na segunda *fase*).

4. Resultados

A aplicação utilizada nos experimentos é o *benchmark* Lulesh [Karlin et al. 2013], o qual representa uma simulação computacional de modelagem hidrodinâmica. Essa simulação reproduz uma onda de explosão em um espaço confinado, utilizando como base o método de Sedov [LLNL 2018]. A aplicação é reconhecida por ter características *Memory-Bound*. Por essa razão, ela foi utilizada pela perspectiva de ganhos potenciais, ao executar tais regiões em uma frequência de processador mais baixa. Para fins de comparação, adotamos o gerenciador de frequência *Ondemand* do pacote *CPUFreq* do Sistema Operacional Linux que, realiza a troca de frequência tendo como base a taxa de uso do processador, sem qualquer relação com a aplicação executada.

¹Código disponível em <https://github.com/tido4410/libnina>

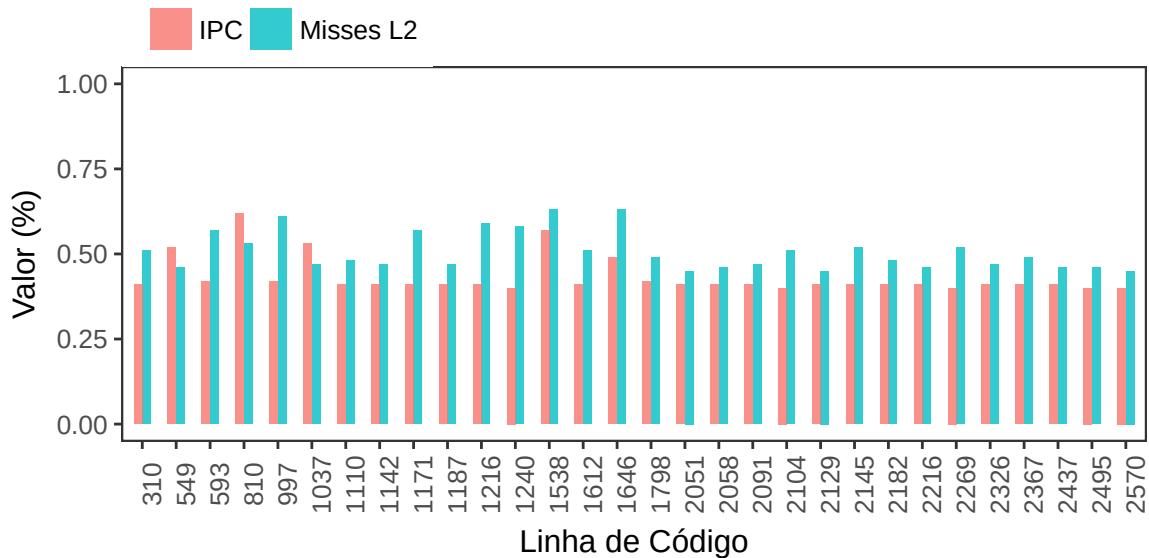


Figura 2. Porcentagem de IPC e Misses na Cache L2 por Linha de Código.

A plataforma utilizada para executar os experimentos é uma *Workstation* de 126G de memória RAM e dois processadores Intel(R) Xeon(R) CPU E5-2650, cada um com 10 núcleos físicos que possuem tecnologia *HyperThreading*. A máquina permite a frequência máxima de 2.30GHz e mínima de 1.2GHz. O Sistema Operacional da plataforma é o Ubuntu 16.04LTS com o framework CPUFreq instalado e suporte as políticas *On-demand*, *Conservative*, *Performance*, *Powersave* e *Userspace*. Todas as combinações de configuração para os experimentos foram executadas utilizando 20 threads fixas nos cores físicos, com a ausência do uso de *HyperThreading* que foi desativado explicitamente. Para lidar com a variabilidade natural do sistema computacional, os experimentos foram replicados e médias aritméticas simples são calculadas e relatadas. Adotamos a ferramenta AppPowerMeter² para realizar as medições de consumo de energia.

A primeira fase do *Workflow* é executar a aplicação Lulesh com uma entrada menor, a fim de avaliar o comportamento de suas regiões paralelas, investigando alguns contadores de hardware. A execução utiliza apenas 10 elementos de entrada, dessa forma é possível obter um tamanho de rastro adequado para análise. No experimento utilizamos quatro contadores: *PAPI_L2_TCA*, *PAPI_L2_TCM*, *PAPI_TOT_INS* e *PAPI_TOT_CYC*. Esses contadores permitem obter as métricas IPC (*Instruction per Cycle*) e a taxa de *Misses* na *Cache L2*. O resultado da primeira fase pode ser visto na Figura 2 que, apresenta a investigação das duas medidas por região paralela do código da aplicação Lulesh. Adotamos como identificador da região paralela o número da linha que começa aquela região (o local do `#pragma omp parallel`), sendo utilizada portanto no eixo horizontal da figura (o código é contido em um único grande arquivo fonte).

Na Figura 2 é possível visualizar a relação entre as duas medidas obtidas: o IPC e a taxa de *Misses* na *Cache L2*. O eixo “x” apresenta o identificador (Linha de Código) na região paralela onde a medida é obtida. A coleta dos contadores de hardware é realizada em dois momentos, ao entrar e sair de uma região, com isso é possível obter as medidas

²Disponível em <https://github.com/kentcz/rapl-tools>

de IPC e taxa de *Misses* na *Cache* L2 para cada região. Com essa investigação, adotamos várias estratégias diferentes para economia de energia. Cada estratégia se reflete em um arquivo de configuração diferente para a segunda fase de nosso *Workflow*. Dentre as estratégias, as mais proeminentes são: (a) o uso de frequência máxima nas regiões onde o IPC é maior do que 50%, sendo que nas demais localizações o uso de frequência mínima; (b) o uso de frequência máxima em todas localizações exceto em regiões onde a taxa de misses é maior do que 50% (nessas regiões o uso de frequência mínima); (c) e a combinação das duas configurações anteriores, uso de frequência baixa quando o IPC é menor do que 50% e a taxa de misses é maior do que 50%, nas demais regiões o uso de frequência máxima. Essas configurações possibilitaram identificar uma sobrecarga na troca de frequência, presente na implementação da biblioteca *LibNina*. A razão desta sobrecarga tem origem na própria aplicação *Lulesh* que, possui regiões cuja assinatura computacional é bastante diferente (Figura 2). Isso obriga a *LibNina* a adaptar a frequência do processador em mais ocasiões em um intervalo de tempo curto, gerando um custo extra no consumo de energia. Com isso, nós identificamos que, em regiões de curta duração (tempo), caso seu comportamento solicitasse uma troca de frequência, seria mais vantajoso executar tal região sem a troca de frequência, a fim de evitarmos o sobre-custo. Os melhores resultados foram obtidos com esta estratégia adaptada que combina a assinatura computacional (contadores de hardware) com a duração das regiões paralelas.

A Figura 3 apresenta o consumo de energia (em X) em função da política de gerenciamento da frequência do Linux *Ondemand* e da biblioteca proposta (em Y), para dois tamanhos de entrada: 160 e 180. Os dados do gráfico foram obtidos a partir de uma bateria experimental que inclui cinco replicações de cada configuração experimental totalizando vinte execuções que são executadas em ordem aleatória, cada uma com uma duração média de 1 hora. A partir dos resultados, é possível verificar que, ao utilizar 160 elementos, o uso da biblioteca reduziu cerca de aproximadamente 1.44% o consumo de energia, comparado com o modo *Ondemand*. Já utilizando 180 elementos, ao utilizar a biblioteca, o consumo de energia reduziu em cerca de 1.89%. Embora pequenos, os ganhos são consistentes tendo em vista a longa duração (≈ 1 hora) dos experimentos.

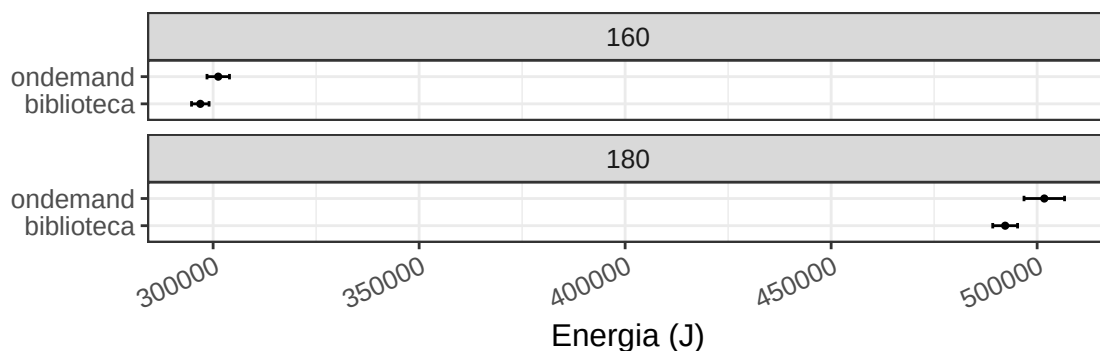


Figura 3. Lulesh - Consumo de Energia - Governador Ondemand vs Biblioteca.

A Figura 4 apresenta o consumo energético (em Y) correlacionado com o tempo de execução (em X) da aplicação *Lulesh* quando esta é executada com o tamanho da entrada de 180 em dois cenários: o caso *Ondemand*, utilizado como referência, e a biblioteca (nossa abordagem). O uso da biblioteca possibilita um ganho no consumo de energia, conforme visto anteriormente, mas em termos de tempo de execução, o modo *Ondemand*

é aproximadamente 0,09% mais rápido. No gráfico 4 é possível visualizar essa relação entre o consumo de energia, como também a perda correspondente no tempo de execução, da biblioteca sobre a política *Ondemand*.

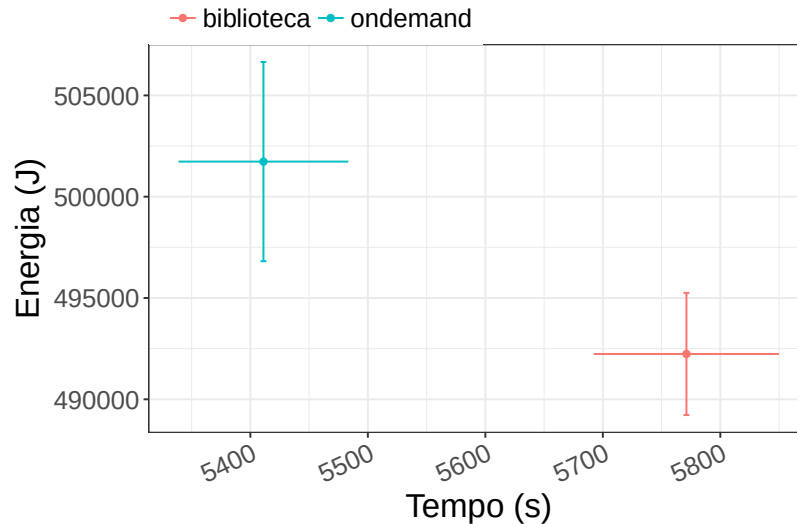


Figura 4. Lulesh - Tamanho 180 - Tempo e Energia.

5. Trabalhos Relacionados

Molka et al. [Molka et al. 2017] apresentam uma abordagem que investiga quais contadores de hardware devem ser utilizados para se obter uma melhor estimativa do uso de componentes da hierarquia de memória. Para isso, os autores utilizam um conjunto de *benchmarks* específicos, os quais tem por objetivo o “estresse” individual dos componentes do sistema de memória (*Caches* e Memória Principal). Os *benchmarks* utilizados no trabalho permitem configurações como: restrição para medir apenas uma única CPU, configuração de uso de apenas 50% da *Cache* L1, diminuição do tamanho do último nível de *Cache* para menos de 10 vezes, dimensionamento de *loads* e *stores*, entre outras configurações para melhor examinar os componentes de memória utilizando apenas um processador, como também recursos compartilhados no cenário de uma plataforma *multi-core*. Os ciclos de processador são divididos primeiramente em dois grupos, os ciclos de produtividade que são aqueles onde ocorre o processamento útil de instruções e os ciclos de espera que existem em função da latência gerada pelo acesso à memória ou por outras razões. Nos ciclos *Memory-Bound*, os autores identificaram duas situações, a primeira quando os ciclos de processador eram limitados pela espera por memória e a segunda situação quando os ciclos eram limitados pela largura de banda da memória. Os autores mapearam os ciclos ativos e os ciclos de espera em uma aplicação OpenMP do *benchmark* NAS, após isso é realizado uma redução de frequência de processador nos momentos onde a aplicação possuía ciclos de espera, através disso, os autores reduzem o consumo de energia em cerca de 4,16% com uma penalidade de 0,2% no tempo de execução.

Millani e Schnorr [Millani and Schnorr 2016] investigam a relação entre tempo e energia, ao adotar frequências de processador por região paralela. Primeiramente, os autores adotam um projeto experimental do tipo *screening* para identificar as regiões paralelas mais propícias para redução do consumo energético, sem penalidade em tempo de

execução. Nesta fase, as regiões paralelas são assinaladas com duas possíveis frequências (baixa ou alta), as quais devem ser suportadas pela plataforma alvo. Os autores utilizam as técnicas de Análise de Variância (ANOVA) e análise de efeitos principais (*Main Effects Analysis*) para identificar as regiões paralelas potenciais, aquelas que possuem maior impacto no consumo de energia e menor impacto no tempo de execução. Dessa forma, em uma nova execução do programa, apenas as regiões específicas (potenciais) sofrerão a alteração de frequência de processador. Dentre os resultados obtidos, os autores conseguiram uma redução de aproximadamente 9,27% no consumo de energia para o *benchmark* MiniFE, sem perdas significativas de desempenho. Por outro lado, a abordagem é totalmente dependente de instrumentação de código, exigindo um conhecimento prévio da aplicação ao atribuir a frequência de processador para as regiões paralelas, sendo uma desvantagem em cenários onde a aplicação é completamente desconhecida pelo analista de desempenho.

Existem trabalhos como o de Wang et al. [Wang et al. 2015] que adotam uma abordagem voltada mais às técnicas de gerenciamento de potência, as quais são disponibilizadas pelo próprio hardware. Ao invés de utilizar DVFS para alternar a frequência de processador a ser utilizada, os autores utilizam *CPU Clock Modulation* para definir a frequência adequada para os diferentes *loops* da aplicação. Dessa forma, é possível visualizar ganhos ao utilizar execuções com múltiplas frequências que são selecionadas com *CPU Clock Modulation*. Ao entrar em cada laço do programa, a frequência de processador é reduzida e ao sair a mesma é aumentada. Segundo os autores, a técnica pode em alguns casos acarretar um aumento no tempo de execução, mas um ganho considerável no consumo de energia. O melhor resultado apresentado pelos autores é de uma redução média de 8,6% no consumo de energia, com a penalidade inferior a 1,5% no tempo de execução. Por utilizar *CPU Clock Modulation*, a metodologia só pode ser aplicada em plataformas com processadores recentes.

Laurenzano et al. [Laurenzano et al. 2011] apresenta uma técnica que permite selecionar frequências de processador adequadas por laço de repetição da aplicação. Na caracterização das aplicações, os autores utilizam a taxa de *hit* na *Cache* L1, L2 e L3; a taxa de operações com ponto flutuante para o número de operações de memória; e a taxa de operações com números inteiros por operações com ponto flutuante. Os autores formam uma base de conhecimento de acordo com inúmeras execuções, caracterizando o consumo de energia gasto de acordo com padrões de execução e frequências de processador executadas anteriormente, obtidas por rastro. Dessa maneira é possível buscar nesse conjunto de cobertura, qual a melhor frequência para determinado padrão de aplicação. O melhor caso apresentado pelos autores é uma redução de aproximadamente 10,6% no consumo de energia, utilizando determinada carga de trabalho para uma arquitetura de 32 cores AMD Opteron 8380. Existem basicamente duas diferenças com nosso trabalho: (1) deve ser construída uma base de conhecimento baseada em código artificial; (2) a base deve ser consultada em tempo de execução para selecionar a frequência de processador mais apropriada, utilizando métricas de desempenho coletadas dinamicamente.

Em contrapartida, Shafik et al. [Shafik et al. 2015] apresentam uma abordagem para redução do consumo de energia utilizando políticas de hardware, tais como DVFS e *Dynamic Concurrency Throtling* (DCT). A abordagem investiga o desempenho da parte sequencial e paralela do programa, gerando anotações que serão utilizadas como uma

espécie de marcadores, os quais descrevem o desempenho do programa. Esses marcadores são posteriormente incorporados a biblioteca OpenMP, como modificações na *libgomp*. Através de estatísticas obtidas por contadores de hardware em intervalos regulares são aplicadas as técnicas DVFS e DCT, tanto na parte sequencial como na parte paralela da aplicação. Dentre os resultados, os autores apresentam uma redução de aproximadamente 17% no consumo de energia. Como a técnica proposta por [Wang et al. 2015], a abordagem de [Shafik et al. 2015] depende de recursos de hardware, os quais podem ser encontrados apenas em processadores mais recentes.

Dentre os trabalhos analisados é perceptível que existem várias formas de caracterizar uma aplicação paralela, sendo pelo uso de contadores de hardware, medidas de tempo de computação/comunicação, intervalos de tempo específicos em que a CPU fica ociosa e assim por diante. Algumas abordagens utilizam recursos disponibilizados pelo próprio hardware, o que possibilita melhores resultados na maioria dos casos. Vale salientar que, a maior parte das abordagens que reduzem o consumo de energia acarretam sobrecargas no tempo de execução. Na maioria dos casos é possível observar que, os trabalhos que utilizam técnicas fornecidas pelo próprio hardware resultam em maiores ganhos de energia. Além do que, se limitam a plataformas específicas ou processadores recentes. Nesse contexto, o presente trabalho propõe uma abordagem em formato de *Workflow* que pode ser aplicada em qualquer plataforma que possua suporte a política *Userspace* do *driver* CPUFreq. A abordagem proposta não requer nenhum conhecimento prévio sobre a aplicação alvo, assim o analista de desempenho pode utilizá-la em qualquer aplicação OpenMP.

6. Considerações Finais

O principal objetivo desse trabalho foi o de projetar e implementar um *Workflow* de investigação de regiões paralelas em aplicações OpenMP. Organizada em duas fases, a partir dos contadores de hardware (fase 1), a metodologia propõem a alteração da frequência de execução das regiões (fase 2) para uma configuração adequada a sua assinatura computacional: caso a região paralela seja limitada pela memória, reduz-se a frequência de execução do processador naquele trecho. A abordagem foi implementada na biblioteca LibNina, de código aberto, a qual utiliza informações a respeito do comportamento da aplicação fornecida pela primeira fase e depois altera a frequência de processador do programa de acordo com o comportamento respectivo de cada região paralela, alterando entre frequência máxima e mínima. Os resultados obtidos com a aplicação *Lulesh* indicam um ganho de 1,89% no consumo de energia sobre a política *Ondemand* do Linux, utilizada como referência. Vale salientar que, ocorreu um aumento no tempo de execução, cerca de 0,09%, no pior caso testado. Acreditamos que esse aumento no tempo de execução esteja relacionado a sobrecarga imposta pela implementação da biblioteca e aos frequentes comandos de adaptação da frequência do processador.

Como trabalhos futuros, pretendemos investigar alternativas para reduzir o custo da troca de frequência em regiões paralelas de curta duração, as quais possuem uma duração temporal igual ou menor ao tempo gasto pela operação de troca de frequência. Nesse cenário, a sobrecarga da operação de troca de frequência não se justificaria e poderia ser amenizada. Também, pretendemos analisar outros contadores de hardware, como por exemplo, quantidade de *loads* e *stores* realizados pela aplicação em cada região paralela. Como a LibNina suporta a configuração de quais contadores de hardware utilizar,

essa investigação seria facilitada pela biblioteca.

Agradecimentos

Nós agradecemos os projetos FAPERGS/Inria ExaSE, FAPERGS 16/2551-0000 354-8 (PPP 2014), FAPERGS 16/2551-0000 488-9 (PRONEX 2014), CNPq 447311/2014-0, e o programa H2020 da União Europeia e o MCTI/RNP através do projeto HPC4E, identificador 689772.

Referências

- [Brodowski et al. 2013] Brodowski, D., Golde, N., Wysocki, R. J., and Kumar, V. (2013). Linux cpufreq governors – information for users and developers. Technical report, Linux.
- [Diamond et al. 2011] Diamond, J., Burtcher, M., McCalpin, J. D., Kim, B.-D., Keckler, S. W., and Browne, J. C. (2011). Evaluation and optimization of multicore performance bottlenecks in supercomputing applications. In *Performance Analysis of Systems and Software (ISPASS), 2011 IEEE International Symposium on*, pages 32–43. IEEE.
- [Hutcheson and Natoli 2011] Hutcheson, A. and Natoli, V. (2011). Memory bound vs. compute bound: A quantitative study of cache and memory bandwidth in high performance applications. Technical report, Stone Ridge Technology.
- [Jesshope and Egan 2006] Jesshope, C. and Egan, C. (2006). *Advances in Computer Systems Architecture: 11th Asia-Pacific Conference, ACSAC 2006, Shanghai, China, September 6-8, 2006, Proceedings*. LNCS sublibrary: Theoretical computer science and general issues. Springer.
- [Karlin et al. 2013] Karlin, I., Keasler, J., and Neely, R. (2013). Lulesh 2.0 updates and changes. Technical Report LLNL-TR-641973, LLNL.
- [Laurenzano et al. 2011] Laurenzano, M. A., Meswani, M., Carrington, L., Snively, A., Tikir, M. M., and Poole, S. (2011). Reducing energy usage with memory and computation-aware dynamic frequency scaling. In *European Conference on Parallel Processing*, pages 79–90. Springer.
- [LLNL 2018] LLNL (2018). Hydrodynamics Challenge Problem, Lawrence Livermore National Laboratory. Technical Report LLNL-TR-490254, LLNL.
- [Millani and Schnorr 2016] Millani, L. F. and Schnorr, L. M. (2016). Computation-aware dynamic frequency scaling: Parsimonious evaluation of the time-energy trade-off using design of experiments. In *European Conference on Parallel Processing*, pages 583–595. Springer.
- [Molka et al. 2017] Molka, D., Schöne, R., Hackenberg, D., and Nagel, W. E. (2017). Detecting memory-boundedness with hardware performance counters. In *Proceedings of the 8th ACM/SPEC on International Conference on Performance Engineering*, pages 27–38. ACM.
- [Moro and Schnorr 2017] Moro, G. B. and Schnorr, L. M. (2017). Detecting memory-bound parallel regions to improve the energy-efficiency of applications. In *XIV Workshop de Processamento Paralelo e Distribuído*, pages 10–13.

- [Orgerie et al. 2014] Orgerie, A.-C., de Assuncao, M. D., and Lefevre, L. (2014). A Survey on Techniques for Improving the Energy Efficiency of Large-scale Distributed Systems. *ACM Comput. Surv.*, 46(4):47:1–47:31.
- [Shafik et al. 2015] Shafik, R. A., Das, A., Yang, S., Merrett, G., and Al-Hashimi, B. M. (2015). Adaptive energy minimization of openmp parallel applications on many-core systems. In *Proceedings of the 6th Workshop on Parallel Programming and Run-Time Management Techniques for Many-core Architectures*, pages 19–24. ACM.
- [Silveira et al. 2016] Silveira, D. S., Moro, G. B., Cruz, E. H. M., Navaux, P. O. A., Schnorr, L. M., and Bampi, S. (2016). Energy Consumption Estimation in Parallel Applications: an Analysis in Real and Theoretical Models. In *WSCAD 2016 - XVII Simpósio em Sistemas Computacionais de Alto Desempenho*, pages 134–145.
- [Subramanian 2015] Subramanian, L. (2015). Providing high and controllable performance in multicore systems through shared resource management. *arXiv preprint arXiv:1508.03087*.
- [Sueur and Heiser 2010] Sueur, E. L. and Heiser, G. (2010). Dynamic voltage and frequency scaling: the laws of diminishing returns. In *Proceedings of the 2010 international conference on Power aware computing and systems*, pages 1–8.
- [Wang et al. 2015] Wang, W., Porterfield, A., Cavazos, J., and Bhalachandra, S. (2015). Using per-loop cpu clock modulation for energy efficiency in openmp applications. In *Parallel Processing (ICPP), 2015 44th International Conference on*, pages 629–638. IEEE.

Strategies to Improve the Performance and Energy Efficiency of Stencil Computations for NVIDIA GPUs

Pablo José Pavan¹, Matheus da Silva Serpa¹, Víctor Martínez¹,
Edson Luiz Padoin^{1,2}, Jairo Panetta³, Philippe O. A. Navaux¹

¹ Informatics Institute - UFRGS - Porto Alegre - RS - Brazil

{pablo.pavan, msserpa, victor.martinez, navaux}@inf.ufrgs.br

²Department of Exact Sciences and Engineering - UNIJUI - Ijuí - RS - Brazil

padoin@unijui.edu.br

³Computer Science Division ITA - São José dos Campos – SP – Brazil

jairo.panetta@gmail.com

Abstract. *Energy and performance of parallel systems are an increasing concern for new large-scale systems. Research has been developed in response to this challenge aiming the manufacture of more energy efficient systems. In this context, we improved the performance and achieved energy efficiency by the development of three different strategies which use the GPU memory subsystem (global-, shared-, and read-only- memory). We also develop two optimizations to use data locality and use of registers of GPU architecture. Our developed optimizations were applied to GPU algorithms for stencil applications achieve a performance improvement of up to 201.5% in K80 and 264.6% in P100 when used shared memory and read-only cache respectively over the naive version. The computational results have shown that the combination of use read-only memory, the Z-axis internalization of stencil application and reuse of specific architecture registers allow increasing the energy efficiency of up to 255.6% in K80 and 314.8% in P100.*

1. Introduction

Different applications in areas such as computational physics, weather forecast, oil exploration, climate modeling and atomic simulation require high performance with acceptable power consumption. These scientific applications use stencil computations that include both implicit and explicit partial differential equations (PDE) solvers [Datta et al. 2008]. Besides the stencils scientific importance, they are interesting as a performance and energy consumption evaluation benchmark because they have abundant parallelism and low computational intensity, offering opportunities for on-chip parallelism and challenges for associated memory systems [Datta et al. 2008].

Scientific simulations may consume weeks and most of these time is spent in stencil computations [de la Cruz and Araya-Polo 2011]. Continuous changes in the microprocessors fabrication process have increased the performance of its products and influenced state-of-the-art HPC systems. However, this exponential increase in performance also leads to an exponential growth in power demand [Laros et al. 2009, Dong et al. 2010, Padoin et al. 2013a]. Reductions in the applications execution time are also relevant for energy consumption because energy is saved when hardware resources are used for a shorter time.

Memory performance is particularly important for stencil computations once they are typically memory-bound. For this reason, optimize the memory accesses is one of the keys to reducing the performance and energy consumption. In this paper, we improved the performance and energy efficiency of stencil applications by better use of the memory subsystem. We focus on analyzing the use of different Graphics Processing Units (GPU) memories aiming to improve performance, energy consumption, and energy efficiency. The main contributions of this paper are:

- propose of three strategies which use different GPU memories (*global*-, *shared*-, and *read-only*- memory);
- extend these strategies by use data locality and registers optimization;
- performance evaluation of these optimizations on Tesla K80 and Tesla P100 GPUs.

The remaining sections of this paper are organized as follows. Section 2 discusses the related work. In Section 3 we present the stencils application, the details of the optimizations we developed and the evaluation methodology. Also, in Section 4 we address the results obtained from the experiments. Finally, the Section 5 emphasizes the scientific contribution of the work and notes several challenges that we can address in the future.

2. Related Work

Several studies evaluated the stencil performance and energy efficiency in CPUs and GPUs. Despite that, processors and accelerators remain as the component with the highest power demand of the systems [Feng et al. 2005]. GPUs are made aiming massively parallel processing and to achieve this they use hundreds of processing units working together. These characteristics lead to its superior energy efficiency when compared with CPUs [Padoin et al. 2013b].

Micikevicius *et al.* [Micikevicius 2009] compared the performance of a stencil ported from CPU to GPU. Their version of the stencil running in a GPU achieved an order of magnitude higher than running in a contemporary CPU. They conclude that is possible to improve their results by the usage of shared memory to reduce communication overhead. Bauer *et al.* [Bauer et al. 2011] showed that the main bottleneck in GPU applications is related to the memory subsystem. To reduce its impact, they used DMA warps to improve memory transfer between on-chip and off-chip memories. They achieved a speedup up to $3.2\times$ on several kernels from scientific applications.

Schäffer and Fey [Schäfer and Fey 2011] evaluate a set of algorithms on Fermi GPUs. They evaluate micro-benchmarks using shared memory and found that using only the L1 cache creates a problem for its limited throughput. Also, the L2 cache is not a good option because of cache blocking. They conclude that a new alternative to use shared memory was needed to overcome communication bottleneck. Falch and Elster [Falch and Elster 2014] proposed the usage of a manually managed cache to combine the memory from multiple threads. Using their technique, they achieved a speedup of up to 2.04 in a synthetic stencil. They concluded that manual caching is an effective approach to improve memory access and that applications with regular access patterns are suitable to implement their technique.

Zhou *et al.* [Zhou et al. 2016] points that the use of GPUs enables considerable gains in performance compared to using CPU. They have applied GPUs successfully in many computations and memory intensive realms due to its superior performances in the float-pointing calculation, memory bandwidth, and power consumption. The results obtained show a speedup of up to 50 times using GPU algorithm rather than CPU algorithm. In

similar works, Zhou *et al.* [Zhou et al. 2012] obtained a speedup between 10 and 15 times using a GPU rather than CPU.

Xue *et al.* [Xue et al. 2015] also make comparisons between GPU and CPU implementation. They obtained a speedup up to 18 times in the GPU-based implementation of a time-reversal imaging micro-seismic event location. Also, Nikitin *et al.* [Nikitin et al. 2012] obtained average speedup up to 46 times using GPU for compared to CPU for processing a synthetic seismic data set (data compression, de-noising, and interpolation).

Maruyama and Aoki [Maruyama and Aoki 2014] present a method for stencil computations on the NVIDIA Kepler architecture that uses shared memory for better data locality combined with warp specialization for higher instruction throughput, their method achieves approximately 80% of the value from roof line model estimation. Hamilton *et al.* [Hamilton et al. 2015] investigate the computational performance of GPU-based stencil operations using stencils of varying shape and size (ranging from seven to more than 450 points in size). They found that using an NVIDIA K20 GPU, data movement, rather than computing, was the bottleneck, and as such, the performance obtained can be attributed to the effects of the L2 and texture caches on the card.

Nasciutti and Panetta [Nasciutti and Panetta 2016] did a performance analysis of 3D stencils on GPUs focusing on the proper use of the memory hierarchy. They conclude that the preferred code is the combination of read-only cache reuse, inserting the Z loop into the kernel and register reuse. Different to other approaches that allocate workload on CPU and GPU architectures, or works that use GPUs to achieve considerable performance gains when compared to traditional CPU architecture. Our goal is to increase performance and energy efficiency of stencil application applying methods and optimization to use different memory levels of the GPUs.

3. Evaluation Methodology

In this section, we show the stencil model we use as a case study. We also present the GPUs architectures, the optimizations evaluated and the experimental methodology.

3.1. Stencil Computations

The computational performance of GPU-based stencils has great scientific importance as it is used in many areas of scientific computing. Stencil computations are present from a simple Jacobi iterations until extremely complex solutions of non-linear Partial Differential Equations (PDE) [Dubey 2014]. A stencil application calculates the value of one point of the grid in the current iteration using the value of this same point and its neighbors in the previous iteration [Datta et al. 2008].

3.1.1. Fletcher Model in Isotropic Acoustic Wave Propagation

The modeling simulates the collection of data in a seismic survey, as in Figure 1. At intervals of, equipment coupled to the ship emits waves that reflect and refract on changes of the medium in the subsoil. Eventually, these waves return to the surface of the sea, being collected by specific microphones (geophones) coupled to cables towed by the ship. The set of signals received by each geophone over time constitutes a seismic trace. For each wave emission, the seismic traces of all cable geophones are recorded. The ship continues to sailing and emits signals over time.

Acoustic wave propagation approximation is the current backbone for seismic imaging tools. It has been extensively applied to imaging potential oil and gas reservoirs beneath

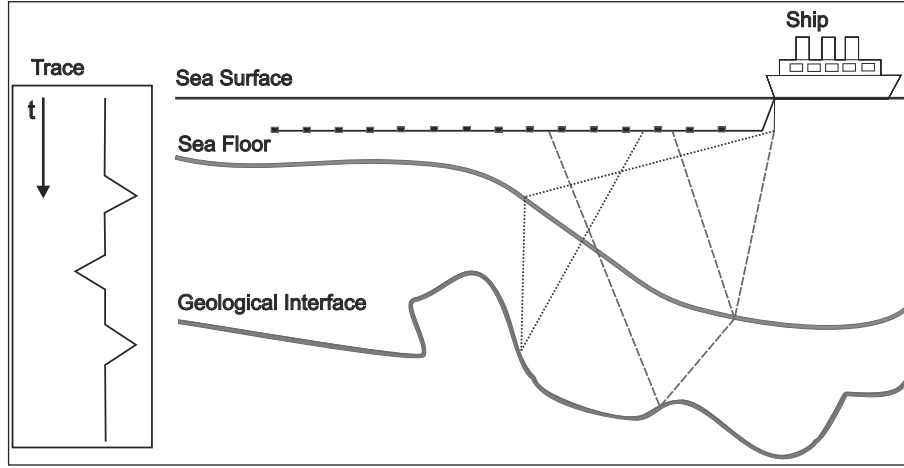


Figure 1: Data collection (traces) in maritime seismic survey.

salt domes. We consider the model formulated by the isotropic acoustic wave propagation under Dirichlet boundary conditions over a finite 3D rectangular domain, prescribing to all boundaries, and the isotropic acoustic wave propagation. Propagation speed depends on variable density, the acoustic pressure, and the media density. The Numerical method solves Equation 1 and is detailed in [Vilela 2017]. Figure 2 illustrates the acoustic wave propagation stencil.

$$\begin{aligned}
 C_{i,j,k} = & a_0 C_{i,j,k} \\
 & + a_1 (C_{i-1,j,k} + C_{i+1,j,k} + C_{i,j-1,k} + C_{i,j+1,k} + C_{i,j,k-1} + C_{i,j,k+1}) \\
 & + a_2 (C_{i-2,j,k} + C_{i+2,j,k} + C_{i,j-2,k} + C_{i,j+2,k} + C_{i,j,k-2} + C_{i,j,k+2}) \\
 & + a_3 (C_{i-3,j,k} + C_{i+3,j,k} + C_{i,j-3,k} + C_{i,j+3,k} + C_{i,j,k-3} + C_{i,j,k+3})
 \end{aligned} \tag{1}$$

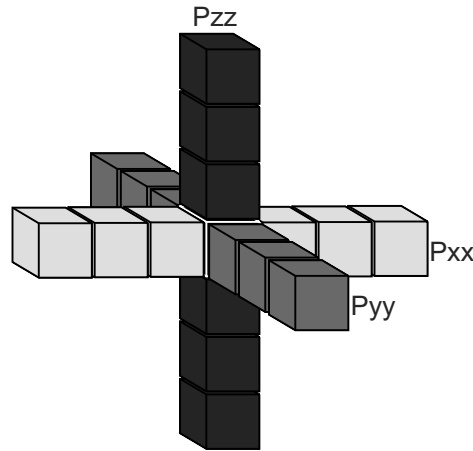


Figure 2: Acoustic Wave Propagation 7 point Stencil.

3.2. Optimization Strategies

Current GPU architectures provide memories with different characteristics compared with CPUs. One of the main differences between GPUs and CPUs is the way their memory

subsystem work. In a CPU, access to memory is done by obtaining their data from caches. Usually looking on L1, L2, L3, and DRAM in that order. On the other hand, in a GPU the L1 memory cache, is used specifically for accesses to the stack and register spill, i.e., when too many local variables do not fit in the register file, and thus some of it has to be cached. L2 memory is used for global accesses requested by stream processors. There are also registers files, a shared memory, a texture memory and a global memory with different characteristics such as size, speed, read-only memory and in the way that is possible to use them. These registers were not available in NVIDIA GPUs before Kepler architecture. In Figure 3, we show an overview of the Kepler GPU architecture.

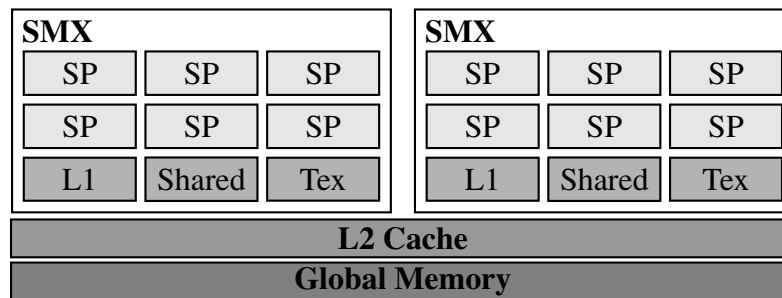


Figure 3: Sample of the memory subsystem on a NVIDIA Kepler architecture

To exploit the use of different memory levels available on current GPU, we develop three versions of a stencil kernel using each one of the GPU memories. Each stencil version, give us a different insight of the performance and capabilities of the GPU memory subsystem.

- The first version called *naive* take no advantage of any of the GPU high-speed memories and access data only from global memory.
- The second version called *shared* stores one part of the stencil data in the shared memory scratchpad. The shared-memory version also uses the GPU resources that the naive version uses, the main difference is that this version also uses the shared-memory available on each SMX (Streaming Multiprocessors). Each one of the SMX has one internal shared-memory to store data as shown in Figure 3. In this version, data is manually allocated by the programmer through the use of the *shared* directive, indicating such data will be shared among all the GPU threads. The compiler automatically configures the space division between the L1 cache memory and the shared cache memory, choosing one of three options: 16 KB for the L1 cache and 48 KB for the shared cache, 32 KB for each, or 48 KB for the L1 cache and 16 KB for the shared cache.
- The third version called *read-only* stores most read data in a read-only texture memory which is faster than shared memory but works with read-only data. This version takes advantage of the read-only cache which is the SMX memory bank that stores only read data, it is also called texture memory. Originally it was used only for textures, but starting with the Kepler architecture any data can be stored in this cache by using the C-99 directive `const restrict`. The programmer may also explicitly use this cache through the intrinsic `lgd()`.

We developed two optimizations for each of the versions to evaluate improvements in performance and energy efficiency by reusing the Z direction data. Reusing Z direction data is named *internalization*.

- The *int.z* version takes advantage of data locality by storing stencil data for direction Z. This optimization consists of the internalization of the Z-axis into the threads. Doing the internalization ensures that neighboring Z-blocks execute sequentially, increasing the reuse of L2 cache data. Direction Z data is used to calculate subsequent points in the X-Y direction.
- The *int.z.reg* version consists of combining the *int.z* with the usage of registers to store the Z direction points. For example, to calculate the point Z3 in a 13 points stencil, the neighboring points in X and Y, as well as points Z1, Z2, Z3, Z4 and Z5 are required. In order to calculate the points in Z4, points Z2, Z3, Z4 and Z5 would be available, and it is necessary to request the global memory only points Z6, as well as the neighbors in X and Y.

3.3. Experimental Methodology

Our experiments were developed in two NVIDIA GPU card. The first is an NVIDIA K80 GPU. This card is a Kepler architecture GPU with 2496 CUDA cores. The second card is an NVIDIA P100 GPU. This card is a Pascal architecture GPU with 3584 CUDA cores. Table 1 describes in detail the environments we used.

Device	Tesla K80	Tesla P100
CUDA Cores	2496	3584
Registers	13 x 512kB	56 x 256KB
Memory	13 x 128KB L1 / <i>shared</i> 1536KB L2 13 x 48KB <i>texture (read-only)</i> 12GB GDDR5 <i>global memory</i>	56 x 64KB <i>shared</i> 4096KB L2 56 x 24KB L1 / <i>texture (read-only)</i> 16GB GDDR5 <i>global memory</i>

Table 1: Configuration of GPU system.

We used NVIDIA Management Library (NVML) to measure the power usage. Regarding the energy efficiency, we used performance divided by average power. Each experiment was executed 20 times, we show average values as well a 95% confidence interval calculated with Student's t-distribution.

4. Results

This section shows the optimizations techniques we used to improve the performance and energy efficiency of a stencil application. The stencil we used simulates the propagation of a single wavelet over time. To create the simulation, it solves the isotropic acoustic wave propagation with constant density under Dirichlet boundary conditions over a 3D domain. The stencil is a 31-arm with a $(1024 \times 256 \times 256)$ input size.

In the following subsections, we describe each optimization and analyze how they address the performance and energy efficiency improvements. We also show the results obtained by using the three different memories and the results of the optimizations applied in each of them, on an NVIDIA Kepler architecture and NVIDIA Pascal architecture.

4.1. Performance Improvements

In this subsection, we show the Performance improvements obtained by using different GPU memory subsystem (*global-*, *shared-*, and *read-only-memory*) and two optimization techniques over a stencil kernel. In Figure 4 is showed the performance achieved for

each kernel version and optimizations performance. *P100* features 5x more performance than *K80* GPU for this stencil kernel using Naive version. The performance achieved in *K80* was 98.1 GFLOPS while in *P100* was 489.3 GFLOPS.

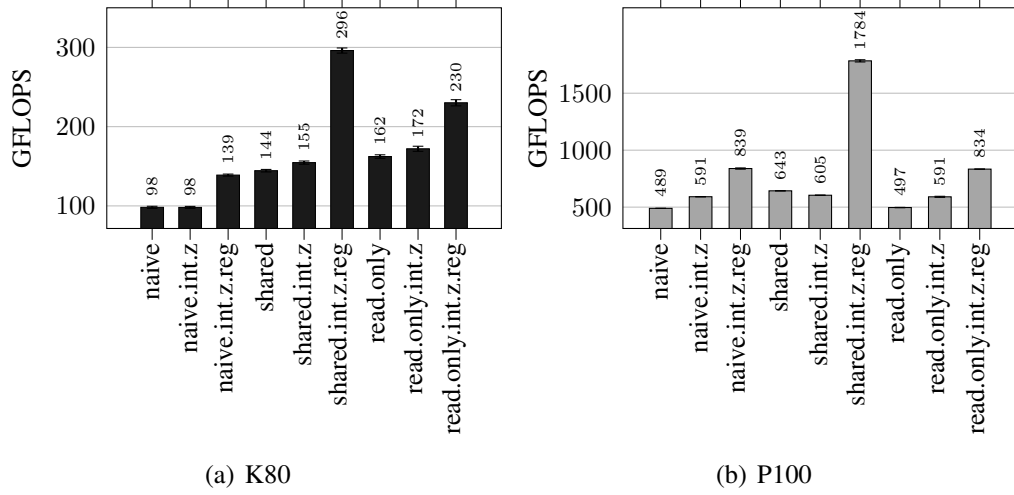


Figure 4: Performance gains over *K80* and *P100*

Applying the first optimization technique *int.z* which stores data from direction *Z* in local variables to take advantage of the data locality by reusing these data in the subsequent iterations, performance gains were not realized in *K80* (Figure 4(a)). Otherwise, in *P100* (Figure 4(b)), the performance were improved by up to 20,8% using the this *int.z* technique over a *naive* version. When the second optimization *int.z.reg* was employed, which consists of the *int.z* optimization along with the use of the register file to store the *Z* points, the performance was increased from 98.1 GFLOPS to 138.8 GFLOPS in *K80*, from 489.3 GFLOPS to 838.9 GFLOPS in *P100*, which represent an increase of 41.5% and 71.4% respectively.

Although the performance was improved using optimizations techniques, the *naive* version does not take advantage of fast GPU memories as *shared* memory. Thus, we improved the *naive* version by used the *shared* memory scratch pad to store a slice of data that is reused by the threads of the same block.

To the second version, the performance was increased from 98.1 GFLOPS to 144.4 GFLOPS in *K80*, from 489.3 GFLOPS to 643.2 GFLOPS in *P100*. In this version the data were manually allocated using the *shared* directive, indicating a piece of data shared among all threads. The performance improvement using the shared memory compared to the naive was 47.10% on the *K80* and 31.45% on the *P100*. In the *P100*, the performance is lower because in Pascal architecture the number of SMs were increased and the number of SPs in each SM were decreased. In this case, each *P100* SM has less shared memory than a *K80* SMX.

Similarly to the first version, we also applied the *int.z* and *int.z.reg* optimizations in this version aiming to improve the performance of the memory operations. The *int.z* optimization over stencil represented an improvement of 7.07% in *K80* performance, which achieve 154.6 GFLOPS. Differently, in *P100* the performance was reduced when used this optimization.

The second optimization (*int.z.reg*) over *shared* memory performed in *K80* increased the performance in 104.9% when compared to *shared* memory version without optimization and 201.5% when compared to *naive* without optimizations. Using this optimization

and *shared* Memory version in a stencil application was achieved the best performance, 296.04 GFLOPS. The gains were greater when the version with the second optimization was applied in *P100*. The performance increased in 177.3% when compared to *shared* memory version without optimization. Since the data are stored in the *shared* memory they are not updated. So, we may take more advantage if the *read-only* memory is used. The *read-only* memory is faster than *shared* memory but exclusively used for read-only operations.

Running the *read.only* version, which define that *global* memory reads are stored in the *read-only* memory using the *lgd()* intrinsic, the performance achieved 162.2 GFLOP in the *K80* 496.5 GFLOPS in the *P100*. The performance improvement of this strategy compared to the naive on *K80* was 65.26% and 1.48% on *P100*. The low improvement occurs on *P100* architecture because on Pascal, the read-only memory is shared with the L1 and the compiler defines the distribution for each memory. In Kepler, the read-only memory is not shared with any other memory.

Adding the *int.z* optimizations over this third version of the kernel the performance increase 6.03% in the *K80* and 18.9% in the *P100*. The performance is also increased when used the *int.z.reg* optimization over stencil performing in the *read-only* memory. Implementing the *int.z.reg* that also uses the register file, the performance was improved by up to 41.8% in the *K80* and 67.9% in the *P100*.

4.2. Energy Efficiency Improvements

In this subsection, we show the energy efficiency improvements obtained by using different GPU memory subsystem (*global*-, *shared*-, and *read-only*-memory) and two optimization techniques over a stencil kernel. Figure 5 showed the energy efficiency achieved for each kernel version and optimizations performance.

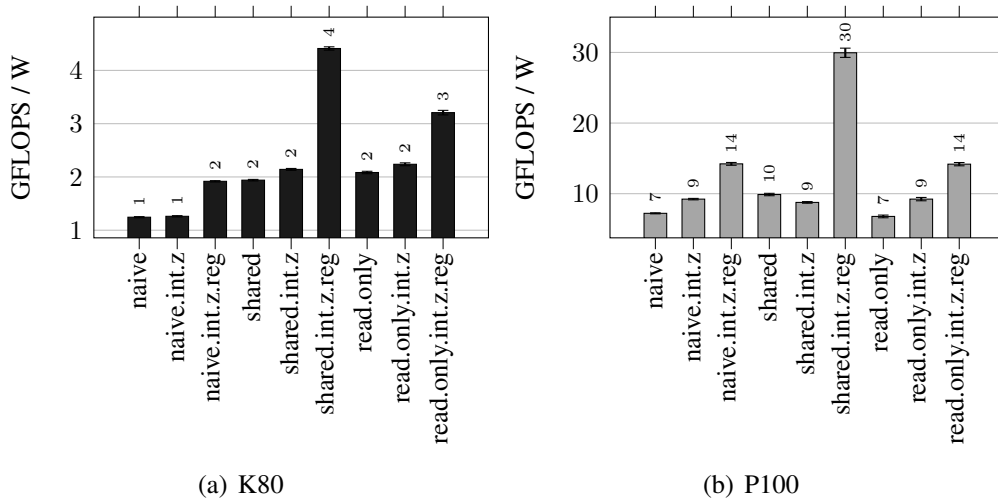


Figure 5: Energy Efficiency gains over *K80* and *P100*

The gains in energy efficiency are similar to those achieved with the increase in performance. The energy efficiency of the *P100* (Figure 5(a)) was 5.8 times greater than *K80* (Figure 5(b)) to *naive* version. The first optimization technique *int.z* overtakes the normal version and also improvement the energy efficiency of up to 28.8% compared with the *naive* version in *P100*. When the second optimization *int.z.reg* was employed, the energy efficiency increase 54.8% in the *K80* and 96.9% in the *P100*.

Analysing the energy efficiency gains to the second version of the kernel, it was 56.4% in the *K80* and 36.8% in the *P100* over Naive version without optimization. Applying the *int.z* and *int.z.reg* optimizations in this version the energy efficiency improvement in *K80* and reduces in *P100*. However, the main gains were achieved with the second optimization (*int.z.reg*) over *shared* memory. It increases the energy efficiency in 255.6% in the *K80* and 314.8% in the *P100* when compared to the naive version without optimizations. The best efficiency was achieved when the *int.z.reg* optimization was used over *read-only* version. In *K80* it achieved 3.21 GFLOP/W.

5. Conclusion

Nowadays, the use of GPUs in HPC systems has become a popular choice among the top-ranked platforms. However, to achieve even greater performance and energy efficiency is necessary to exploit the different memory levels available. Several scientific applications make use of stencil computations to their model simulations and are performed in GPU platforms. Stencils have both implicit and explicit partial differential equations (PDE) being interesting as an architectural evaluation benchmark. Their computing present low computational intensity, once that these applications are typically memory-bound. In this form, memory optimizations are important for to use the fastest memories available in GPUs and increase their energy efficiency.

In this paper, we improved the performance and achieved energy efficiency by the development of three different strategies which use the GPU memory subsystem (*global*, *shared*, and *read-only* memory). We also develop two optimizations to use data locality and use of registers of GPU architecture. To analyze the impact of our proposed strategies and optimizations, we applied them to a stencil application and ran on Tesla K80 and Tesla P100 GPUs. Our developed strategies to GPU algorithms achieved the performance improvement of up to 201.5% in *K80* and 264.6% in *P100* when were used *shared* memory and *read-only* cache respectively over the *naive* version. These increases in computational performance also improve the energy efficiency. The main gains were achieved with (*int.z.reg*) optimization over *shared* memory which increases the energy efficiency in 255.6% in *K80* and 314.8% in *P100* when compared to the naive version without optimizations.

Changes in the GPU architecture, as in the case of the introduction of the *read-only* cache in the Kepler architecture, can generate changes in the results presented in this work. In the future, we plan to investigate methods and optimizations aiming to achieve gains in stencil applications over new NVIDIA architecture and Intel Xeon Phi.

Acknowledgment

This research received funding from the EU H2020 Programme and from MCTI/RNP-Brazil under the HPC4E project, grant n.º 689772. It was also supported by Intel under the Modern Code project, and Petrobras 2016/00133-9.

References

- Bauer, M., Cook, H., and Khailany, B. (2011). Cudadma: Optimizing gpu memory bandwidth via warp specialization. In *Proceedings of 2011 International Conference for High Performance Computing, Networking, Storage and Analysis, SC '11*, pages 12:1–12:11, New York, NY, USA. ACM.
- Datta, K., Murphy, M., Volkov, V., Williams, S., Carter, J., Oliker, L., Patterson, D., Shalf, J., and Yelick, K. (2008). Stencil computation optimization and auto-tuning on state-of-the-art multicore architectures. In *Proceedings of the 2008 ACM/IEEE conference on Supercomputing*, page 4. IEEE Press.

- de la Cruz, R. and Araya-Polo, M. (2011). Towards a multi-level cache performance model for 3d stencil computation. *Procedia Computer Science*, 4:2146–2155.
- Dong, Y., Chen, J., and Tang, T. (2010). Power measurements and analyses of massive object storage system. In *Proceedings of CIT*, pages 1317–1322. International Conference on Computer and Information Technology (CIT), IEEE Computer Society.
- Dubey, A. (2014). Stencils in scientific computations. In *Proceedings of the Second Workshop on Optimizing Stencil Computations*, pages 57–57. ACM.
- Falch, T. L. and Elster, A. C. (2014). Register caching for stencil computations on gpus. In *2014 16th International Symposium on Symbolic and Numeric Algorithms for Scientific Computing*, pages 479–486. IEEE.
- Feng, X., Ge, R., and Cameron, K. W. (2005). Power and energy profiling of scientific applications on distributed systems. In *International Parallel and Distributed Processing Symposium (IPDPS)*, pages 34–34. International Conference on Performance Engineering, IEEE.
- Hamilton, B., Webb, C. J., Gray, A., and Bilbao, S. (2015). Large stencil operations for gpu-based 3-d acoustics simulations. *Proc. Digital Audio Effects (DAFx)*, (Trondheim, Norway).
- Laros, J., Pedretti, K., Kelly, S., VanDyke, J., Ferreira, K., Vaughan, C., and Swan, M. (2009). Topics on measuring real power usage on high performance computing platforms. In *Proceedings...*, pages 1–8. International Conference on Cluster Computing and Workshops (ICCC).
- Maruyama, N. and Aoki, T. (2014). Optimizing stencil computations for nvidia kepler gpus. In *Proceedings of the 1st International Workshop on High-Performance Stencil Computations*, Vienna, pages 89–95.
- Micikevicius, P. (2009). 3d finite difference computation on gpus using cuda. In *Proceedings of 2Nd Workshop on General Purpose Processing on Graphics Processing Units, GPGPU-2*, pages 79–84, New York, NY, USA. ACM.
- Nasciutti, T. C. and Panetta, J. (2016). Impacto da arquitetura de memória de gpgpus na velocidade de computação de estênceis. In *XVII Simpósio de Sistemas Computacionais (WSCAD-SSC)*, pages 1–8, Aracaju, SE.
- Nikitin, V. V., Duchkov, A. A., and Andersson, F. (2012). Parallel algorithm of 3d wave-packet decomposition of seismic data: Implementation and optimization for gpu. *Journal of Computational Science*, 3(6):469–473.
- Padoin, E. L., de Oliveira, D. A. G., Velho, P., Navaux, P. O. A., and Mehaut, J.-F. (2013a). ARM-based cluster: Performance, Scalability and Energy Efficiency. In *4th Workshop on Applications for Multi-Core Architectures (WAMCA SBAC-PAD)*, pages 1–6, Porto de Galinhas, PB, Brasil.
- Padoin, E. L., Pilla, L. L., Boito, F. Z., Kassick, R. V., Velho, P., and Navaux, P. O. (2013b). Evaluating application performance and energy consumption on hybrid cpu+ gpu architecture. *Cluster Computing*, 16(3):511–525.
- Schäfer, A. and Fey, D. (2011). High performance stencil code algorithms for gpgpus. *Procedia Computer Science*, 4:2027 – 2036. Proceedings of the International Conference on Computational Science, ICCS 2011.

- Vilela, R. F. (2017). Perfilagem do problema de resolução da equação da onda por diferenças finitas em coprocessador xeon phi.
- Xue, Q., Wang, Y., Zhan, Y., and Chang, X. (2015). An efficient gpu implementation for locating micro-seismic sources using 3d elastic wave time-reversal imaging. *Computers & Geosciences*, 82:89–97.
- Zhou, G., Zhang, X., Lang, Y., Bo, R., Jia, Y., Lin, J., and Feng, Y. (2016). A novel gpu-accelerated strategy for contingency screening of static security analysis. *International Journal of Electrical Power & Energy Systems*, 83:33–39.
- Zhou, J., Unat, D., Choi, D. J., Guest, C. C., and Cui, Y. (2012). Hands-on performance tuning of 3d finite difference earthquake simulation on gpu fermi chipset. *Procedia Computer Science*, 9:976–985.

Consumo de Energia em Função da Taxa de Transmissão e do Tamanho do Pacote em Redes *Ad Hoc* Aloha de um Salto

Bruna A. da Silva¹, Renato M. de Moraes¹

¹Centro de Informática – Universidade Federal de Pernambuco (UFPE)
CEP 50740-560 – Recife – PE – Brasil

bas4@cin.ufpe.br, renatomdm@cin.ufpe.br

Abstract. *The autonomy of a communication device, together with its form of operation, is related to its energy consumption. In ad hoc and sensor wireless networks we seek that the device can operate with high efficiency and with minimum human interference as possible during its operation. In this paper, a study was carried out on the energy expended to perform the communication at one hop E_{1hop} (energy consumption per bit for one-hop successful transmission) employing slotted Aloha medium access control protocol in order to provide, at the physical and link layer level, the optimization of the transmission power (P_t) as a function of packet length (N_b) and data transmission rate (R). Our results indicate that there are possible optimum values of operations for P_t as a function of N_b and R which result in reduced energy consumption for such networks.*

Resumo. *O aumento da autonomia de um dispositivo de comunicação, aliado à sua forma de operação, está relacionado com seu consumo de energia. Em redes ad hoc e de sensores sem fio deseja-se que o dispositivo possa operar com a melhor eficiência possível e com o mínimo de interferência humana durante seu funcionamento. Neste artigo, um estudo foi realizado sobre a energia gasta para realização da comunicação em um salto E_{1hop} (energia consumida por bit para transmissão com sucesso em um salto) empregando slotted Aloha como protocolo de acesso ao meio a fim de proporcionar, na camada física e de enlace, a otimização da potência de transmissão (P_t) em função do número de bits do pacote (N_b) e da taxa de transmissão de dados (R). Nossos resultados indicam que é possível se obter valores ótimos de P_t em função de N_b e R que resultam em reduzido consumo de energia nestas redes.*

1. Introdução

A causa predominante do consumo de energia em redes sem fio *ad hoc* e também de sensores está relacionada à comunicação. Esse consumo de energia, contudo, é em geral diretamente relacionado a diversos parâmetros das camadas física e de enlace. Por outro lado, deseja-se que os dispositivos destas redes consigam se autoajustar às diversas condições de operações de forma a resultar no melhor desempenho dentre as diversas métricas de funcionamento. Em redes *ad hoc* e de sensores que operam com bateria, o consumo de energia é uma das principais medidas de desempenho, pois está diretamente ligada à duração (ou longevidade) de operação dos dispositivos que a compõem [Gupta e Shekhar 2016], [Mobin et al. 2017].

Em [Gupta e Shekhar 2016] um novo modelo é proposto a fim de promover o tempo de vida de um nó sensor por otimização do tamanho do pacote, de modo que

não haja perdas. Considerando que o consumo de energia é diretamente proporcional ao tamanho do pacote e a distância entre os vários nós participantes na comunicação, os autores propuseram um algoritmo que consiste de quatro fases. Nesse contexto, o tempo de vida da rede é usado como um parâmetro para analisar e mostrar que a rede pode ser energeticamente eficiente. Porém, esse trabalho focou apenas no tamanho do pacote.

Para [Abdulhadi et al. 2013] a eficiência energética é usada como uma métrica de otimização cujo problema é formulado como não linear, ou seja, cuja definição consiste em um sistema de equações e desigualdades, coletivamente denominadas restrições, por meio de um conjunto desconhecido de variáveis reais que, juntamente a uma função objetivo, podem ser maximizadas ou minimizadas. Em seu trabalho o autor formula a eficiência energética e o problema da otimização do pacote, buscando um tamanho de pacote ótimo que maximiza a eficiência energética em redes *ad hoc* sem fio de comunicação cooperativa. Novamente o foco do estudo foi o tamanho do pacote.

Diante do estudo desenvolvido por [Akbas et al. 2016], mostrou-se que há um tamanho de pacote ótimo para cada cenário especificado onde o tempo de vida da rede é maior. No entanto, este resultado é observado apenas se o nível de sensibilidade do receptor é baixo a ponto de permitir que transceptores sejam capazes de operar em uma parcela significativa da região de transição tal como especificado em [Zuniga e Krishnamachari 2004]. O principal objetivo do trabalho foi caracterizar o impacto do tamanho do pacote e dos níveis da potência de transmissão no que diz respeito ao tempo de vida da rede.

Assim, motivados pela necessidade de explorar novas abordagens que considerem não apenas o tamanho do pacote, mas também outros importantes parâmetros como a taxa de transmissão e a potência de transmissão, que podem ser automaticamente ajustáveis pelos próprios dispositivos, o presente trabalho busca analisar o consumo de energia para a realização da comunicação a um salto em função desses parâmetros, considerando também o protocolo de acesso ao meio. Aqui, para simplificar a análise, todos os nós estão ao alcance dos demais, não sendo portanto considerado o efeito do terminal oculto, caracterizando assim uma rede de um único salto.

Este trabalho está dividido conforme descrito a seguir. A Seção 2 aborda os trabalhos relacionados. A Seção 3 revisita a formulação de [Zhang e Gorce 2008] e em seguida a ampliamos para incluir o efeito do protocolo de comunicação Aloha e agora com foco no consumo de energia em função da taxa de transmissão de dados (R), do tamanho do pacote (N_b) para uma potência ótima de transmissão. Os resultados e discussão são apresentados na Seção 4 e a Seção 5 conclui o artigo.

2. Trabalhos Relacionados

Tendo em vista que geralmente a distribuição dos nós em redes *ad hoc* e de sensores são nos mais variados ambientes e que lá permanecerão por tempo indeterminado, sem que ocorra recarga de suas baterias, é de fundamental importância que possuam um consumo de energia mínimo a fim de que se conservem ativos pelo máximo de tempo possível. Uma vez que esses nós sejam utilizados, por exemplo, para aplicações de monitoramento e sensoriamento, é primordial a realização de estudos do consumo de energia de cada um dos componentes dessa rede [Rana e Zaveri 2011], [Stankovic e He 2012], [Ye et al. 2002], [Feeney e Nilsson 2001].

Por se tratar de redes que apresentam seu desempenho relacionado com o consumo de energia, contenção e colisão de pacotes, o tema abordado neste trabalho vem sendo amplamente discutido. Em [Tang et al. 2017] um estudo é desenvolvido com base em métodos de redução do consumo de energia de protocolos de roteamento geográficos. Esses tipos de protocolo roteiam um pacote salto-a-salto até que se chegue ao destino. Neste contexto, o nó transmissor precisa obter a informação de sua posição geográfica, da posição de seus vizinhos de um salto e do nó receptor. Destaca, ainda, o consumo de energia na transmissão de um pacote de dados, a um salto, do nó transmissor para o nó receptor, incluindo as retransmissões. Contudo, o trabalho não considera a taxa de erro de bit (BER, do inglês *Bit Error Rate*) e nem a energia gasta para enviar um bit com sucesso. Também não faz análises referentes ao tamanho do pacote de dados e nem à taxa de transmissão.

[Ito et al. 2016], por sua vez, desenvolve um protocolo para redes *ad hoc* utilizando BLE (do inglês *Bluetooth Low Energy*) e realiza seu experimento considerando a latência medida em um salto. Porém, também não é verificado em suas análises um estudo sobre a taxa de erro de bit, sobre a energia gasta para a realização da transmissão de um pacote de bits a um salto e nem sobre o efeito do tamanho do pacote nessa transmissão.

Um algoritmo genético é utilizado em [Shehab et al. 2017] para maximizar a vazão e minimizar o consumo de energia em redes, cujo os nós de um mesmo *cluster* estão a um único salto um do outro. Considera como parâmetros a distância média entre os nós e o consumo esperado de energia. No entanto, não é feito um estudo sobre a influência de protocolos, a nível de camada de enlace, que certifiquem a probabilidade de um dado nó transmitir com sucesso considerando o número de retransmissões necessárias em decorrência das eventuais colisões.

Em [Sadiq et al. 2017] é feito um estudo na qual a camada de rede utiliza as informações da camada física e da camada de enlace a fim de encontrar um caminho para o envio de dados que promova uma comunicação energeticamente eficiente. Foram considerados parâmetros como o número de nós, a distância entre os nós, a taxa de erro de bit, o consumo de energia de cada nó, a potência de transmissão, dentre outros. É empregada uma análise em termos da taxa de transmissão de dados e do consumo de energia por bit. No entanto, não se faz um estudo detalhado em termos do tamanho do pacote, bem como da energia gasta para a realização em cada salto.

Um esquema de otimização adaptativa do consumo dinâmico de energia foi proposto por [Liu et al. 2015]. Nesse contexto, a otimização do consumo de energia é feito em função do BER, da distância de referência em transmissão de sinais sem fio, do número de nós em um *cluster*, dos pacotes codificados na rede e do consumo de energia para enviar ou receber um bit. Por outro lado, não é explorada a probabilidade de sucesso na entrega do dado, não faz uso de nenhum protocolo de camada de enlace e nem verifica a influência da taxa de transmissão no consumo de energia da rede.

[Goratti et al. 2014], em seu artigo, faz uso do protocolo *slotted* Aloha a fim de obter um sistema livre de colisão, uma vez que nesse tipo de protocolo, um nó transmite por vez. Em sua abordagem, considera o consumo de energia e toma como parâmetros a potência e a taxa de transmissão. No entanto, não considera parâmetros como o tamanho do pacote e a taxa de erro de bit. [Li et al. 2015], por sua vez, faz uso do protocolo *slotted*

Aloha como objeto de comparação em redes de sensores. Faz análises do consumo de energia apenas baseadas na densidade da rede e não considera a influência de parâmetros como a taxa de transmissão e o tamanho do pacote.

Em [Gao 2002], [Gorce et al. 2007] e [Zhang e Gorce 2008] foi desenvolvido um robusto modelo de consumo de energia para redes *ad hoc* e de sensores onde foi levado em consideração diversos parâmetros reais da camada física. Denominada EDRb (do inglês *Energy Distance Ratio per bit*), cuja unidade é dada em Joule por bit por metro (J/bit/m), essa métrica indica o consumo energético em Joules para se enviar um bit de dado sobre uma distância padrão de um metro. Deste modo, quanto menor for o valor de EDRb, mais eficiente energeticamente a rede se comporta e, portanto, menor o consumo de energia resulta.

A eficiência energética expressa nessa métrica traz consigo o embasamento necessário para estudo e análise de parâmetros como o raio de transmissão ótimo, cuja definição representa o alcance ideal no qual um nó possa enviar informação e esta ser recebida com êxito. Caso o comprimento do salto de comunicação entre os nós seja muito grande, será necessário gastar uma maior quantidade de energia para viabilizar a comunicação de forma a não ocorrer falha na comunicação e consequente redução da conectividade na rede. Por outro lado, se o comprimento do salto for muito pequeno, a energia gasta em cada salto será bem menor, mas se fará necessário mais saltos para se alcançar o destino resultando em um consumo acumulado demasiado.

O estudo detalhado do consumo de energia apresentado por [Zhang e Gorce 2008] procurou determinar a distância ótima de comunicação entre os nós em função da potência de transmissão. Entretanto, em casos práticos, o posicionamento dos nós pode não ser um parâmetro controlado com muita precisão, seja porque os nós são dispostos na rede de forma aleatória, ou porque a possibilidade de mobilidade aleatória deva ser levada em consideração. Também não foi levado em consideração o efeito do protocolo de acesso ao meio.

No presente artigo, portanto, é realizado um estudo do consumo de energia da comunicação em um salto. Diferentemente das análises citadas anteriormente, trazemos agora uma investigação do efeito conjunto da comunicação a um salto em função da taxa de transmissão de dados (R) e do tamanho do pacote (N_b) empregando a potência de transmissão (P_t) ótima levando também em consideração o protocolo de acesso ao meio (MAC, do inglês *medium access control*). Consideramos a possibilidade da variação da distância entre os nós comunicantes e a probabilidade de sucesso do nó transmitir através do uso do protocolo MAC *slotted* Aloha [Roberts 1975] que foi escolhido para análise devido a sua simplicidade de modelagem. Entretanto, a metodologia aqui empregada pode ser estendida para outros protocolos de acesso ao meio.

3. Consumo de Energia

Utilizamos como ponto de partida a formulação da energia consumida para a realização de um salto com retransmissão desenvolvida em [Zhang e Gorce 2008] que atualizou a descrição de [Karl e Willig 2007] e empregou as propostas de [Gao 2002] e [Gorce et al. 2007]. Trata-se, portanto, de um modelo robusto e optamos partir deste por verificarmos que o mesmo está bem definido, incorpora vários outros parâmetros de camada física e permitiu sua alteração para considerar o protocolo MAC.

Nossa análise considera uma rede sem fio contendo n nós e parte do pressuposto que todos os nós estão ao alcance uns dos outros, caracterizando uma rede *ad hoc* sem fio de um único salto. Isso permite simplificar a análise, pois não será considerado o efeito do terminal oculto. Entretanto, um dos nossos trabalhos futuros será estender a análise para uma rede com múltiplos saltos.

O consumo de energia (E_p) para o envio de um pacote é a soma entre a energia empregada na transmissão E_{Tx} e a energia gasta para recepção E_{Rx} . Assim, resulta que [Zhang e Gorce 2008]

$$E_p = E_{Tx} + E_{Rx}. \quad (1)$$

E_{Tx} e E_{Rx} são descritas, respectivamente, de acordo com [Karl e Willig 2007] por

$$E_{Tx} = T_{start}P_{start} + \frac{N_b}{R}(P_{txElec} + P_{amp}) \quad (2)$$

e

$$E_{Rx} = T_{start}P_{start} + \frac{N_b}{R}P_{rxElec}, \quad (3)$$

onde N_b é o número de bits por pacote, R é a taxa de transmissão de dados, P_{txElec} é a potência para alimentação do circuito de transmissão, P_{rxElec} é a potência para alimentação do circuito de recepção, T_{start} é o tempo de inicialização do circuito e P_{start} é a potência requerida para inicialização. P_{amp} é definido como a potência para amplificação do sinal na transmissão, e é dada por

$$P_{amp} = \alpha_{amp} + \beta_{amp}P_t, \quad (4)$$

em que α_{amp} é um nível constante de potência e β_{amp} é uma constante de proporcionalidade da potência de transmissão P_t .

A fim de obter o consumo médio de energia por bit para uma transmissão a um salto com sucesso, deve-se dividir o consumo para transmissão de um pacote pelo número de bits N_b deste pacote. Consequentemente, define-se o consumo de energia por bit E_b (expresso em Joules por bit (J/bit)) como sendo

$$E_b = \frac{E_p}{N_b} = E_c + K_1P_t, \quad (5)$$

onde E_c e K_1 são dados, respectivamente, por

$$E_c = \frac{2T_{start}P_{start}}{N_b} + \frac{P_{txElec} + P_{rxElec} + \alpha_{amp}}{R}, \quad (6)$$

$$K_1 = \frac{\beta_{amp}}{R}. \quad (7)$$

Por outro lado, para um canal não-confiável, a probabilidade de sucesso de transmissão de um pacote contendo N_b bits é $p_l(\gamma) = (1 - BER(\gamma))^{N_b}$ em que $BER(\gamma)$ é a taxa de erro de bits em função da relação sinal-ruído (γ) [Gorce et al. 2007]. Aqui modificamos essa probabilidade para incluir o efeito do protocolo *slotted* Aloha, em que a probabilidade de acesso ao meio com sucesso de um nó é dada por $\delta(n) = \frac{1}{n} \left(1 - \frac{1}{n}\right)^{n-2}$

para n nós na rede. Logo, resulta que a probabilidade de transmissão com sucesso de um pacote é

$$P_s = p_l(\gamma)\delta(n) = \frac{(1 - BER(\gamma))^{N_b} \left(\frac{n-1}{n}\right)^{n-2}}{n}. \quad (8)$$

A relação sinal-ruído é definida como sendo

$$\gamma = C_2 P_t d^{-\alpha} R^{-1} \quad (9)$$

em que d indica a distância entre dois nós comunicantes, sendo α o coeficiente de atenuação do sinal com a distância, e P_t a potência de transmissão. A constante C_2 é expressa por

$$C_2 = \frac{G_T G_R \lambda^2}{(4\pi)^2 N_0} \quad (10)$$

onde G_T e G_R representam, respectivamente, os ganhos das antenas no transmissor e no receptor, N_0 é a densidade espectral de potência de ruído no canal de comunicação, λ é o comprimento de onda do sinal transmitido e igual a $\frac{c}{f_c}$ em que c é a velocidade da luz e f_c é a frequência da onda portadora.

É importante notar que a relação sinal-ruído não deve considerar a interferência causada pelos demais nós da rede visto que isto é tratado no protocolo de controle de acesso ao meio. Assim, se houver transmissão simultânea de dois ou mais nós, os pacotes de dados colidem entre si e a informação é perdida ocasionando necessariamente a retransmissão, sendo tal efeito incorporado aqui no protocolo Aloha.

A partir de então, podemos definir o consumo médio de energia por bit para um único salto, em unidades de Joules/bit, como

$$\bar{E}_{1hop} = E_b(P_t) \sum_{k=1}^{\infty} k P_s (1 - P_s)^{k-1} = \frac{E_b(P_t)}{P_s} \quad (11)$$

onde k é o número de retransmissões necessárias para o envio do pacote de dados com sucesso. Sendo assim, o consumo médio de energia por bit para uma transmissão bem sucedida a um salto resulta

$$\bar{E}_{1hop} = \frac{(E_c + K_1 P_t)}{(1 - BER(\gamma))^{N_b}} \cdot \mu, \quad (12)$$

em que

$$\mu = \frac{n}{\left(\frac{n-1}{n}\right)^{n-2}}. \quad (13)$$

O canal de comunicação que este trabalho considera é o que emprega o desvanecimento Rayleigh, que é amplamente utilizado na literatura em casos de redes *ad hoc* e de sensores para modelar canais sem fio e sem linha de visada direta entre as antenas comunicantes. Para uma relação sinal-ruído maior que 5, que é o caso para os valores de parâmetros aqui empregados, pode-se aproximar taxa de erro de bit por [Goldsmith 2005]

$$BER(\gamma) \approx \frac{\alpha_m}{2\beta_m \gamma} \quad (14)$$

onde α_m e β_m são constantes que dependem da modulação empregada. Será considerada a modulação BPSK (do inglês *Binary Phase Shift Keying*), que é comumente encontrada nos dispositivos de comunicação sem fio, e nesse caso $\alpha_m = 1$ e $\beta_m = 2$. Substituindo a Eq. (14) em (12), resulta que

$$\bar{E}_{1hop} = \frac{E_c + K_1 P_t}{\left(1 - \frac{\alpha_m}{2\beta_m C_2 P_t d^{-\alpha} R^{-1}}\right)^{N_b}} \cdot \mu. \quad (15)$$

É possível, ainda, determinar a potência de transmissão ótima (P_0), para posterior ajuste conforme as variações da taxa de transmissão (R) e do tamanho do pacote (N_b), através da resolução da derivada parcial a seguir

$$\left. \frac{\partial \bar{E}_{1hop}}{\partial P_t} \right|_{P_t=P_0} = 0. \quad (16)$$

Desenvolvendo essa equação, obtém-se a potência de transmissão ótima em função da taxa de transmissão (R) e em função do tamanho do pacote (N_b), ou seja,

$$P_0(R, N_b) = \frac{d^\alpha \alpha_m (1+N_b)}{4C_2 \beta_m R^{-1}} + \frac{\sqrt{d^\alpha \beta_{amp} \alpha_m (d^\alpha \beta_{amp} \alpha_m (1+N_b)^2 + 8\beta_m C_2 R^{-1} (C_3 R + C_4 N_b))}}{4C_2 \beta_{amp} \beta_m R^{-1}}. \quad (17)$$

As constantes C_3 , C_4 são oriundas do desmembramento de E_c , para representação em função de R e N_b , e são dadas, respectivamente, por

$$C_3 = 2T_{start} P_{start}, \quad (18)$$

$$C_4 = P_{txelec} + P_{rxelec} + \alpha_{amp}. \quad (19)$$

Diante das equações acima apresentadas se propõe, portanto, avaliar o consumo médio de energia por bit para transmissão bem sucedida em um salto em função da taxa de transmissão de dados (R) e do tamanho do pacote (N_b) empregando-se a potência ótima de transmissão em função destes parâmetros (isto é $P_t = P_0(R, N_b)$). Assim, R e N_b podem ser variados a fim de atingir valores ótimos de desempenho fixando os demais parâmetros de projeto que estão descritos na Tabela 1. Desta forma, das Eqs. (15) e (17), tem-se que $\bar{E}_{1hop}$ pode ser dado em função de R e de N_b para os valores ótimos de $P_0(R, N_b)$, da seguinte forma

$$\bar{E}_{1hop}(P_t = P_0(R, N_b)) = \frac{C_3 R + (C_4 + \beta_{amp} P_0(R, N_b)) N_b}{R N_b \left(1 - \frac{\alpha_m}{2\beta_m C_2 P_0(R, N_b) d^{-\alpha} R^{-1}}\right)^{N_b}} \cdot \mu, \quad (20)$$

onde $P_0(R, N_b)$ é dado pela Eq. (17).

4. Resultados e Discussões

No intuito de explorar o desempenho energético das redes *ad hoc* e de sensores aqui modeladas, realizou-se simulações numéricas para transmissão de um salto e apresentamos os resultados para as Eqs. (17) e (20).

Tabela 1. Parâmetros empregados no modelo de consumo de energia [Karl e Willig 2007], [Zhang e Gorce 2008]

Variável	Descrição	Valor
P_{rxElec}	Potência para recepção	$279mW$
P_{start}	Potência para inicialização	$58,7mW$
T_{start}	Tempo de inicialização	$446\mu s$
P_{txElec}	Potência para transmissão	$151mW$
n	Número de nós	2 ou 10
α_{amp}	Nível de potência Eq. (4)	$174mW$
β_{amp}	Constante de proporcionalidade Eq. (4)	5
N_0	Densidade espectral de ruído	$-154dBm/Hz$
f_c	Frequência da portadora	$2,4GHz$
c	Velocidade da luz	$3 \times 10^8 m/s$
G_T	Ganho da antena na transmissão	1
G_R	Ganho da antena na recepção	1
α	Nível de atenuação de percurso	3
d	Distância entre o par de nós comunicantes	$10m, 30m, 50m$

Buscou-se, portanto, dentre as variações no tamanho do pacote e na taxa de transmissão de dados, os valores ótimos dessas variáveis que retornassem o menor consumo de energia por bit para um salto e, conseqüentemente, maior eficiência energética do sistema. Os valores dos parâmetros empregados estão descritos na Tabela 1, que representam as características físicas do transceptor $\mu AMPS-1$ [Karl e Willig 2007].

As Figuras 1 e 2 mostram o desempenho da potência de transmissão ótima (Eq. (17)), em Watts (W), em função da taxa de transmissão R e do tamanho do pacote N_b , respectivamente. Em ambos os casos observa-se que a potência ótima necessária para envio de um pacote aumenta com a taxa de transmissão, com o tamanho do pacote e também com a distância entre o par de nós comunicantes.

Nas Figuras 3 e 5 é possível verificar o consumo médio de energia por bit (em unidades de dBmJ/bit) para uma transmissão bem sucedida em um salto (Eq. (20)). A análise de ambas figuras difere na quantidade de nós que constitui a rede em questão.

Na Figura 3 observa-se que a energia necessária para transmitir um bit diminui com o aumento da taxa de transmissão e com a redução da distância entre os nós. Porém, transmitir a altas taxas demandará um aumento na potência de transmissão ótima requerida. Deve-se, portanto, respeitar-se o limite da taxa de transmissão em que não resulte uma potência acima do permitido pelo hardware do nó transmissor.

Outro aspecto importante de projeto é se empregar uma taxa de transmissão que resulte uma taxa de erros de bit aceitável, pois a BER é uma função direta de R conforme Eqs. (9) e (14). A Figura 4 ilustra o comportamento da taxa de erros de bit em função da taxa de transmissão de dados e percebe-se que o desempenho do sistema nesse quesito piora com o aumento de R . Portanto, deve haver um compromisso no uso deste parâmetro.

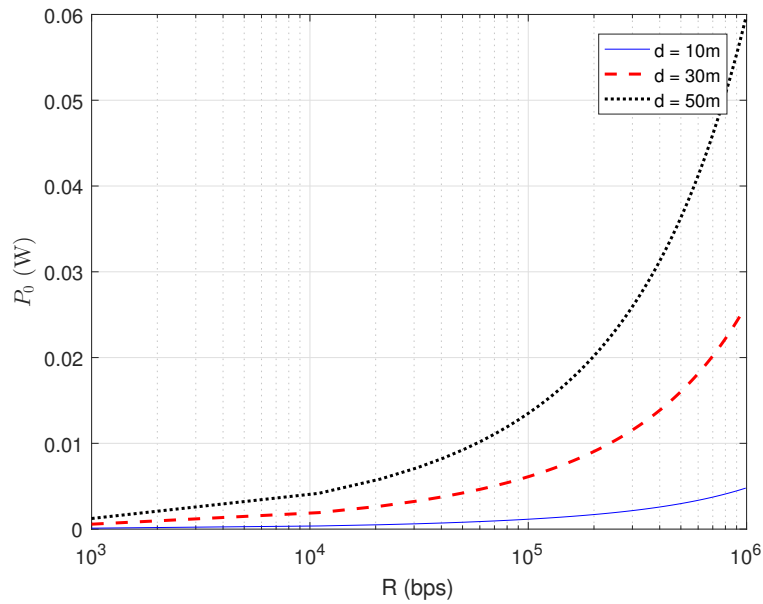


Figura 1. Potência ótima de transmissão em função da taxa de transmissão de dados (R) com o tamanho do pacote $N_b = 100$ bits e $n = 10$ nós.

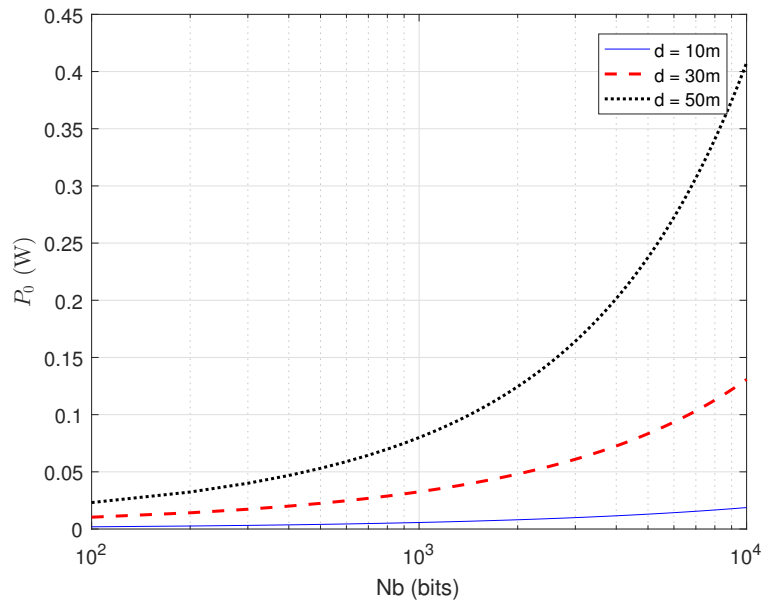


Figura 2. Potência ótima de transmissão em função do tamanho do pacote (N_b) com uma taxa de transmissão $R = 250\text{kbps}$ e $n = 10$ nós.

Para efeito de comparação, a Figura 5 mostra o consumo de energia, quando não existem nós interferentes, ou seja, no cenário em que há apenas dois nós se comunicando ($n = 2$) onde um desses nós transmite e o outro recebe. Como esperado, devido a ausência de nós interferentes na rede, observa-se um menor consumo de energia em função da taxa de transmissão quando se compara com a Figura 3.

As Figuras 6 e 7 retratam o consumo médio de energia por bit para um salto em

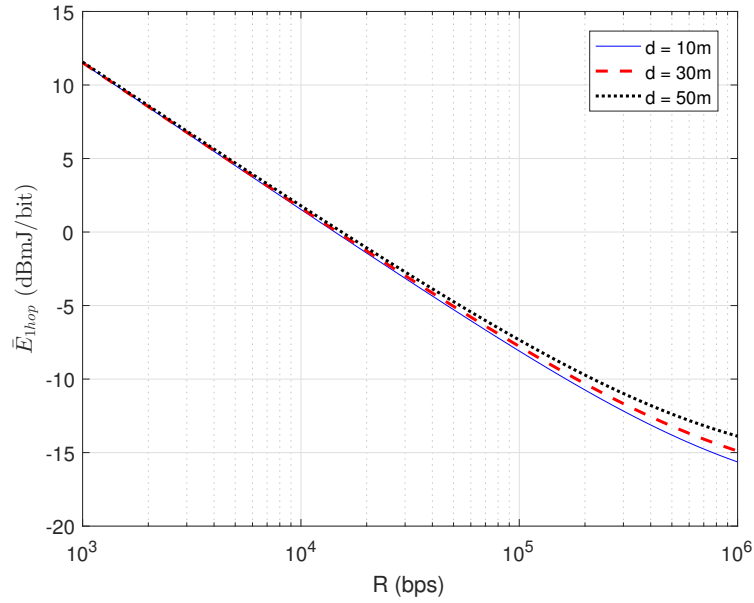


Figura 3. $\bar{E}_{1hop}$ em função da taxa de transmissão de dados (R) com o tamanho do pacote $N_b = 100$ bits e com $n = 10$ nós.

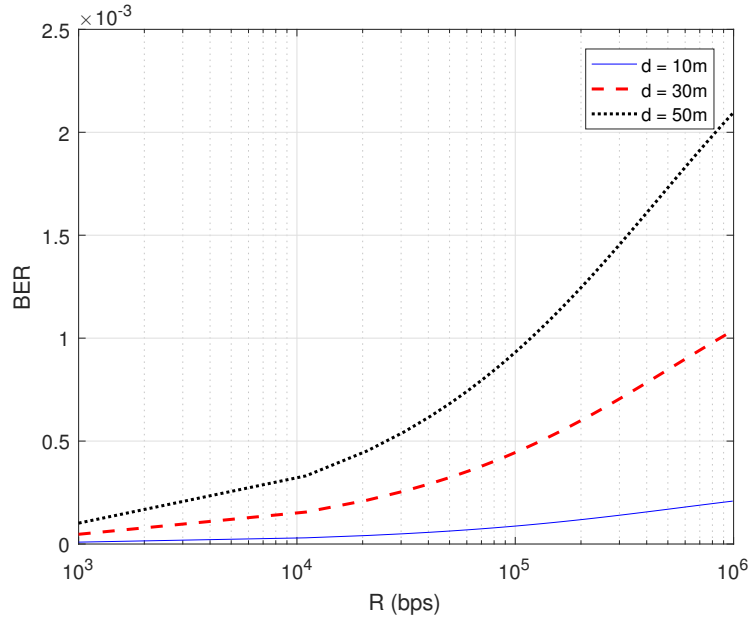


Figura 4. Taxa de erro de bit (Eq. (14)) em função da taxa de transmissão de dados (R).

função do tamanho do pacote de dados (Eq. (20)). Observa-se que o consumo mínimo de energia se dá para N_b entre 100 e 1000 bits, dependendo da distância entre os nós.

Comparando a Figura 6 para uma rede com $n = 10$ nós e a Figura 7 para um cenário com apenas $n = 2$ nós comunicantes, observa-se no cenário de dois nós um melhor desempenho, pois neste caso resulta menor probabilidade de erro na transmissão devido à ausência de interferência dos outros nós da rede e consequente menor consumo

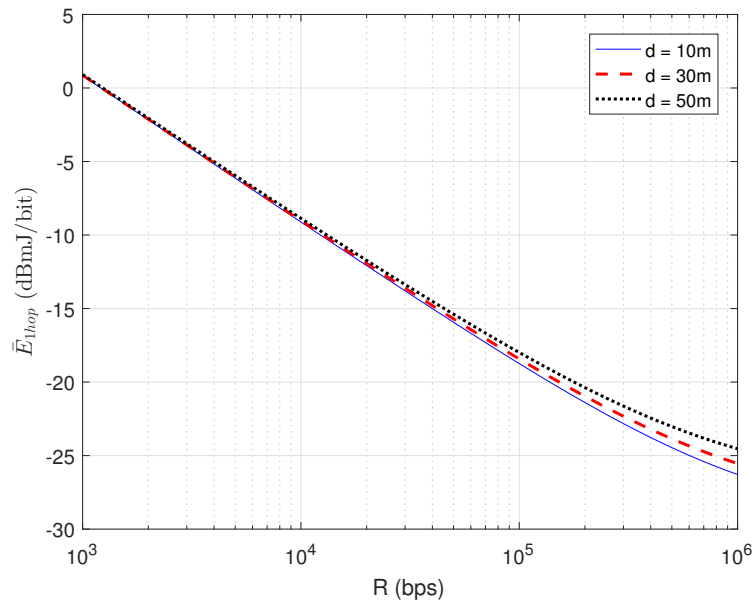


Figura 5. $\bar{E}_{1hop}$ em função da taxa de transmissão de dados (R) com o tamanho do pacote $N_b = 100$ bits e com $n = 2$ nós comunicantes.

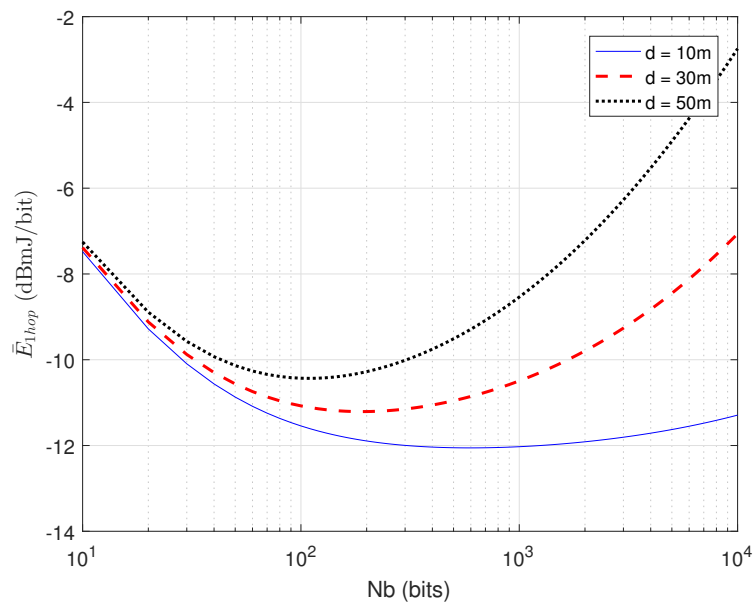


Figura 6. $\bar{E}_{1hop}$ em função do tamanho do pacote (N_b) com uma taxa de transmissão $R = 250$ kbps e $n = 10$ nós.

de energia.

Por fim, para uma completa visualização da variação da energia consumida em um salto, a Figura 8 ilustra o comportamento como uma função simultânea da taxa de transmissão de dados e do tamanho do pacote.

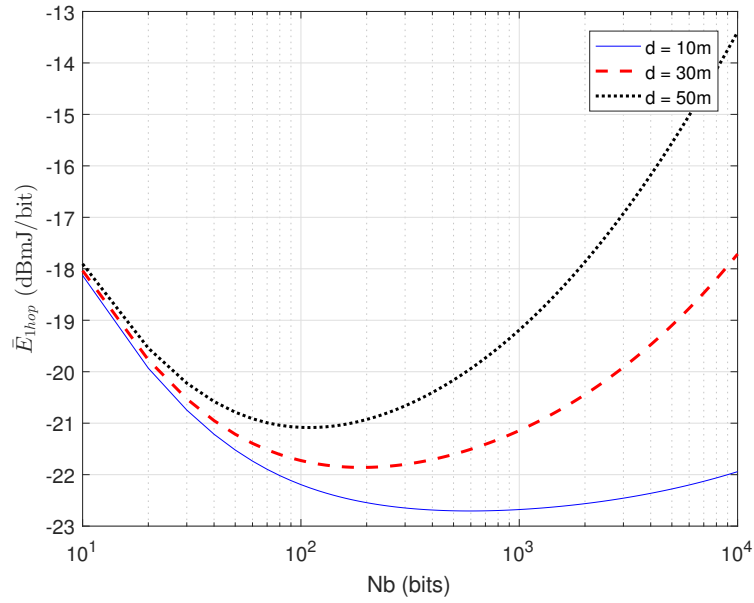


Figura 7. $\bar{E}_{1hop}$ em função do tamanho do pacote (N_b) com uma taxa de transmissão $R = 250\text{kbps}$ e $n = 2$ nós comunicantes.

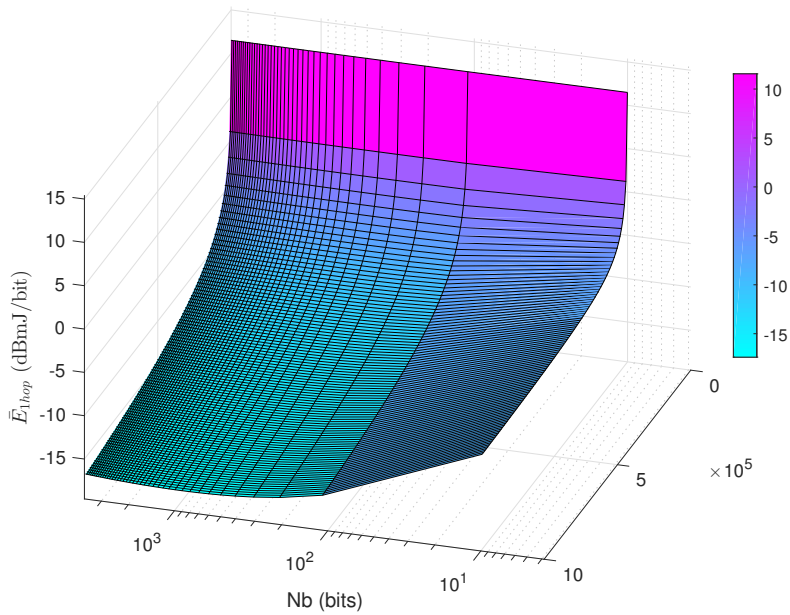


Figura 8. $\bar{E}_{1hop}$ em função da taxa de transmissão de dados (R) e em função do tamanho do pacote (N_b) para $n = 10$ nós na rede e distância $d = 10$ metros entre o par de nós comunicantes.

5. Conclusões

Este trabalho teve por finalidade investigar o comportamento do consumo médio de energia por bit para uma transmissão bem sucedida em uma rede *ad hoc* sem fio de um único salto. Uma das contribuições do estudo foi partir de uma robusta formulação de consumo

de energia empregada na literatura e ampliá-la para considerar o protocolo de acesso ao meio Aloha. Outra contribuição do trabalho foi determinar a potência ótima de transmissão em função da taxa de transmissão de dados e do tamanho do pacote que resulte reduzido consumo de energia.

Como trabalhos futuros, pretende-se investigar o cenário de um salto e de múltiplos saltos de forma multi-objetiva integrado com o protocolo MAC. Procurar-se-á ainda comparar e refinar o modelo em simuladores de redes, aplicando um protocolo MAC mais eficiente, e também testando em módulos (kits) práticos de sensores.

Agradecimentos

Este trabalho foi financiado em parte pela Fundação de Amparo a Ciência e Tecnologia do Estado de Pernambuco (FACEPE), pela Coordenação de Aperfeiçoamento de Pessoal de Nível Superior (CAPES) e pelo Conselho Nacional de Desenvolvimento Científico e Tecnológico (CNPq).

Referências

- Abdulhadi, S., Naeem, M., Jaseemuddin, M., and Anpalagan, A. (Jun. 2013). Optimized packet size for energy efficient cooperative wireless ad-hoc networks. In *IEEE International Conference on Communications Workshops (ICC)*, Budapest, Hungary.
- Akbas, A., Yildiz, H. U., Tavli, B., and Uludag, S. (2016). Joint optimization of transmission power level and packet size for wsn lifetime maximization. *IEEE Sensors Journal*, 16(12):5084–5094.
- Feeney, L. M. and Nilsson, M. (Apr. 2001). Investigating the energy consumption of a wireless network interface in an ad hoc networking environment. In *Twentieth Annual Joint Conference of the IEEE Computer and Communications Societies (INFOCOM). Proceedings*, volume 3, Anchorage, USA.
- Gao, J. (2002). Analysis of energy consumption for ad hoc wireless sensor networks using a bit-meter-per-joule metric. *IPN Progress Report*, 42(150):1–19.
- Goldsmith, A. (2005). *Wireless communications*. Cambridge university press.
- Goratti, L., Yaprak, E., Savazzi, S., and Pomalaza-Raez, C. (2014). An urn occupancy approach for modeling the energy consumption of distributed beaconing. *IEEE/ACM Transactions on Networking (TON)*, 22(1):203–216.
- Gorce, J.-M., Zhang, R., and Parvery, H. (2007). Impact of radio link unreliability on the connectivity of wireless sensor networks. *EURASIP Journal on Wireless Communications and Networking*, 2007(1):1–16.
- Gupta, A. L. and Shekoker, N. (Mar. 2016). A novel approach to improve network lifetime in wsn by energy efficient packet optimization. In *IEEE International Conference on Engineering and Technology (ICETECH)*, Coimbatore, India.
- Ito, A., Kato, T., Hatano, H., Sato, M., Watanabe, Y., Utsunomiya, E., Hiramatsu, Y., and Sato, F. (Oct. 2016). A study on designing ad hoc network protocol using bluetooth low energy. In *7th IEEE International Conference on Cognitive Infocommunications (CogInfoCom)*, Wroclaw, Poland.

- Karl, H. and Willig, A. (2007). *Protocols and architectures for wireless sensor networks*. John Wiley & Sons.
- Li, N., Lin, K., Yong, S., Chen, X., Wang, X., and Zhang, X. (Nov. 2015). Design and implementation of a mac protocol for a wearable monitoring system on human body. In *IEEE 11th International Conference on ASIC (ASICON)*, Chengdu, China.
- Liu, X., Xiong, N., Li, W., and Xie, Y. (2015). An optimization scheme of adaptive dynamic energy consumption based on joint network-channel coding in wireless sensor networks. *IEEE Sensors Journal*, 15(9):5158–5168.
- Mobin, I., Mohammed, N., and Momen, S. (Feb. 2017). Optimal range estimation for energy efficient dynamic packet size. In *International Conference on Electrical, Computer and Communication Engineering (ECCE)*, Cox’s Bazar, Bangladesh.
- Rana, K. and Zaveri, M. (2011). A-star algorithm for energy efficient routing in wireless sensor network. *Trends in Network and Communications*, pages 232–241.
- Roberts, L. (1975). Aloha packet system with and without slots and capture. *ACM SIG-COMM Computer Commun. Review*, 5(2):28–42.
- Sadiq, A. S., Almohammad, T. Z., Khadri, R. A. B. M., Ahmed, A. A., and Lloret, J. (May. 2017). An energy-efficient cross-layer approach for cloud wireless green communications. In *Second International Conference on Fog and Mobile Edge Computing (FMEC)*, Valencia, Spain.
- Shehab, A., Elhoseny, M., Sahlol, A. T., and El Aziz, M. A. (Mar. 2017). Self-organizing single-hop wireless sensor network using a genetic algorithm: Longer lifetimes and maximal throughputs. In *IEEE International Conference on Intelligent Techniques in Control, Optimization and Signal Processing (INCOS)*, Srivilliputhur, India.
- Stankovic, J. A. and He, T. (2012). Energy management in sensor networks. *Phil. Trans. R. Soc. A*, 370(1958):52–67.
- Tang, Y.-J., Lee, C.-W., Lin, M.-H., Liu, B.-H., and Tsai, M.-J. (May. 2017). Energy consumption reduction methods of geographic routing protocols with out-of-date location information in mobile ad hoc networks. In *IEEE International Conference on Communications (ICC)*, Paris, France.
- Ye, W., Heidemann, J., and Estrin, D. (Jun. 2002). An energy-efficient mac protocol for wireless sensor networks. In *Twenty-First Annual Joint Conference of the IEEE Computer and Communications Societies (INFOCOM)*. *Proceedings*, volume 3, New York, USA.
- Zhang, R. and Gorce, J.-M. (Mar. 2008). Optimal transmission range for minimum energy consumption in wireless sensor networks. In *Wireless Communications and Networking Conference (WCNC)*, Las Vegas, USA.
- Zuniga, M. and Krishnamachari, B. (Oct. 2004). Analyzing the transitional region in low power wireless links. In *First Annual IEEE Communications Society Conference on Sensor and Ad Hoc Communications and Networks (SECON)*, Santa Clara, USA.

O Problema de Detecção e Localização de Eventos em Séries Temporais Aplicado a Redes de Computadores

Diego Ximenes¹, Gabriel Mendonça¹, Gustavo H. A. Santos¹,
Edmundo de Souza e Silva¹, Rosa M.M. Leão¹, Daniel S. Menasché¹

¹Universidade Federal do Rio de Janeiro (UFRJ), Rio de Janeiro, RJ.

{diego,gabriel,gustavo,edmundo,rosam,sadoc}@land.ufrj.br

Abstract. Nowadays, ISPs provide the infrastructure for essential services. Therefore, detecting and localizing network problems automatically is an important issue. Despite current monitoring efforts, ISPs still do not have appropriate tools for coping with the non-trivial task of processing an enormous amount of data generated by QoS measurements performed dozens of times per day. We propose an approach for detecting and localizing network problems from timeseries resulting from measurements and traceroutes. Our data analysis framework is based on the so called “change point detection problem”, unsupervised learning and network topology inference.

Resumo. Atualmente provedores de serviço de Internet (ISPs) fornecem infraestrutura para diversos serviços essenciais. Portanto, é fundamental detectar e localizar problemas na rede de forma automática. Apesar de esforços de monitoramento feito pelos ISPs, eles ainda não têm as ferramentas apropriadas para processar a enorme quantidade de dado gerado por medições de QoS realizadas dezenas de vezes por dia. Essa tarefa não é trivial. Propomos uma abordagem para detectar e localizar automaticamente problemas nas redes dos ISPs a partir de séries temporais resultantes de medições de QoS e traceroutes. Nosso framework é baseado no problema denominado de change point detection, em classificação não-supervisionada e inferência de topologia de rede. Aplicamos nosso framework em dados reais obtidos durante vários meses.

1. Introdução

Provedores de serviço de Internet (ISPs) prestam um serviço de infraestrutura básica. Pelo fato da Internet prover uma infraestrutura crítica é fundamental detectar automaticamente problemas nas redes dos ISPs. Entretanto, as ferramentas disponíveis ainda não permitem uma análise precisa e automática de detecção de problemas. Na medida em que a complexidade e a capilaridade da topologia dos ISPs aumenta, os problemas de detecção, localização e correção de falhas tendem a se agravar.

É possível monitorar a qualidade de serviço através de medições realizadas em casas de usuários voluntários. As métricas de QoS podem ser obtidas a partir de *software* embarcado em roteadores residenciais conectados diretamente ao modem ou de computadores, celulares ou outros equipamentos ligados à rede doméstica. Este tipo de *software* pode realizar medições periódicas entre os roteadores residenciais e servidores localizados em pontos estratégicos da operadora. Neste caso, obtém-se milhões de amostras, cuja análise não é trivial. Por exemplo, para fins de detectar problemas na rede, a identificação

visual de mudanças de padrões nas séries temporais é desafiadora, requerendo soluções automatizadas. A detecção de mudança de padrão estatístico em uma série temporal é denominada na literatura de *change point detection problem*, as soluções para esses problemas devem ser capazes de lidar com séries temporais não estacionárias, a fim de identificar *change points* que sinalizem potenciais problemas.

A detecção automatizada de *change points*, em geral, pode se beneficiar de dados históricos sobre eventos reportados pelos administradores de rede. Neste caso tais eventos poderiam, por exemplo, indicar o início de uma mudança de comportamento de uma medida de QoS. Entretanto, um *dataset* que aponte a falha e estabeleça a correlação com as medidas fim a fim, isto é, que possibilite obter o *ground truth* dos eventos de rede, não é simples e em geral não existe. Assim, caracteriza-se como um desafio adicional a necessidade de uma abordagem não-supervisionada para a solução do problema.

Este trabalho propõe um *framework* baseado em medições fim-a-fim em redes reais e *traceroutes* para a detecção e localização de eventos de rede de maneira automática. Um evento de rede pode ter várias causas raiz como, por exemplo: um congestionamento transitório em uma ou mais regiões da rede; mudanças de rotas; alguma mudança no funcionamento de um equipamento como uma falha parcial, intermitente ou total em um roteador. Em resumo, alguma mudança que venha a afetar a qualidade do serviço percebida pelo usuário final. O procedimento de localização de eventos define um conjunto de possíveis pontos onde o evento pode ter iniciado ou terminado.

A principal contribuição deste trabalho é a elaboração de um *framework* de análise de dados capaz de detetar e localizar eventos de maneira automática. A detecção de eventos é realizada a partir da identificação de mudanças estatísticas nas séries temporais de medições **reais** de QoS de diferentes clientes. A localização dos eventos é feita a partir da correlação espaço-temporal destas mudanças considerando a topologia da rede estimada a partir das medições de *traceroute*. É importante ressaltar que nosso objetivo é a localização de eventos, isto é determinar mudanças estatísticas de QoS que tenham afetado um ou mais usuários, e as regiões prováveis que possam explicar mudanças de QoS medidas a partir dos roteadores domésticos. Não é nosso propósito a detecção das causas raiz desses eventos. O *framework* foi aplicado em dados reais e, na falta de um *dataset* de causas raiz, foi aplicada uma análise não supervisionada.

O artigo é organizado da seguinte forma. A seção 2 aborda o problema de *change point detection*. A seção 3 explica o funcionamento do *framework* de análise de dados utilizado para detecção e localização de eventos. A seção 4 mostra os resultados obtidos utilizando a técnica proposta. A revisão da literatura é apresentada na seção 5. Por fim, a seção 6 apresenta a conclusão e trabalhos futuros.

2. Change Point Detection

Algoritmos de *change point detection* têm por objetivo identificar pontos no tempo em que ocorrem alterações em propriedades estatísticas de uma série temporal, sem levar em consideração um comportamento padrão pré-definido. A literatura sobre *change point detection* é bastante vasta, sendo possível a utilização de diferentes técnicas para a identificação das mudanças estatísticas nas séries temporais, como janelas deslizantes [Kifer et al. 2004], cadeias de Markov ocultas [Kehagias 2004, Luong et al. 2013, Montañez et al. 2015] e inferência bayesiana [Fearnhead 2006,

Adams and MacKay 2007]. A técnica utilizada neste trabalho foi aplicada a partir do processo de otimização definido em [Maidstone et al. 2017]. Uma visão geral sobre o problema de *change point detection* pode ser encontrada em [Truong et al. 2018, Aminikhanghahi and Cook 2017].

Na literatura, um *change point* é um ponto em que o modelo usado para capturar características da série temporal precisa ser alterado devido a mudanças na série tal que o modelo em uso não mais descreve o comportamento observado de maneira acurada. Não é simples detectar mudanças estatísticas. Considere, por exemplo, que os valores de uma série temporal observados até um ponto x_{t^*} sigam uma distribuição gaussiana $N(\mu, \sigma_a)$ e, após o ponto x_{t^*} a variância da série temporal mude, de forma que a distribuição que mais se adéque à descrição da série seja uma nova gaussiana $N(\mu, \sigma_b)$. Neste exemplo, x_{t^*} representa o ponto a partir do qual a distribuição muda, ou seja, x_{t^*} é um *change point*. Um *outlier*, por outro lado, é um ponto ou um conjunto de pontos que fogem do comportamento padrão da série temporal, mas cujo comportamento pode ainda ser explicado pelo modelo vigente com boa precisão. Para exemplificar brevemente, suponha que em uma série temporal modelada por uma gaussiana $N(\mu, \sigma)$ o ponto x_{t_i} tenha uma probabilidade baixa segundo o modelo, mas que os pontos seguintes $(x_{t_{i+1}}, x_{t_{i+2}}, \dots, x_{t_{i+n}})$ tenham alta probabilidade de serem gerados pela gaussiana $N(\mu, \sigma)$. Desta forma, embora um *outlier* possa ter uma probabilidade baixa de ser gerado, ainda assim ele é previsto pelo modelo considerado, isto é, o evento tem uma probabilidade não nula de ocorrer, e a sua existência não significa uma mudança na distribuição. Em problemas característicos de detecção de anomalia, após a ocorrência do *outlier*, a série temporal volta ao seu comportamento original, o que não ocorre quando existe um *change point* [Su et al. 2013]. Através desse exemplo, não é difícil notar os problemas para se detectar mudanças estatísticas: qual o intervalo de tempo para considerar que houve mudanças na série temporal? como detectar e descartar *outliers*? como lidar com dados incompletos? quantas amostras são necessárias para ajustar os parâmetros do algoritmo usado?

Algoritmos de detecção de anomalia assumem que uma série temporal possui um comportamento padrão e localizam pontos que desviam deste padrão (*outliers*). O sistema Argus [Yan et al. 2012] é um exemplo de aplicação de detecção de anomalia para a identificação e localização de problemas de rede. No entanto, foi percebido nos dados coletados que mudanças na distribuição de probabilidade de métricas de qualidade de rede persistem por períodos relativamente longos de tempo. Portanto, a abordagem de *change point detection* é mais adequada para detecção de potenciais problemas. As Figuras 9(c,d) são exemplos de mudança de comportamento de séries temporais que persistem por um longo período de tempo após o *change point* detectado. Note também o comportamento periódico da série temporal presente na Figura 9(d), o que torna o problema mais difícil.

Neste trabalho, para detectar um *change point*, consideramos alterações na distribuição de probabilidade das amostras. *Outliers* não são considerados como mudanças estatísticas. O número de *change points* é desconhecido e o número de amostras dentro dos intervalos definidos por estes pontos também não é fixo. Utilizamos séries temporais univariadas de diferentes tamanhos e intervalo entre amostras não uniforme. Existem diferentes conjuntos de séries, cada uma correspondendo a uma métrica de QoS obtida de medições realizadas a partir de roteadores residenciais (por exemplo, roteadores tipo TP link) de clientes de uma rede real.

2.1. Notação

Uma série temporal composta de n pontos é definida por dois vetores, $\mathbf{x} = (x_1, \dots, x_n)$ e $\mathbf{y} = (y_1, \dots, y_n)$, onde y_i indica o valor da i -ésima amostra e x_i indica o tempo associado à i -ésima amostra. Os pontos são ordenados no tempo, ou seja, $x_{i-1} < x_i$, $i = 2, \dots, n$. Séries temporais com intervalos não-uniformes são consideradas neste trabalho. Logo, o intervalo entre amostras $x_i - x_{i-1}$ pode ser distinto para diferentes valores de i . Um subconjunto de amostras é definido através da notação $\mathbf{y}_{s:t} = (y_s, \dots, y_t)$, $s \leq t$.

A existência de k *change points* implica na separação dos dados em $k+1$ segmentos, também conhecidos como janelas. Seja τ_i o i -ésimo *change point* para $i = 1, \dots, k$, com $\tau_0 = 0$ e $\tau_{k+1} = n$. Desta forma, o i -ésimo segmento é definido por $\mathbf{y}_{\tau_{i-1}+1:\tau_i}$, assumindo que $\tau_{i-1} < \tau_i$ para $i = 1, \dots, k+1$.

Utilizando esta notação, dada uma série temporal um algoritmo de *change point detection* tem como objetivo a obtenção do número de segmentos e dos *change points*, ou seja, a obtenção de k e $\boldsymbol{\tau} = (\tau_0, \tau_1, \dots, \tau_{k+1})$.

2.2. Algoritmo de *change point detection*

Diferentes algoritmos de *change point detection* são encontrados na literatura, e cada um pode ser mais adequado para lidar com um determinado tipo de problema. Considerando as características das séries temporais de medições em redes, escolhemos na literatura um algoritmo de *change point detection* baseado em um modelo de otimização definido em [Maidstone et al. 2017]. Essa abordagem se mostrou adequada no estudo que realizamos, pois foi capaz, por exemplo, de eliminar adequadamente outliers, foi computacionalmente eficiente para lidar com o grande número de séries de que dispomos, etc. (Esse é um estudo preliminar e pretendemos continuar avaliando diferentes métodos.)

Na abordagem de [Maidstone et al. 2017], o problema é definido a partir de uma função de custo que mede a "homogeneidade" de cada janela. O processo de otimização é responsável pela escolha dos pontos de mudança que otimizem de maneira global a função de custo.

O número de pontos de mudança nos dados de medição não é conhecido previamente. Desta forma, o problema é definido a partir do caso penalizado [Maidstone et al. 2017]. A função de otimização é definida pela função de custo de cada segmento $C(\mathbf{y}_{\tau_{i-1}+1:\tau_i})$ e por uma função não-decrescente de penalidade $g(k)$, sendo o valor ótimo obtido através de programação dinâmica. A função a ser otimizada é dada por:

$$\min_{k, \boldsymbol{\tau}_{1:k}} \sum_{i=1}^{k+1} C(\mathbf{y}_{\tau_{i-1}+1:\tau_i}) + g(k) \quad (1)$$

Diversas funções de custo C e penalidade $g(k)$ podem ser utilizadas pelo processo de otimização. Escolhas comuns para a função de custo incluem o erro quadrático médio e o *negative maximum log-likelihood* de uma distribuição de probabilidade.

3. Framework

A Figura 1 ilustra os processos que compõem nosso *framework* para detecção e localização de eventos em redes de computadores. Dado um conjunto de séries temporais de medições

de QoS de diversos clientes em uma rede, a análise consiste na execução dos seguintes passos:

1. **Filtragem de clientes:** Remoção de séries com problemas, por exemplo, aquelas com um número pequeno de amostras de medição.
2. **Avaliação de correlação temporal (via *change point detection*):** Processamento das séries temporais de medições de cada cliente para identificar *change points*, denominados *eventos de clientes*. Classificação dos pontos em eventos do tipo (1) "piora", no caso de piora da métrica de QoS; (2) "melhora", caso contrário.
3. **Avaliação de correlação espacial (via *clusterização*):** Agrupamento de clientes de acordo com sua posição na rede, permitindo a posterior localização dos eventos. Chamaremos de *grupo* clientes agrupados em um mesmo *cluster* (vide Seção 3.1).
4. **Avaliação de correlação espaço-temporal:** Agrupamento de *eventos de clientes* de um mesmo grupo que sejam do mesmo tipo (piora / melhora) e tenham ocorrido aproximadamente ao mesmo tempo, identificando os *eventos de rede* (vide Seção 3.2).
5. **Análise de possível causa raiz (*root cause analysis*):** Mapeamento dos eventos de rede com a estrutura de *grupos de clientes*, visando localizar o equipamento ou grupo de equipamentos possíveis causadores do evento (vide Seção 3.3).

A avaliação de correlação temporal foi discutida na seção 2.2. Detalharemos os processos seguintes nas próximas seções.

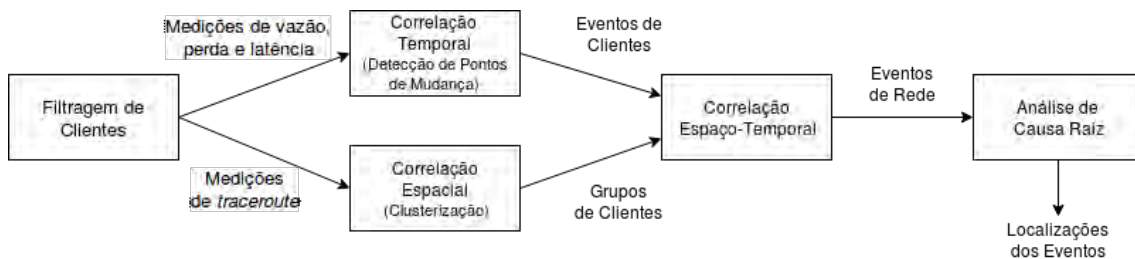


Figura 1. Visão geral dos processos que compõem o *framework* para detecção e localização de eventos em redes.

3.1. Correlação Espacial

O objetivo deste processo é agrupar clientes de acordo com sua localização na rede. Um *grupo de clientes* é formado por todos aqueles cujas rotas até o servidor de medição compartilham um determinado nó, sendo que um nó pode conter um ou mais equipamentos de rede. O processo de correlação espacial deve definir uma estrutura hierárquica de *grupos de usuários*, como exemplifica a Figura 2. A árvore criada nesta etapa será usada durante o processo de correlação espaço-temporal.

Quando a topologia do ISP é conhecida, essa tarefa torna-se mais simples. Entretanto, quando não temos acesso a essa informação podemos tentar reconstruir a estrutura da rede a partir de *traceroutes*.

A Figura 2 ilustra um exemplo de topologia real reconstruída com base em medições de *traceroute* feitas a partir dos gateways dos clientes. Os nós foram anonimizados. As arestas são orientadas dos clientes para o servidor de medição. Nesta árvore orientada, os nós com grau de entrada zero indicam equipamentos que aparecem apenas no primeiro salto (*hop*) do *traceroute* dos clientes.

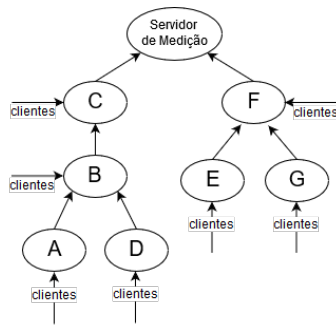


Figura 2. Exemplo de topologia reconstruída a partir de *traceroutes*.

3.2. Correlação Espaço-Temporal

A etapa de Correlação Espaço-Temporal tem como objetivo inferir *eventos de rede* que expliquem os pontos de mudança *change points* na etapa de Correlação Temporal e identificados para um mesmo *grupo de clientes* (obtido na Correlação Espacial). Exemplos de *eventos de rede* incluem a falha de um equipamento da rede, a configuração incorreta de um roteador, um congestionamento, etc.

Um mesmo *evento de rede* pode ser detetado em momentos distintos por clientes de um mesmo grupo, por exemplo, por conta do intervalo não-uniforme entre amostras das séries temporais. Portanto, é necessário elaborar um algoritmo para agrupar *eventos de clientes* de um mesmo grupo que aconteçam em um intervalo de tempo próximo.

Para resolver esse problema, criamos uma heurística para agrupamento de *eventos de clientes* detetados em um mesmo intervalo de tempo. O algoritmo proposto, denominado Votação Inexata Múltipla em Espaço Totalmente Ordenado, é inspirado no problema de Votação Inexata em Espaço Totalmente Ordenado [Parhami 1994]. Ilustramos a execução de nossa heurística na Figura 3. A cada passo do algoritmo, criamos um grupo de eventos a partir do intervalo de tempo — com tamanho máximo 2δ — que possui o maior número de *eventos de clientes* (*change points*) detetados. Para cada grupo, definimos um *evento de rede* com tempo dado pelo centro do intervalo. O algoritmo prossegue até que todos os eventos tenham sido agrupados.

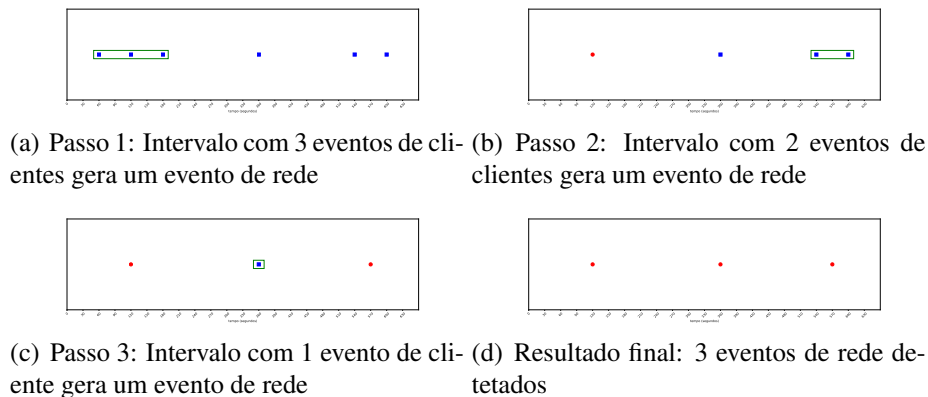


Figura 3. Exemplo de execução do algoritmo de Votação Inexata Múltipla em Espaço Totalmente Ordenado.

3.3. Análise de Possível Causa Raiz

A etapa de Análise de Causa Raiz permite a identificação de um conjunto de possíveis localizações para cada *evento de rede*, possibilitando, por exemplo, a localização de equipamentos com problema.

Para localizar eventos, precisamos relacionar *eventos de rede* de diferentes *grupos de clientes* buscando uma causa comum. Assumimos que, se um evento ocorre em um grupo, todo o tráfego do agrupamento correspondente é afetado. Essa suposição é razoável já que cada *grupo de clientes* compartilha um mesmo conjunto de equipamentos de rede. Assumimos também que o processo de correlação espacial é capaz de produzir uma estrutura hierárquica de grupos com no máximo uma aresta entre dois equipamentos.

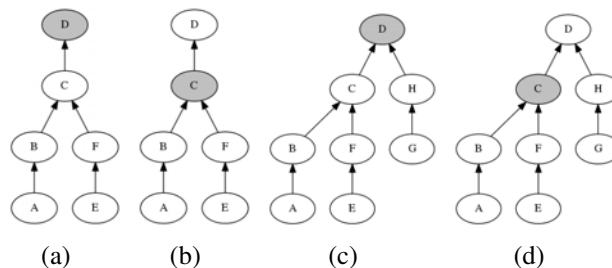


Figura 4. Exemplos de localização de eventos de rede.

Mostramos na Figura 4 alguns exemplos de topologia com a localização do *evento de rede* marcada em cinza. O procedimento para localização de *eventos de rede* se inicia nos vértices de grau de entrada zero ($\{A, E, G\}$ nos exemplos). Se uma fração pequena (parametrizável) dos clientes de um nó deteta um evento, assumimos que esse nó não é a causa do problema. Nesse caso, inferimos que os eventos detetados foram causados por um equipamento localizado antes do primeiro salto, afetando apenas alguns clientes.

Por outro lado, caso muitos clientes de um nó com grau de entrada zero detetem o evento, são possíveis três cenários: (1) o nó é a causa do problema; (2) os clientes compartilham um equipamento anterior ao nó que explica o problema; ou (3) o responsável pelo evento é um nó posterior. Quando não há informação sobre a topologia abaixo da camada IP (como no caso em que inferimos a estrutura da rede a partir de *traceroutes*), os casos (1) e (2) são indistinguíveis. No caso (3), devemos avaliar os demais nós no caminho até o servidor. Se há vários clientes ligados ao nó seguinte que não detetaram o evento, o problema certamente se encaixa nos casos (1) ou (2). Caso contrário, o procedimento é reexecutado para o nó seguinte.

De modo geral, se caminhos distintos detetam o mesmo *evento de rede*, então eles devem compartilhar ao menos um nó. Os nós que não são comuns podem ser descartados da lista de possíveis localizações.

Retornando aos exemplos da Figura 4 vemos que na Figura 4(a) um evento detetado em um subconjunto de clientes do nó A resulta na lista de possíveis locais com problemas: $\{A, B, C, D\}$. Um evento detetado por clientes de grau zero conectados ao nó E, implica nos possíveis locais com problema: $\{E, F, C, D\}$. Correlacionando as duas listas, temos $\{C, D\}$ como possíveis causas do evento. A Figura 4(b) mostra um cenário distinto (o evento ocorre em C) e o algoritmo chega à mesma conclusão anterior. Na Figura 4(c), a lista de possíveis locais construída a partir de cada nó com grau de entrada 1

mostra apenas D como causa do problema. Já na Figura 4(d), a análise encontra $\{A, B, C, D\}$ partindo de A e $\{E, F, C, D\}$ partindo de E. Como o evento não é detectado pelos clientes em G, C é a única localização possível para o problema.

4. Resultados

4.1. Dataset

A Figura 4.1 exemplifica a infraestrutura de medição a partir da qual os dados foram obtidos.

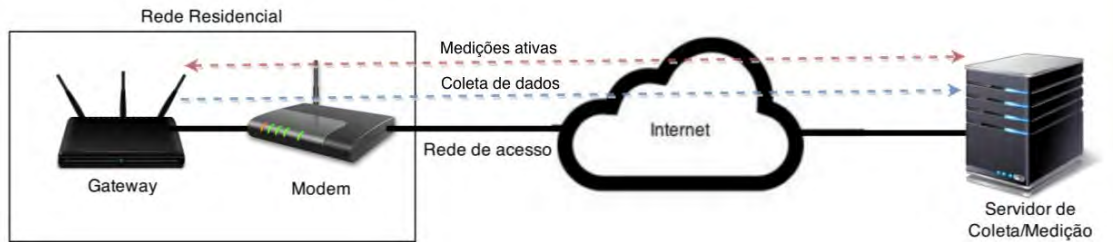


Figura 5. Infraestrutura de medição.

As medições de QoS que geraram as séries temporais consideradas foram realizadas durante 7 meses, entre maio e novembro de 2016 com mais de 2000 clientes voluntários. Foram consideradas como métricas de QoS a latência, a taxa de perda e a vazão *upstream*.

Os dados foram separados em *batches* de 10 dias. Aplicamos o algoritmo de *change point detection* para cada *batch*. As medições incluem 35 servidores espalhados em diversos estados do Brasil. O número médio de pares cliente-servidor com ao menos uma medição em cada *batch* é de 2246, sendo consideradas em média 741 séries temporais após a filtragem de usuários finais definida na seção 3.

4.2. Metodologia

Para reduzir o ruído, as séries temporais são pré-processadas antes da aplicação do algoritmo de *change point detection*. Utilizamos um filtro de medianas com janelas deslizantes centradas de tamanho w . Nesse filtro, o valor de y_i é dado pela mediana das amostras com índices entre $[i - w, i + w]$. O valor de w foi definido para cada métrica de rede após inspeção visual. Para medições de latência, a janela foi definida como $w = 13$. Para medições de vazão e perda, a janela foi definida como $w = 5$.

Os *change points* foram detectados a partir do algoritmo de otimização descrito na seção 2.2, que é aplicado considerando a série temporal completa. A falta de um *dataset* de eventos, isto é, de um *dataset* que indique instantes de tempo onde existiram problemas na rede, impossibilita uma análise supervisionada, de forma que a escolha dos parâmetros utilizados foi realizada a partir de inspeção visual. Utilizamos como função de custo C o *negative maximum log-likelihood* de uma Gaussiana. A função de penalidade varia de acordo com a métrica considerada. Para séries temporais de latência, $g(k) = 100k^2$. Para as séries temporais de perda e vazão, $g(k) = 200k$. Quando todas as amostras em uma

janela são iguais (levando a distribuições degeneradas de variância zero), adotamos um custo C constante, $C = \ln(0.000001)$.

O parâmetro δ usado na etapa de Correlação Temporal foi definido como 4 horas. Na Correlação Espaço-Temporal, consideramos que um *evento de rede* está associado a um *grupo de clientes* se ao menos 75% dos clientes do grupo geraram um *evento de cliente* na janela de tempo correspondente. Assumimos nas análises que qualquer mudança detetada por múltiplos clientes de uma mesma região em uma mesma janela de tempo têm como causa um único evento.

4.3. Eventos detetados

Nesta seção são apresentados alguns eventos detetados automaticamente pelo *framework* proposto. As séries temporais que serão mostradas foram escolhidas de forma a facilitar a identificação visual do leitor.

A Figura 6 mostra a série temporal de vazão *upstream* de dois clientes que pertencem a um mesmo vértice com grau de entrada igual a zero. Percebe-se que um evento de rede aconteceu na janela de tempo entre as linhas verticais. O vértice considerado agrupa 4 usuários diferentes e apenas os 2 usuários mostrados na figura identificaram o evento, o que indica que estes clientes compartilham um equipamento físico presente antes do primeiro *hop* identificado.

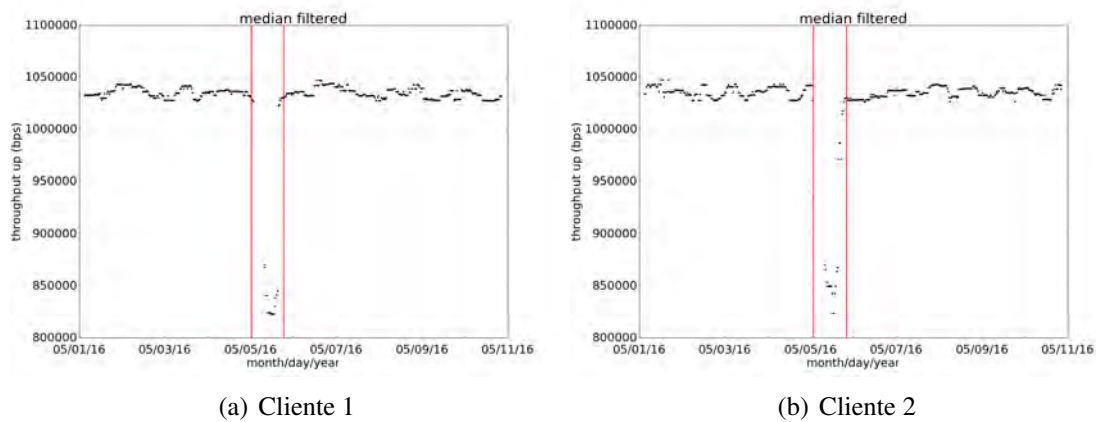
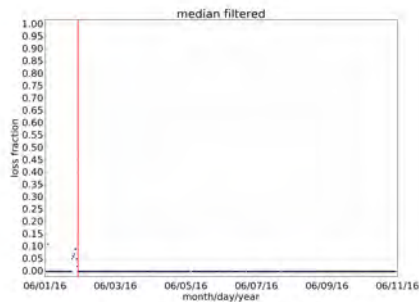
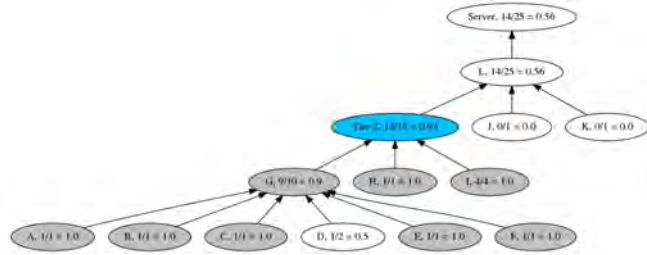


Figura 6. Evento identificado entre clientes do mesmo grupo

A Figura 7(a) mostra um evento identificado por um cliente que pertence ao vértice azul da Figura 7(b) (A Figura 7(a) aparentemente indica que o intervalo de mudança de comportamento da série seja pequeno. Entretanto esse é um problema de escala do gráfico, o intervalo corresponde a várias horas.) Cada vértice representado na figura é definido por uma tupla, onde o primeiro campo corresponde a um rótulo associado ao grupo de usuários e o segundo campo especifica a fração de clientes que detetaram o evento considerado. Os nós cinzas correspondem a possíveis locais de ocorrência do evento segundo a análise iniciada nos vértices de grau de entrada zero, sendo o nó azul identificado como o local onde o evento ocorreu após a correlação entre os resultados. Todos os nós que identificaram o evento mostraram comportamento semelhante ao da Figura 7(a).

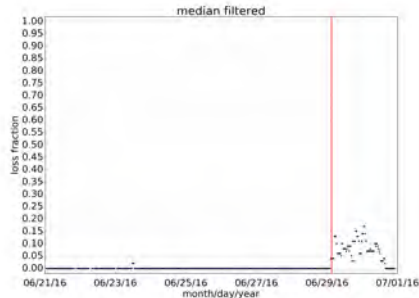


(a) Cliente 1

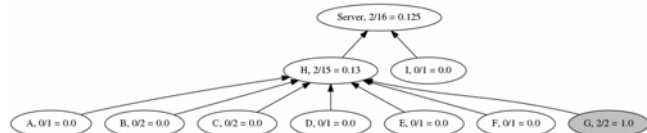


(b) Grupo de usuários identificado

Figura 7. Evento localizado a partir da correlação espaço-temporal de múltiplos grupos de clientes



(a) Cliente 1



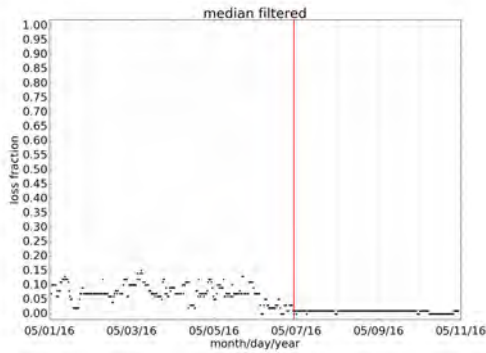
(b) Grupo de usuários identificado

Figura 8. Evento identificado em um único vértice

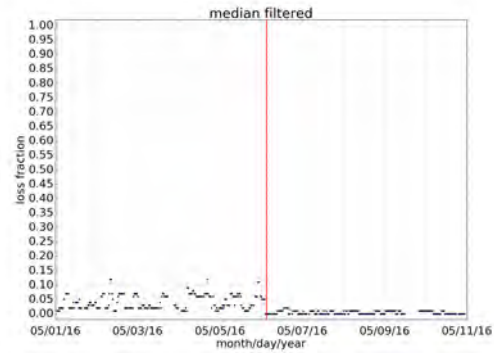
A Figura 8(a) exemplifica um evento detectado em um único vértice com grau de entrada igual a zero. Neste caso nenhuma correlação espacial com outros vértices foi identificada, o que indica que o problema é específico da região determinada pelo vértice cinza na Figura 8(b).

As Figuras 9(a) e 9(b) mostram a série temporal de perda de pacotes de 2 clientes de um mesmo vértice de grau de entrada zero. Visualmente ambas indicam um *change point* aparentemente no mesmo instante. No entanto, o cliente representado pela figura da esquerda detectou o *change point* no fim do dia 07/05/2016, enquanto o outro cliente, representado pela figura da direita, apresentou a mudança no início do dia. A diferença de tempo entre as mudanças foi maior do que a capturada pelo parâmetro δ , ou seja, o sistema identificou cada mudança como um evento distinto. Não podemos afirmar então se trata-se de um único evento ou de 2 eventos distintos. Este exemplo mostra a dificuldade de ajustar os parâmetros do algoritmo sem a existência de um *ground truth*.

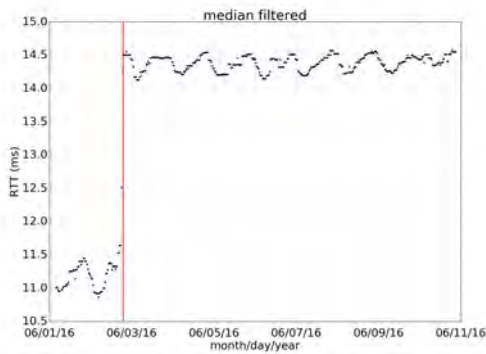
Um comportamento contra-intuitivo é exemplificado nas Figuras 9(c) e 9(d), que apresentam duas séries temporais de latência de clientes de um mesmo vértice com grau de entrada zero. Ambos os clientes apresentaram uma mudança de comportamento em um período de tempo muito próximo, mas os resultados de um cliente apresentaram melhora enquanto o outro cliente apresentou piora nos seus resultados. Neste mesmo período foi identificado um comportamento similar em outros clientes que não compartilham os mesmos atributos (servidores diferentes, caminhos de rede distintos, estados da federação



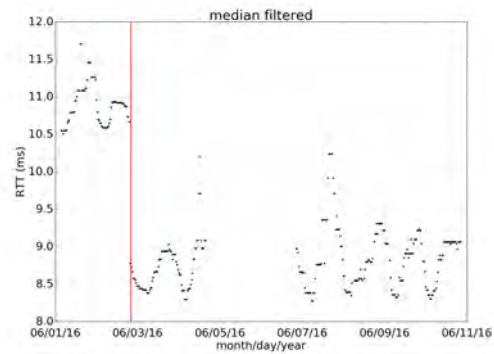
(a) Cenário 1, cliente 1



(b) Cenário 1, cliente 2



(c) Cenário 2, cliente 1



(d) Cenário 2, cliente 2

Figura 9. Cenário 1: Clientes com mesmo comportamento e intervalo fora da janela de correlação temporal. Cenário 2: Clientes de mesmo grupo com comportamentos distintos durante mesmo evento.

diferentes).

O comportamento ilustrado nas Figuras 9(c) e 9(d) pode indicar a presença de um evento global na rede que afeta diferentes regiões da rede, como por exemplo uma mudança centralizada nos parâmetros do protocolo de acesso. Como não tivemos acesso a nenhum *dataset* de eventos não foi possível confirmar a causa deste comportamento.

Vários eventos relacionados à vazão *upstream* corresponderam a detecção de um aumento substancial no valor da vazão coletada. A Figura 10 mostra um destes casos. Este tipo de evento indica mudanças na largura de banda contratada pelo assinante.

Por fim, a Figura 11 mostra um dos problemas encontrados durante a execução do *framework* proposto na rede real. É possível perceber que a presença de clientes nesta região da rede é esparsa, o que faz com que diversos agrupamentos possuam apenas um único cliente. Neste caso não foi possível identificar de maneira precisa em qual vértice o evento de rede estava localizado. Por outro lado, é possível identificar a região do possível problema que é o conjunto de nós cinzas na Figura 11(b). A escolha bem planejada dos pontos em que a rede é monitorada evita a ocorrência deste tipo de problema.

A Tabela 1 sumariza os eventos identificados pelo *framework* de análise de dados. Ela classifica os eventos conforme o tipo, melhora ou piora, de acordo com o item 2 do

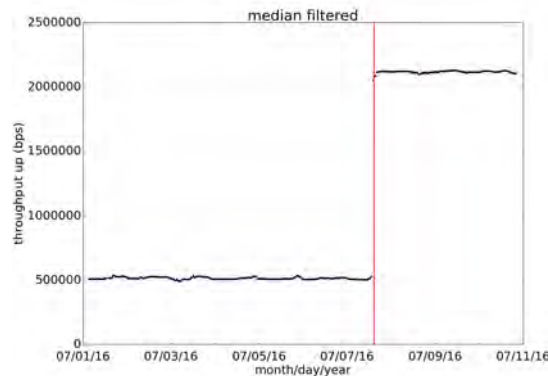


Figura 10. Possível mudança na banda contratada.

início da seção 3. Note-se que nos estudos que fizemos sobre os mais de dois mil clientes foram detetados milhares de eventos em cada uma das séries de latência, taxa de perda de pacotes e vazão máxima *upstream*.

Métrica	Eventos	
	Melhoria	Piora
Latência bidirecional	1520	1532
Taxa de perda bidirecional	310	262
Vazão máxima <i>upstream</i>	730	641

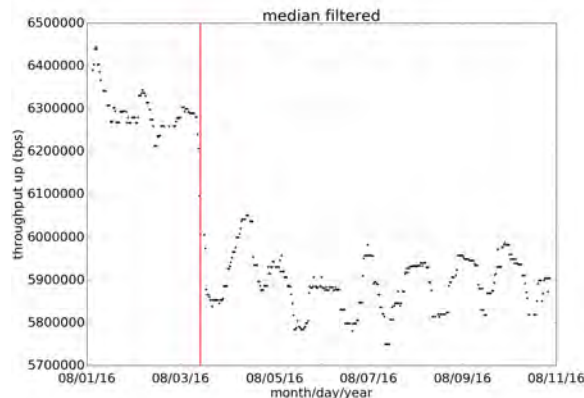
Tabela 1. Resumo dos eventos detetados

5. Trabalhos Relacionados

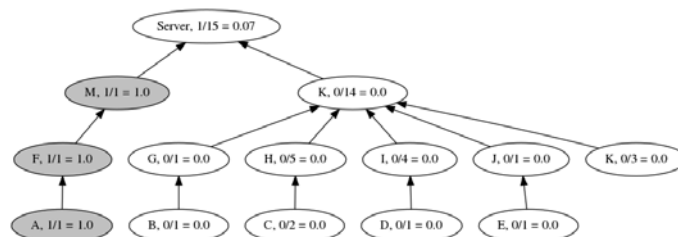
Outros trabalhos na literatura tratam do problema de detecção e localização de falhas em redes de computadores. Em [Yan et al. 2012] é definido o sistema Argus, que utiliza a informação global da rede e dados coletados passivamente para inferir a qualidade de serviço fim-a-fim. A identificação e localização de problemas é realizada a partir de técnicas de agregação temporal e agregação espacial. Diferentemente do trabalho proposto, o sistema Argus utiliza técnicas de detecção de anomalia para a identificação de eventos e não possui uma capilaridade tão grande quanto a obtida pelas medições deste trabalho.

O projeto NetNorad [Adams et al. 2016] foi desenvolvido pelo Facebook com o objetivo de automatizar a análise de falhas de rede na infraestrutura da empresa. A ferramenta utiliza medições de latência para diferentes pontos da rede utilizando ping UDP e aplica técnicas de agregação espacial para a localização de problemas. Esta ferramenta é aplicada em um contexto diferente do utilizado neste trabalho, que coleta as informações a partir de gateways residenciais e portanto abrange uma capilaridade bem maior do que uma rede interna da empresa.

Em [Choffnes et al. 2010] é proposto o *framework* de monitoramento de rede CEM. A ferramenta funciona a partir de medições realizadas por um *software* que é executado em *hosts* de usuários finais. Cada *host* captura métricas de desempenho relacionadas a um serviço específico, como vídeo sob demanda, sendo também responsabilidade dos *hosts* a identificação dos problemas locais. Após a análise local, os *hosts* enviam as informações obtidas para um sistema de armazenamento distribuído. A partir das



(a) Cliente 1



(b) Grupo de usuários identificado

Figura 11. Dificuldade de localização do evento para clientes esparsos.

informações de diversos clientes as falhas de rede são localizadas utilizando um modelo estatístico, correlacionando os resultados de maneira espacial e temporal. Medições realizadas por aplicações em *hosts* possuem diversos fatores que diminuem a robustez dos resultados, como interferência de outras aplicações concorrentes, tráfego concorrente e problemas no sinal de redes sem fio. Além disso, o *framework* CEM possui foco maior em medições passivas realizadas quando a aplicação está em uso, de forma que a frequência das medições depende do comportamento de cada usuário.

O nosso trabalho se difere da literatura quando consideramos que as métricas de qualidade são obtidas a partir de *gateways* residenciais presentes na borda da rede, que geram medições contínuas, robustas e com uma capilaridade muito maior do que a considerada nos outros trabalhos. A utilização de técnicas de *change point detection* também difere este trabalho de sistemas como o Argus, que utilizam detecção de anomalia para a identificação dos eventos.

6. Conclusão

Este trabalho propõe um *framework* de análise de dados para detecção e localização de eventos de rede. Com este objetivo, são detetadas mudanças estatísticas em séries temporais de métricas de QoS coletadas de diferentes clientes, as quais são correlacionadas utilizando a topologia inferida a partir das medições de *traceroute*. Os resultados mostram que é possível identificar e localizar os eventos utilizando apenas as métricas de QoS fim-a-fim e as medições de *traceroute* de diferentes clientes. Apesar deste ser um primeiro estudo, ele foi feito com séries reais de milhares de clientes voluntários. Em trabalhos futuros, pretendemos fazer uma comparação detalhada de diferentes métodos de detecção

de *change point*.

Métricas de qualidade de serviço fim-a-fim tem como principal vantagem a relação direta com a experiência percebida pelo usuário. Uma possível extensão deste trabalho é a criação de métodos para a priorização dos eventos de rede que causem maior impacto na qualidade de experiência do usuário final. Também é de interesse a análise da correlação entre as diferentes métricas de QoS utilizadas para a detecção de eventos.

Por fim, a criação de um *dataset* de eventos de rede tornaria possível a realização de uma análise supervisionada para a escolha dos parâmetros do modelo. O presente trabalho mostrou que o uso de técnicas de *change point detection* é uma ferramenta viável para detetar problemas em redes de ISPs.

Referências

- Adams, A., Lapukhov, P., and Zeng, J. H. (2016). NetNORAD: Troubleshooting networks via end-to-end probing. <https://code.facebook.com/posts/1534350660228025/netnorad-troubleshooting-networks-via-end-to-end-probing/>.
- Adams, R. P. and MacKay, D. J. (2007). Bayesian online changepoint detection. *arXiv preprint arXiv:0710.3742*.
- Aminikhanghahi, S. and Cook, D. J. (2017). A survey of methods for time series change point detection. *Knowledge and information systems*, 51(2):339–367.
- Choffnes, D. R., Bustamante, F. E., and Ge, Z. (2010). Crowdsourcing service-level network event monitoring. *SIGCOMM Comput. Commun. Rev.*, 40(4):387–398.
- Fearnhead, P. (2006). Exact and efficient bayesian inference for multiple changepoint problems. *Statistics and Computing*, 16(2):203–213.
- Kehagias, A. (2004). A hidden markov model segmentation procedure for hydrological and environmental time series. *Stochastic Environmental Research and Risk Assessment*, 18(2):117–130.
- Kifer, D., Ben-David, S., and Gehrke, J. (2004). Detecting change in data streams. In *Proceedings of the Thirtieth International Conference on Very Large Data Bases - Volume 30, VLDB '04*, pages 180–191. VLDB Endowment.
- Luong, T. M., Rozenholc, Y., and Nuel, G. (2013). Fast estimation of posterior probabilities in change-point analysis through a constrained hidden markov model. *Computational Statistics and Data Analysis*, 68:129 – 140.
- Maidstone, R., Hocking, T., Rigai, G., and Fearnhead, P. (2017). On optimal multiple changepoint algorithms for large data. *Statistics and Computing*, 27(2):519–533.
- Montañez, G. D., Amizadeh, S., and Laptev, N. (2015). Inertial hidden markov models: Modeling change in multivariate time series. In *Proceedings of the Twenty-Ninth AAAI Conference on Artificial Intelligence, AAAI'15*, pages 1819–1825. AAAI Press.
- Parhami, B. (1994). Voting algorithms. *IEEE transactions on reliability*, 43(4):617–629.
- Su, W.-x., Zhu, Y.-l., Liu, F., and Hu, K.-y. (2013). On-line outlier and change point detection for time series. *Journal of Central South University*, 20(1):114–122.
- Truong, C., Oudre, L., and Vayatis, N. (2018). A review of change point detection methods. *arXiv preprint arXiv:1801.00718*.
- Yan, H., Flavel, A., Ge, Z., Gerber, A., Massey, D., Papadopoulos, C., Shah, H., and Yates, J. (2012). Argus: End-to-end service anomaly detection and localization from an isp's point of view. In *2012 Proceedings IEEE INFOCOM*, pages 2756–2760.

Ranking Influential and Influenced Shares Based on the Transfer Entropy Network

José de Paula Neves Neto¹, Daniel Ratton Figueiredo¹

¹ Programa de Engenharia de Sistemas e Computação (PESC)
Instituto Alberto Luiz Coimbra de Pós-Graduação e Pesquisa de Engenharia (COPPE)
Universidade Federal do Rio de Janeiro (UFRJ)
Rio de Janeiro – RJ – Brazil

{neves,daniel}@cos.ufrj.br

Abstract. *Influence is a concept found in nature and society and is related to the interdependency among a set of objects. In the context of a stock market, the variation in price of shares can influence the variation in price of other shares, leading to influential and influenced shares. In this work we leverage the notion of transfer entropy to build a network of shares and pairwise directed influence that is used to rank the most influential and influenced shares. Classical network centrality metrics such as PageRank and HITS are leveraged to rank the nodes. We apply our methodology to the shares in the greater stock market in Brazil, we rank nodes to find source and destination of influence in that market, while also comparing the different rankings and their correlation with traded volume.*

1. Introduction

Influence is a fundamental concept in nature and society as all objects both exert and suffer some kind of influence from one another. Human intuition says that climate influences crop and unemployment rate influences wages, for example. Indeed, there are many situations in which the action of an object clearly affects the behavior of another. This kind of relationship between objects is what we refer to as *influence*.

Influence can also be seen as *information*. In particular, when one object influences another, one can say that the first object has passed information to the second. Thus, influence can be interpreted as *information transfer*. An approach to quantify this kind of influence has been recently proposed, known as *transfer entropy*, and is being widely applied in different contexts [Bossomaier et al. 2016].

Shares traded in a stock market intuitively influence one another, as the price variation in a share may trigger (or be correlated with) a subsequent price variation in another share. Our focus is the *influence* between shares in a stock market exchange. By considering a pairwise directed measure of influence, we construct a directed weighted network and leverage its structure to determine the most *influential* and the most *influenced* shares in the market.

To achieve this goal, we consider the time series to represent the price change (not the price value) of shares and measure the information transfer between every two ordered pair of shares. We construct a directed weighted network where nodes are shares and edges weights represent transfer entropy values. Thus, we can analyze this network using classic centrality measures to assess the importance of nodes (shares).

Note that the edge weight $w_{i,j}$ denotes how much share i influences share j . Intuitively the sum of incoming edges to node i characterizes how influenced is this share, while the sum of outgoing edges of node i characterizes its influence power. Ranking shares based on its influence power or its subjection to influence yields a metric of *importance* for shares.

The ranking of shares can be done using algorithms like PageRank or HITS, originally developed for ranking web pages [Brin and Page 1998, Page et al. 1999, Kleinberg 1999], which leverage the weights and structure of the network. Moreover, nodes can also be ranked by their incoming and outgoing total (sum) edge weights, here called *node weight*.

We apply this methodology to shares traded in the BM&FBOVESPA, the most important stock market in Brazil. Using public available data (time series of share prices over a 21 year period), we measure pairwise transfer entropy and construct the network. We rank nodes according to three different metrics yielding for each metric a ranking for influential and influenced shares. We assess the coherency among the top ranked shares across the different rankings, to better characterize influenced and influential shares. We also investigate the relationship between influence (according to the ranking of different metrics) and traded volume of shares, and results indicate there is very little correlation.

The novelty of this paper is to consider the relation between movements of different shares instead of the relation between the price of shares. Intuitively, an investor or a stockbroker observes the movement (and its bias) to make their subsequent decision about buying or selling a stock, based on the belief that it goes up or down in the future. We have not found other papers using actually the same approach. But, in this way, we have another measure to assess importance of stock market shares.

The reminder of this paper is organized as follows: In Section 2 we revise some fundamentals and other related works. The proposed methodology is presented in Section 3. Section 4 presents the empirical data that was collected and analyzed. Section 5 exposes the results of the different rankings and correlation analysis. Finally, Section 6 summarizes the work.

2. Related Works

2.1. Measuring Dependency

There are many ways to measure dependency and information and possibly the most widely accepted notion is that of *entropy*, introduced by Shannon [Shannon 1948], to measure information as follows. Given a random variable I , with probability distribution $p(i)$, the Shannon entropy H_I is defined as

$$H_I = - \sum_{i \in D(I)} p(i) \log_2 p(i) \quad (1)$$

where $D(I)$ is the image of random variable I . The larger the value of H_I the more information the random variable carries. In particular, the maximum value of H_I is achieved for a uniform random variable.

When dealing with two objects a useful measure is the *mutual information*

[Shannon 1948]. The mutual information M_{IJ} of two random variables I and J , with distribution $p(i)$ and $p(j)$ respectively, and joint-probability $p(i, j)$ is defined as

$$M_{IJ} = \sum_{\substack{i \in D(I) \\ j \in D(J)}} p(i, j) \log_2 \frac{p(i, j)}{p(i)p(j)} \quad (2)$$

Mutual information does not suit our goals because it does not consider the direction of the relationship nor the causality (which comes after) between the two objects. To account for these notions we use instead the concept of transfer entropy, defined below.

2.2. Transfer entropy

Transfer entropy was introduced by Schreiber [Schreiber 2000] and is based on the concept of *entropy* and *mutual information*. The novelty of Schreiber's proposal is the introduction of a time delay between the observed variables, a concept useful in several applications [Bossomaier et al. 2016]. Thus, it considers an object (event) that occurs "after" the others.

The transfer entropy $T_{J \rightarrow I}$ from random variable J to random variable I is given by

$$T_{J \rightarrow I} = \sum_{\substack{i_+, i \in D(I) \\ j \in D(J)}} p(i_+, i, j) \log_2 \frac{p(i_+ | i, j)}{p(i_+ | i)} \quad (3)$$

where i_+ is the value taken by random variable I in its next occurrence. Note that transfer entropy measures the ratio in the conditional probabilities of i_+ given i when j is also observed. Thus, if observing j allows to increase the probability of a particular i_+ , reducing its entropy, we have transferred information from J to I .

2.3. Computing Transfer Entropy

Consider two time series I and J , both with length N . In order to apply Equation 3 to empirical data the (joint and conditional) probabilities of $p(i_+, i, j)$, $p(i_+ | i, j)$ and $p(i_+ | i)$, where $i_+, i \in I$ and $j \in J$, must be properly estimated. This is done using the time series to compute the relative frequency of the observed values.

To ease calculation, those probabilities are transformed into their versions without conditionals, yielding only the (joint) empirical probabilities $p(i)$, $p(i_+, i)$, $p(i, j)$ and $p(i_+, i, j)$. Thus, the number of occurrences for each instance (single value, pair of values and triple values) is counted across the two time series to determine the corresponding relative frequencies, which are used as their probabilities.

Finally, the transfer entropy from J to I is determined by the summation described below, running over all values contained in I and J series.

$$T_{J \rightarrow I} = \sum_{\substack{i_{n+1}, i_n \in D(I) \\ j_n \in D(J)}} p(i_{n+1}, i_n, j_n) \log_2 \frac{p(i_{n+1}, i_n, j_n) \cdot p(i_n)}{p(i_{n+1}, i_n) \cdot p(i_n, j_n)} \quad (4)$$

where $n \in 1, 2, \dots, N - 1$ and i_k, j_k are the k -th values into I and J series, respectively.

As discussed ahead in Sub-section 3.2, values for i_{n+1} , i_n and j_n are discrete and can assume only four different symbols, instead of a larger number of values or even a continuous scale, because this would enlarge the space of possible combinations of values, and would hamper estimation of respective probabilities and, consequently, the transfer entropy computation would be more noisy.

2.4. Applications of transfer entropy

Market networks can be defined as the network of equities created by enterprises and their relationships within the financial market. Market networks take advantage of network analysis tools and metrics which are leveraged to analyze the market, understand relationship between enterprises, support forecasting and portfolio strategies [Baker 1990, Kauê Dal'Maso Peron et al. 2012].

There are various works on market network analysis [Tabak et al. 2010, Leahy et al. 2014], market network measurements [Lee and Djauhari 2012], market network correlations [Shirokikh et al. 2013, Lyócsa et al. 2012, Namaki et al. 2011, Boginski et al. 2005], stock market network forecasting [Atsalakis and Valavanis 2009] and financial modeling [Petelin et al. 2011]. For example, Tabak et al. [Tabak et al. 2010] considers a market network where nodes are shares in a stock market and undirected edge weights denote a distance based on cross-correlation between the prices time series. In particular, correlation values of $+1$, 0 and -1 were mapped into distances of 0 , 1.4142 and 2 , respectively, which are then used to compute the Minimal Spanning Tree (MST) of the network.

Approaches based on transfer entropy have recently increased and are being addressed in different scenarios, including financial markets [Bossomaier et al. 2016, Schreiber 2000, Yang et al. 2017]. In particular, there are recent studies on information transfer between stock markets (of different countries) using transfer entropy methods [Kwon and Yang 2008, Marschinski and Kantz 2002, Yang et al. 2017, Korbel et al. 2017, He and Shang 2017], including the use of variations like *Effective Transfer Entropy*, *Phase Transfer Entropy*, *Rényi's Transfer Entropy* and *Effective Phase Transfer Entropy*. However, our focus is information transfer between shares of the same market.

2.5. Node Ranking in Networks

There are many available models and metrics for ranking nodes in directed weighted networks. Directed edges give us to distinguish two types of nodes in the network: *authorities* and *hubs*. Authorities are nodes which act as sinks, receiving many links (or valuable ones), while hubs are nodes with many outgoing edges (or valuable ones).

In order to rank network nodes two techniques that can be applied are PageRank [Page et al. 1999] and HITS [Kleinberg 1999, Chakrabarti et al. 1998]. PageRank ranks web pages by their importance, considering the importance of the pages that point to it. HITS has a similar recursive structure, but considering that nodes are both hubs and authorities, and uses this categorization in defining importance. In particular, an important authority is a node that is pointed to by important hubs. Similarly, a hub is important if it points to important authorities.

t	0	1	2	3	4	5	6	7	8	9	10
OP	8.69	0.28	9.18	5.66	2.77		1.51		8.85	8.88	4.98
CP	0.28	9.18	5.66	2.77	7.18		8.85		8.88	4.98	3.12
M	down	up	down	down	up	null	up	null	draw	down	down

t = time; OP = opening price; CP = closing price; M = movement symbol

Figure 1. Mapping stock prices series into stock biases series.

Since the graph that represents the market network is complete (all edges are present), such algorithms are only meaningful in their extended version that uses edge weights: *weighted PageRank* [Xing and Ghorbani 2004] and *weighted Hyperlink-induced topic search* [Li et al. 2002, Zhang et al. 2007].

3. Methodology

3.1. Time series considered

The basic time series element considered in many analysis is not the stock price movement, but its value. However, our interest has been on how a stock movement can influence a movement of another stock. In other words, does the price change of a certain stock today influence the price change of another stock tomorrow?

Several studies make comparisons between stock prices which are synchronized in time, that is, prices of two stocks A and B are compared in the same time interval $t_i, \dots, t_f$, where t_i and t_f are initial and final time instants, respectively. Such studies search for a kind of binding between two stocks, trying to understand how and why they move together over time, possibly being influenced by a third party.

On the other hand, our study focuses on stock price movements. They are preferable to individual price values because we focus on understanding influence among stocks in a market. Trading decisions (to sell or to buy) a stock occur after observation and analysis (foresee) of a previous stock movement. There is clearly a time shift between the observed fact (one share movement) and the action (other share trading). Thus, the model we propose follows along this intuition.

In order to understand how past movements influence future movements we will consider a time shift between two time series. We convert prices into symbols that will represent our basic series elements, namely movements of a bond price.

3.2. Symbols

We will adopt a set of symbols to investigate influence in stock price movements. Using the daily *opening* and *closing* prices for shares we map a price series (float type) into a symbol series (discrete type) as shown in Figure 1. If a stock price goes up from market opening time to its close time, we have an *up* symbol for that stock in that day; if it goes down, we have a *down* symbol; if it remains *almost the same*, we have a *draw* symbol; and there are cases where we have no opening neither/or closing price for the share, because it is not traded in that day, and in this case we have a *null* symbol. But, in this way, we have another measure to assess importance of stock market shares.

As a remark, when we say “*almost the same*”, we mean prices whose opening and closing values are similar, with a value difference smaller than $\varepsilon > 0$. This avoids false positives for irrelevant *up* or *down* movements. Actually ε is an arbitrary value, but it is reasonably coherent to use a factor of $\varepsilon = 4.5 \cdot 10^{-5} = 0.0045\%$ because it represents a daily percentage that corresponds to a 1% (one percent) monthly variation divided by 22 (twenty two) work days of a month.

Therefore each stock has a time series in which elements are one of the four symbols and its length is the number of work days in time period.

3.3. Pairwise Transfer Entropy

The time series of movements will be used to compute the transfer entropy for each pair of stocks. The idea is to measure the transfer of information from one share to the other one, in a pairwise comparison. Notice that this is a directed measure, so we can evaluate the transfer of information in both directions. The transfer entropy will be calculated using a lag of one position in the series (*e.g.*, one day).

3.4. The Directed Network and Ranking

We will consider all ordered pairs of shares, and thus, our model generates a complete directed graph where nodes are shares and directed links have weights that correspond to the transfer entropy of the ordered pair.

In order to rank nodes and capture their influence in the network, we use PageRank and HITS. Given the directionality of edges, *hubs* are stocks that *influence* many others hubs, whereas *authorities* are stocks that are *influenced* by many others.

PageRank generates a measure, through a recursive method, where a node is more important the more important are the nodes that point to it, which is the similar idea of an authority in HITS.

Another possible metric to rank nodes is to give every node a number that is equal to the sum of all its incoming weighted edges. Nodes with higher numbers can be seen as authorities as well. Analogously, we can give every node a number equal to the sum of all its outgoing weighted edges, where nodes with high numbers can be seen as hubs.

4. Collected Data

BM&FBOVESPA, the main Brazilian stock exchange, is the target of this work. There are about 700 public companies which negotiate their bonds in BM&FBOVESPA and such information, including bond prices, is publicly available [bmf 2017].

Companies can have a large number of different bonds negotiated in the market and are also responsible for various actions concerning them, such as creation, discontinuation, changing names, joining, or splitting bonds. However, such activities impose limitations in the data analysis since records are not always publicly available, being hard to track names changes, or bonds that have joined or split. This poses some difficulty in tracking a particular bond over long time periods.

4.1. Dealt shares and records

BM&FBOVESPA offers an electronic data set containing daily transactions for each share since August 1980. Our initial evaluation revealed over 40,000 different bonds (each with

a code) negotiated in the market within that period, where each stock within BM&FBOVESPA is identified by a unique code. However, many of them have a short life time (one month, for example) or a very low *liquidity* (not traded frequently). In terms of an analysis over a large time period, shares with one of these two characteristics are not interesting.

Therefore, in the present work, two main filters were applied to the original data. These filters identify stocks that are more relevant to the present study of influence in the market, discarding stocks that are of less importance. The following procedure reduced the amount of codes from over 40,000 to 210:

1. Only those share codes that were negotiated in the last month of the period (December 2016) are considered;
2. Only those share codes that were dealt at least in 50% of the work days in the period are considered.

5. Results

In this section, results obtained by applying the proposed methodology to data from BM&FBOVESPA are presented and discussed. We first discuss general properties of nodes and edges of the market graph, and then ranking of stocks based on three metrics.

5.1. Edges: transfer entropy between stocks

We start by presenting results on the edge weights among all the shares (a total of 43,890 edges). Recall that directed edge weights represent transfer entropy values between the respective shares, according to the direction of information transfer.

Figure 2(a) shows the complementary cumulative distribution function of edge weights. The additional colored grid lines indicate 50% (red), 10% (green) and 5% (yellow) of total points of the distributions. Note that a small amount of edges have a much larger value. The 20 largest values are represented by the green dots and their identification is shown in the box. Note that 50% of the values have a transfer entropy less than 4.27×10^{-3} , whereas 95% have a value less than 3.98×10^{-2} , with an average of 9.71×10^{-3} . Negative values are possible in the calculations and indicates that basically no information is transferred.

Note that some shares appear multiple times as the source of information transfer among the largest edges, while other shares appear multiple times as the target, suggesting that there are bonds with greater presence in each side of information transfer. For example, **ITUB4T** (2), **CSAN3** (2), **CSAN3F** (3) and **EMBR3** (2) appear more than once as source of information transfer among the top 20 (in parentheses one can see number of appearances) of *most influential* shares. Yet when considering not the shares but the organization, aggregating their bonds, **ITUB** (2), **CYRE** (2), **CSAN** (5), **ENBR** (3) and **LREN** (2) occur several times as influential source of information transfer.

Figure 2(b) shows all transfer entropy values at once, in matrix form, with values in logarithmic scale. For example, the four shares mentioned above are in lines: **ITUB4T** (118), **CSAN3** (53), **CSAN3F** (54) and **EMBR3** (74). In general, a “lighter” line suggests that the corresponding share is not an important source of information transfer, whereas a “darker” line indicate that the share transfers much information to other shares. Analogously, a “lighter” column suggests that the share does not receive much information from other shares, and a “darker” column suggests a share that receive information from many other shares, being therefore non susceptible to the action of others.

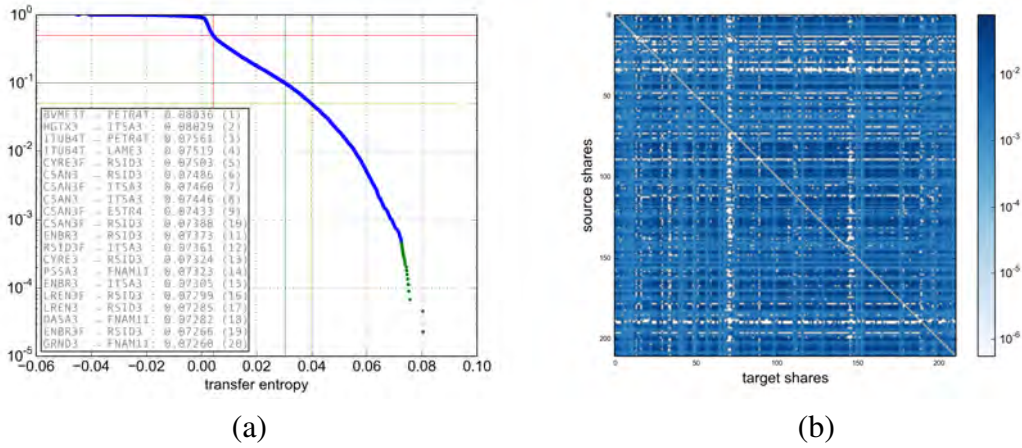


Figure 2. Weight values for all shares: (a) complementary cumulative distribution function, in semi-log scale; (b) color map, in log scale.

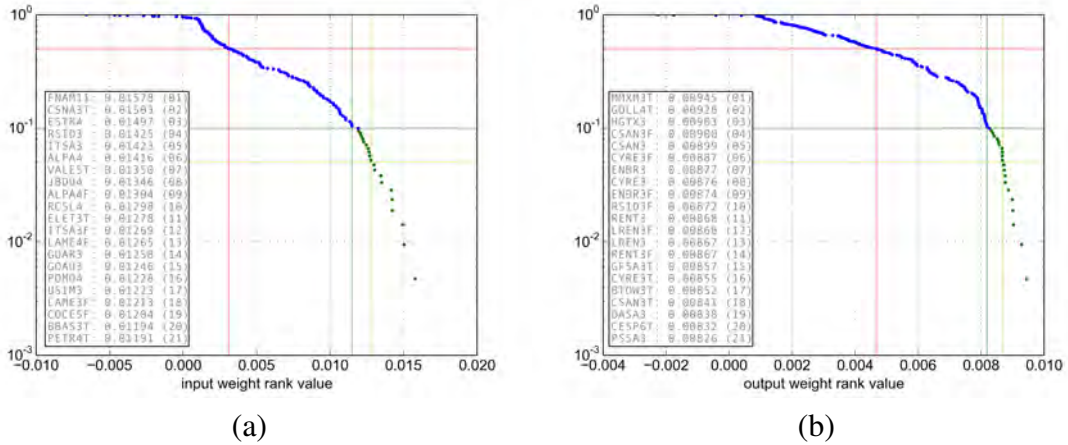


Figure 3. Complementary cumulative distribution function of node weights in semi-log scale: (a) incoming weight (influenced by others); (b) outgoing weight (influence others).

5.2. Incoming and outgoing weight

The outgoing weight of a node is given by the sum of all its outgoing edge weights. Analogously, the incoming weight of a node is the sum of all its incoming edges.

The outgoing and incoming weights of nodes allows to understand their importance in transferring (both sending and receiving) information in the market network. Figure 3 shows the complementary cumulative distribution for incoming and outgoing weights over all nodes. Thus, with respect to each share is now ranked both in terms of being *influenced* or *influential* of information transfer.

Shares with higher incoming weight are those that are more susceptible to changes of other shares, whereas shares with lower incoming weight are relatively robust to other shares movements. Both kinds of shares are interesting, depending on the objectives and market expectations. Shares with higher outgoing weight are the most influential ones, whereas shares with lower outgoing weight are relatively invisible to the other shares. In both cases the tail of the distribution shows that there are very few shares with weights

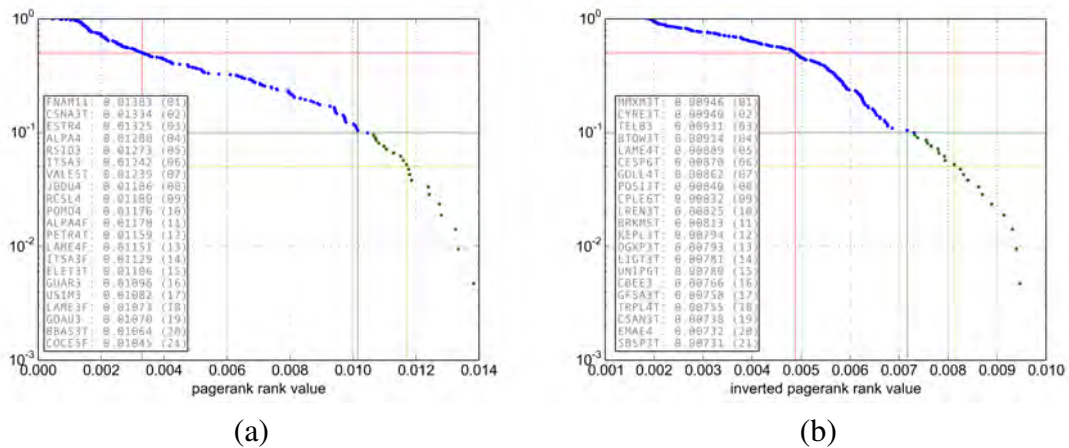


Figure 4. Complementary cumulative distribution function for PageRank values of shares in semi-log scale: (a) original network (influenced by other); (b) reverse edge network (influence others).

much larger than the average. This indicates that only a few shares are really influential and really influenced. Interestingly, there is no overlap between the top 20 largest incoming and outgoing weight.

5.3. Using PageRank algorithm

The weighted version of PageRank algorithm was also used to rank the shares, taking the complete network as input. Note that the weighted PageRank algorithm computes how important a share is with respect to being the target of information transfer or, in other words, how much the market influence the share. Larger PageRank values indicate shares that are more influenced.

In order to capture how influential are the shares, we reverse the direction of the edges in the network. With this transformation, a node with many heavy incoming edges is a great source of information transfer. We run the weighted version of PageRank in the network to identify the most influential nodes.

Figure 4(a) shows the cumulative distribution of PageRank values for the market graph, whereas Figure 4(b) shows the PageRank value distribution for the reverse edge graph. We again observe that a few nodes have PageRank values much larger than the average, being much influenced by others (Figure 4(a)), or much influential (Figure 4(b)). These results can be compared to those in Figure 3 which shows incoming and outgoing node weight.

5.4. Using HITS algorithm

We also use HITS algorithm to rank shares in the network [Chakrabarti et al. 1998]. HITS algorithm assigns two numbers to every node in the network, namely a hub factor and an authority factor. A *hub* is related to the *source*, and an *authority* to the *target* of information transfer. Figure 5 shows the distribution for hub and authority values. As PageRank, HITS shows that there are some shares whose values are very far from the average.

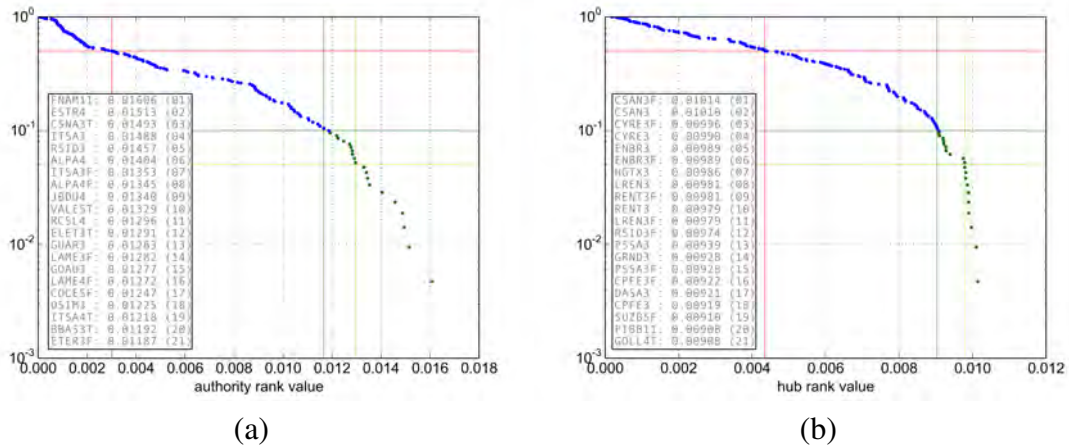


Figure 5. Complementary cumulative distribution function for HITS values of shares in semi-log scale: (a) authorities (are influenced by other); (b) hubs (influence other).

Type of value	weight vs. PageRank	PageRank vs. HITS	HITS vs. weight	weight vs. HITS vs. PageRank
incoming	0.9048 (19)	0.8182 (18)	0.9048 (19)	0.8182 (18)
outgoing	0.2121 (7)	0 (0)	0.4815 (13)	0 (0)

Table 1. Jaccard coefficient for the top-20 shares among different ranking metrics, for incoming (influenced) and outgoing (influential) rankings.

5.5. Comparing between ranking methods

Note a coherence in the tail of the incoming weight distribution (Figure 3) and PageRank (Figure 4). In top 20 shares of these two rankings, 19 shares are common to both rankings. Both rankings have the exact same first three positions, **NAM11**, **CSNA3T** and **ESTR4**, in order, whereas others shares slightly change their position. HITS shows also consistent results with node weight and PageRank rankings.

However, the outgoing weight and reverse edge PageRank rankings do not show the same level of coherence. Among the 20 top shares in each ranking, only 7 are common to both, less than 50%. Analogously, in HITS, a share considered a top hub is not in the top of the ranking of the other two metrics. However, there is some consistency between hubs and outgoing weights.

The Jaccard coefficient was used to compare the top 20 incoming and outgoing values between the different rankings, summed up in Table 1. The first three columns show the comparison between pairs of rankings, while forth column compare all rankings together. In parentheses we have the number of coincident shares among the rankings.

The Spearman correlation coefficient (ρ) between rankings can also endorse the similarity between the rankings. Note some coherence for the different rankings are shown in Table 2. The Spearman coefficient was calculated using the entire rankings of all shares in the network.

The top 20 most influenced shares are the same, no matter what ranking metric is used. On the other hand, the most influential shares depend more on the ranking metric,

Metric	weight vs. PageRank	PageRank vs. HITS	HITS vs. weight
incoming	0.974	0.980	0.995
outgoing	0.779	0.687	0.984

Table 2. Spearman's ρ between rankings.

Metric	weight	PageRank	HITS
Spearman's ρ	0.033	-0.110	0.045

Table 3. Spearman's ρ between sensibility to influence (ranking) and influence power (ranking) of a share considering the same metric.

despite some agreement between HITS and node weights.

Finally, in general the influence power of a share does not agree with its sensibility to influence. In general, these things do not go together as we can see in Table 3, endorsing our intuition that says that the most influenced shares are not the most influential.

5.6. Computing financial volume

An important consideration is the relationship between the financial volume of the shares and their influence with respect to information transfer. Indeed, some shares have very large volumes (compared to the average) which could be related to their influence in the market.

Figure 6 shows the distribution of the total traded volume of the shares in the market for the observed period. It is notorious the volume of the most traded shares, for example, the volume of the top two shares, **PETR4** and **VALE5**, is one order of magnitude larger than 90% of the smaller shares! The top 6 shares by traded volume are shown in Table 4, along their position in every ranking metric (recall that the network has 210 shares).

Table 4 shows that financial volume and influence are not correlated. A highly traded share is not one that the most influences or is influenced by others. Indeed, the Spearman's coefficient between financial volume ranking and each one of the ranking metrics is shown in Table 5. Indeed, the very small values for the coefficient indicates that there is no correlation between the traded volume of a share and its influence in terms of information transfer.

Ranking method	PETR4	VALE5	BBDC4	PETR3	VALE3	BBAS3
incoming weight	163	155	138	146	69	131
outgoing weight	197	192	201	196	176	194
normal PageRank	175	163	141	153	68	140
inverted PageRank	189	200	201	206	177	197
HITS authority	163	156	135	149	69	133
HITS hub	200	188	199	195	175	193

Table 4. Position of certain high financial volume bonds in the lists of applied rank methods, considering 210 bonds.

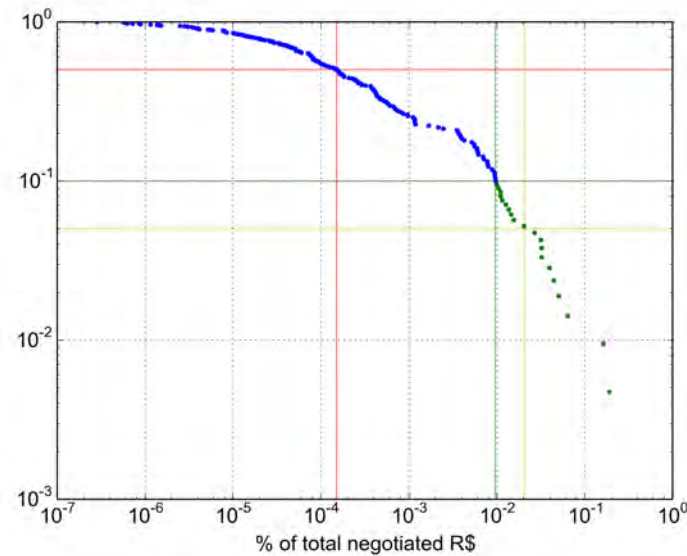


Figure 6. Complementary cumulative distribution function for market shares financial values (volume) in log scale.

Metric	volume vs. weight	volume vs. PageRank	volume vs. HITS
incoming	-0.108	-0.141	-0.124
outgoing	-0.101	-0.143	-0.108

Table 5. Spearman's ρ between volume ranking and influence rankings.

6. Conclusions

Transfer entropy has recently emerged as an ubiquitous metric to measure the influence between two dynamics, capturing the notion of information transfer. This work has leveraged this metric to measure influence among shares traded in the stock market exchange, BM&FBOVESPA. In particular, we have considered a 21-years daily record of stock price movement to characterize the influence (or information transfer) between shares in the market. Beyond computing the pairwise transfer entropy among the shares, we proposed a network-based approach to identify both shares that are influential and shares that are influenced across the market.

We construct a market graph, where shares correspond to nodes and transfer entropy values represent weighted directed edges between shares. Classic network centrality metrics such as PageRank and HITS were used to rank nodes, both in terms of incoming and outgoing edges, revealing influential and influenced shares in the market. We show that a small fraction of the shares have ranking metric values that are significantly larger than average, indicating their prevalence as either influential or influenced stocks. This observation is consistent across rankings.

We also find very good consistency between the three rankings concerning the most influenced shares in the market (top 20), while the agreement concerning the most influential is less pronounced among the three rankings. Moreover, our analysis indicates that the most traded shares (financial volume) in the market are not the most influential nor the most influenced, indicating a lack of correlation between traded volume and

information transfer.

The market network was constructed using the entire time series of price movements, across a period of 21 years. As future work, we plan to characterize influence over time by considering shorter time series and understanding the persistence of top influential and influenced shares.

References

- (2017). Bolsa de valores, mercadorias e futuros do estado de são paulo.
- Atsalakis, G. S. and Valavanis, K. P. (2009). Surveying stock market forecasting techniques - soft computing methods. *Expert Systems with Applications*, 36(3):5932 – 5941.
- Baker, W. E. (1990). Market networks and corporate behavior. *American journal of sociology*, 96(3):589–625.
- Boginski, V., Butenko, S., and Pardalos, P. M. (2005). Statistical analysis of financial networks. *Computational Statistics & Data Analysis*, 48(2):431 – 443.
- Bossomaier, T., Barnett, L., Harré, M., and Lizier, J. T. (2016). *An introduction to transfer entropy: information flow in complex systems*.
- Brin, S. and Page, L. (1998). The anatomy of a large-scale hypertextual web search engine. *Computer Networks and ISDN Systems*, 30(1):107 – 117.
- Chakrabarti, S., Dom, B., Raghavan, P., Rajagopalan, S., Gibson, D., and Kleinberg, J. (1998). Automatic resource compilation by analyzing hyperlink structure and associated text. 30(1):65–74.
- He, J. and Shang, P. (2017). Comparison of transfer entropy methods for financial time series. *Physica A: Statistical Mechanics and its Applications*, 482:772–785.
- Kauê Dal’Maso Peron, T., da Fontoura Costa, L., and Rodrigues, F. A. (2012). The structure and resilience of financial market networks. *Chaos: An Interdisciplinary Journal of Nonlinear Science*, 22(1):013117.
- Kleinberg, J. M. (1999). Authoritative sources in a hyperlinked environment. *Journal of the ACM (JACM)*, 46(5):604–632.
- Korbel, J., Jiang, X., and Zheng, B. (2017). Transfer entropy between communities in complex networks. *ArXiv e-prints*.
- Kwon, O. and Yang, J.-S. (2008). Information flow between stock indices. *Europhysics Letters*, 82(6):68003–p1–68003–p4.
- Leahy, S. P., Levy Carciente, S., Stanley, H. E., and Kenett, D. Y. (2014). Structure and dynamics of the brazilian stock market: A correlation analysis.
- Lee, G. S. and Djauhari, M. A. (2012). An overall centrality measure: The case of us stock market. *International Journal of Electrical & Computer Sciences*, 12(6).
- Li, L., Shang, Y., and Zhang, W. (2002). Improvement of hits-based algorithms on web documents. In *Proc. Int. Conf. on World Wide Web*, pages 527–535.

- Lyócsa, Š., Vÿrost, T., and Baumöhl, E. (2012). Stock market networks: the dynamic conditional correlation approach. *Physica A: Statistical Mechanics and its Applications*, 391(16):4147–4158.
- Marschinski, R. and Kantz, H. (2002). Analysing the information flow between financial times series. *The European Physical Journal B*, 30:275–281.
- Namaki, A., Shirazi, A., Raei, R., and Jafari, G. (2011). Network analysis of a financial market based on genuine correlation and threshold method. *Physica A: Statistical Mechanics and its Applications*, 390(21):3835–3841.
- Page, L., Brin, S., Motwani, R., and Winograd, T. (1999). The pagerank citation ranking: bringing order to the web. Technical Report 1999-66.
- Petelin, D., Šindeläf, J., Přikryl, J., and Kocijan, J. (2011). Financial modeling using gaussian process models. *Proceedings of the 6th IEEE International Conference on Intelligent Data Acquisition and Advanced Computing Systems*, 2:672–677.
- Schreiber, T. (2000). Measuring information transfer. *Physical Review Letters*, 85:461–464.
- Shannon, C. E. (1948). A mathematical theory of communication. *Bell System Technical Journal*, 27:379–423.
- Shirokikh, O., Pastukhov, G., Boginski, V., and Butenko, S. (2013). Computational study of the us stock market evolution: a rank correlation-based network model. *Computational Management Science*, 10(2-3):81–103.
- Tabak, B. M., Serra, T. R., and Cajueiro, D. O. (2010). Topological properties of stock market networks: The case of brazil. *Physica A: Statistical Mechanics and its Applications*, 389(16):3240–3249.
- Xing, W. and Ghorbani, A. (2004). Weighted pagerank algorithm. In *Communication Networks and Services Research, 2004. Proceedings. Second Annual Conference on*, pages 305–314. IEEE.
- Yang, P., Shang, P., and Lin, A. (2017). Financial time series analysis based on effective phase transfer entropy. *Physica A: Statistical Mechanics and its Applications*, 468:398–408.
- Zhang, X., Yu, H., Zhang, C., and Liu, X. (2007). An improved weighted hits algorithm based on similarity and popularity. In *Second International Multi-Symposiums on Computer and Computational Sciences (IMSCCS 2007)*, pages 477–480.

Caracterização do Uso de uma Rede Sem Fio de Grande Porte Distribuída por uma Ampla Área

Luiz Claudio Schara Magalhães¹ e Diogo Menezes Ferrazani Mattos¹

¹Mídiacom – Departamento de Engenharia de Telecomunicações
Universidade Federal Fluminense (UFF)
Niterói, RJ – Brasil

Resumo. Um dos principais desafios no gerenciamento de redes sem fio é assegurar um limiar de desempenho mesmo em situações de disputa pelo uso compartilhado dos canais. Este artigo apresenta a análise da uma rede sem fio institucional usada para produção e pesquisa. A rede sem fio da Universidade Federal Fluminense usa tecnologia desenvolvida pelo grupo de trabalho SCIFI da RNP, que é composta de um controlador de software e um firmware que executa em pontos de acesso de baixo custo compatíveis com a distribuição OpenWRT. A análise da rede sem fio é realizada através da modelagem da rede como um grafo direcionado, em que nós são os pontos de acesso e arestas são as relações de vizinhança no meio sem fio. É feita uma comparação entre algoritmos para a escolha de canal de forma a melhorar o desempenho da rede sem fio, minimizando a interferência entre pontos de acesso existentes. São mostradas as características das associações dos dispositivos dos usuários na rede. Os resultados das análises mostram que, após uma troca de canal, até 75% das redes não gerenciadas na vizinhança de um ponto de acesso podem trocar seu canal para se adaptar à nova configuração global das redes sem fio.

Abstract. Managing wireless networks is challenging due to the requirement of ensuring good performance while coexisting with other networks. This paper analyses a production wireless network that is also used for research. The wireless network at Universidade Federal Fluminense uses home grown technology which was funded by RNP in the SCIFI work group, which is composed of a software controller and low cost access points compatible with OpenWRT. The analysis of the wireless network is performed modeling the network as a directed graph, in which nodes are the access points and edges represent vicinity. Algorithms for choosing the wireless channel in order to maximize performance by minimizing interference both by access points which are controlled or not by the system. The characteristics of associations of user devices to the wireless network are shown. The results of the analysis show that, after the installation of new channel, up to 75% of unmanaged networks in the vicinity of an access point switch their channel to adapt to the new wireless configuration.

1. Introdução

O número de dispositivos conectados à Internet é crescente, impulsionado pela popularização de celulares inteligentes (*smartphones*) [Lin et al., 2017] e pela

Este trabalho foi realizado com recursos da CNPq, CAPES, FAPERJ e RNP.

diversidade de dispositivos conectados à Internet das Coisas (*Internet of Things* - IoT) [Mattos et al., 2018]. Estima-se que até o final de 2020, o número de objetos conectados à Internet ultrapasse 50 bilhões [Yaqoob et al., 2017]. Embora muitos objetos conectados tenham diversas tecnologias de acesso à rede diferentes [Al-Fuqaha et al., 2015], a tecnologia mais popular continua a ser a das redes sem fio [Li et al., 2011], padrão IEEE 802.11 [IEEE, 2016], devido ao baixo custo dos equipamentos e à facilidade de uso. No entanto, com o aumento do número de redes 802.11, o espectro das redes sem fio tem se tornado altamente disputado e compartilhado.

Um dos principais desafios para se prover um desempenho aceitável para usuários de redes sem fio é a sobreposição de canais [Luiz et al., 2015]. Canais próximos geram interferências e diminuem o desempenho das redes sem fio em suas vizinhanças. Portanto, é necessário que o projeto de instalação de redes sem fio considere a alocação de canais tanto do ponto de acesso instalado, como dos demais pontos de acesso já presentes na área. A alocação de canais ótima em redes sem fio é um problema NP-Difícil [Leung e Kim, 2003], mas existem aproximações que levam a boas escolhas de canal [Maturi et al., 2017]. Em redes sem fio de grande porte, alternativas proprietárias para o gerenciamento de pontos de acesso, e consequente atribuição de canais, têm bom desempenho com o compromisso de um alto custo financeiro [Balbi et al., 2012].

Este artigo foca na análise de uma rede sem fio de grande porte gerenciada por uma tecnologia de código aberto, desenvolvida no contexto do grupo de trabalho SCIFI da RNP (Rede Nacional de Ensino e Pesquisa). A tecnologia visa uso de *software* livre e pontos de acesso de baixo custo. O artigo desenvolve uma metodologia de caracterização do comportamento da rede através da análise do grafo de conectividade dos pontos de acesso no meio sem fio. A rede analisada é a rede sem fio universitária que cobre os *campi* da Universidade Federal Fluminense, na cidade de Niterói, Rio de Janeiro.

Por usar tecnologia desenvolvida na própria Universidade, a rede sem fio da Universidade Federal Fluminense (UFF) funciona também como um enorme laboratório, e permite a obtenção de dados que seriam difíceis de gerar em uma simulação ou mesmo coletar numa instalação de mesmo tamanho num laboratório. Neste artigo, é feita a comparação do algoritmo centralizado no controlador que faz a alocação de canais usados pelos pontos de acesso, com outros algoritmos, um iterativo com conhecimento global que muda os canais a cada rodada, um com conhecimento apenas local e outro com escolha aleatória de canais. Recentemente, o algoritmo centralizado no controlador foi modificado para considerar primeiro a interferência interna, entre os pontos de acesso controlados, e depois a interferência externa, dos pontos de acesso com todos os pontos de acesso nas vizinhanças. Anteriormente o cálculo era feito num único passo, através de um algoritmo de coloração de grafos como pode ser visto em [Balbi et al., 2012]. No trabalho anterior era considerada somente a interferência total e os pontos de acesso externos (não-controlados) já entravam no grafo coloridos, isto é, seus canais já eram pré-definidos.

O método utilizado para a determinação da interferência é a coleta de informação dos *beacons* recebidos que carregam apenas a interferência potencial, mas não a interferência real. A interferência real é causada pelo uso intenso da rede. Em uma área, onde existem 37 pontos de acesso controlados e 82 pontos de acesso não controlados, o algoritmo anterior considera o fator externo como preponderante, apesar do número de usuários dos pontos de acesso controlados ser muito maior e, por conseguinte, a inter-

ferência interna também é muito maior. Isto gera uma distorção, que pode ser corrigida considerando o caso em que o tráfego dos pontos de acesso controlados é muito maior, fazendo a minimização da interferência interna, entre pontos de acesso controlados, antes de se considerar a externa.

O novo algoritmo então funciona em dois passos. No primeiro faz-se a coloração de apenas os pontos de acesso controlados, minimizando a interferência entre eles, mas não se escolhe o canal que será alocado a cada cor. O canal é escolhido usando a interferência externa, de forma a minimizá-la. Ao se fazer isto, notou-se que como hoje em dia uma grande parte dos pontos de acesso usa a escolha automática de canais, existe uma tendência de os pontos de acesso externos migrarem para canais que minimizam a interferência, inclusive com os pontos de acesso controlados. Este fato não só melhora a interferência com os pontos de acesso externos como justifica, *a posteriori*, o uso de dois passos, já que a mudança de canal dos pontos de acesso não controlados que antes eram considerados "imutáveis" ocorre como efeito colateral das mudanças de canal dos pontos de acesso controlados.

O restante do artigo está organizado da seguinte forma. A Seção 2 discute os trabalhos relacionados. A infraestrutura da rede sem fio é apresentada na Seção 3. A coleta de dados é descrita na Seção 4. A Seção 5 analisa os resultados obtidos. A Seção 6 conclui o artigo.

2. Trabalhos Relacionados

A análise de desempenho de redes sem fio é um tema constante de pesquisa, devido à popularidade que as redes baseadas no padrão IEEE 802.11 têm assumido [Balbi et al., 2016, Balbi et al., 2012]. Mais recentemente, as redes IEEE 802.11 têm sido apontadas como uma das principais tecnologias habilitadoras da conexão de dispositivos à Internet das Coisas [Mattos et al., 2018, Li et al., 2011].

Henderson *et al.* analisam o perfil de tráfego de uma rede sem fio instalada em um *campus* universitário [Henderson et al., 2008]. Os autores identificam mudanças no perfil do tráfego dos usuários da rede e ressaltam que a ocorrência de associações de usuários visitantes a pontos de acesso (*roaming*) é uma característica marcante que deve ser levada em consideração no projeto da rede. Paralelamente, Balbi *et al.* verificam que em redes altamente densas o número de trocas entre pontos de acesso que os usuários estão associados é muito alto, mesmo quando o usuário não está se movimentando [Balbi et al., 2016].

Wang *et al.* propõem a otimização do uso do espectro das redes sem fio através da cooperação entre pontos de acesso para realizar *beamforming* [Wang et al., 2018]. Os pontos de acesso são conectados via *Ethernet* para tornar mais ágil a troca de informações. A proposta baseia-se em três pilares: um esquema cooperativo que permite o sistema estimar desvios de fase em cada símbolo transmitido e dinamicamente ajustar as fases para garantir o alinhamento; um mecanismo de estimação que mede a qualidade do canal; e um algoritmo aleatório de escolha de usuários para realizar *beamforming* com custo computacional constante. Os autores mostram que o algoritmo aleatório é capaz de escalar a rede linearmente e tem desempenho equivalente a 70% de algoritmos mais complexos.

Shin *et al.* usam a abordagem de descrever a rede sem fio através de um grafo para diminuir a latência durante o *hand-off* [Shin et al., 2004]. A ideia central é desenvolver

algoritmos que permitam que o *hand-off* ocorra sem que a estação tenha a necessidade de ficar monitorando todos os canais. Para tanto, a estação armazenaria o conjunto de canais que cada ponto de acesso vizinho está operando e o conjunto de pontos de acesso vizinhos em cada canal.

Quanto a alocação de canais, em um trabalho anterior, propôs-se a alocação centralizada de canais através de um algoritmo que reduz a interferência entre pontos de acesso dinamicamente, em uma rede sem fio composta por equipamentos de baixo custo [Balbi et al., 2012]. A ideia central é considerar o canal de redes não gerenciadas pelo controlador centralizado e ajustar as frequências dos pontos de acesso gerenciados para evitar a interferência. Maturi *et al.* propõem um esquema de seleção dinâmica de canais que permite que uma rede 802.11 salte sobre os canais disponíveis sempre escolhendo o com menor utilização [Maturi et al., 2017]. Os autores implementam a proposta e mostram a viabilidade de realizar o salto em frequências para redes 802.11 com equipamento padrão de mercado. Outras propostas realizam a alocação de canal através da interferência observada entre pontos de acesso [Lin et al., 2017, Monteiro et al., 2016] e entre clientes e pontos de acesso [Luiz et al., 2015]

A abordagem deste artigo considera a camada de enlace sem fio como sendo um grafo da vizinhança entre os pontos de acesso. Sobre esse grafo calculam-se as métricas para identificar interferências e o comportamento das redes gerenciadas e não gerenciadas por um controlador centralizado. Diferentemente de trabalhos anteriores, a caracterização da rede é realizada sobre a vizinhança dos nós sem fio e não sobre o tráfego da rede. O trabalho aborda ainda uma análise de desempenho de algoritmos de seleção de canais e verifica o resultado da seleção após a consolidação na rede.

3. A Infraestrutura de Redes Sem Fio SciFi

A rede sem fio institucional da Universidade Federal Fluminense (UFF), apesar de ainda não ter atingido seu tamanho projetado de quatro mil pontos de acesso, já cobre boa parte da UFF com mais de quinhentos pontos de acesso e tem picos de uso de mais de 5 mil usuários simultâneos, atendendo uma população total de 60 mil usuários internos entre alunos, docentes e funcionários e visitantes através da rede *eduroam*¹ na UFF. A Figura 1 mostra a variação do número dos usuários no período de um ano, entre março de 2017 e março de 2018. Pode-se notar os meses de férias, bem como os recessos no gráfico, caracterizados pelo baixo número de usuários.

A tecnologia usada na rede foi desenvolvida na própria UFF, com apoio do programa de grupos de trabalho da RNP (Rede Nacional de Ensino e Pesquisa) - o GT SCIFI (Sistema de Controle Inteligente para redes sem fio). O SCIFI criou um controlador de *software* e um *firmware* que pode ser instalado em pontos de acesso de baixo custo. Isto permitiu criar uma rede corporativa a um custo muito mais baixo que uma rede corporativa tradicional. Estima-se que o custo tenha sido cerca de um vigésimo de uma rede tradicional, gerando uma economia para a UFF estimada em mais de dois milhões de reais. A estrutura da rede atual é composta de seis redes sobrecamada (*overlay*) sobre a rede cabeada da UFF. A rede sem fio usa sete VLANs, três que abrangem todo o *campus*. Nestas três últimas trafegam os dados de controle, da rede cadastro e da rede de visitantes. A rede de controle serve para a comunicação entre os pontos de acesso (APs) e o

¹ Informações disponíveis em <https://www.rnp.br/servicos/servicos-avancados/eduroam>.

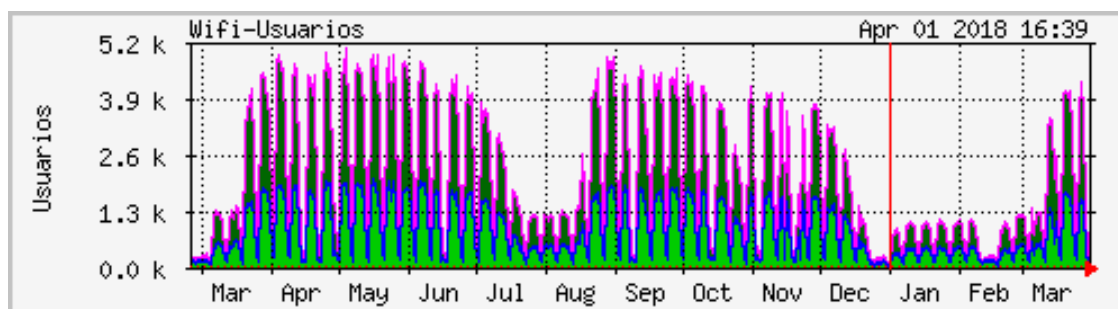


Figura 1. Número de usuários simultâneos da rede sem fio que se estende pelos *campi* da Universidade Federal Fluminense, na cidade de Niterói, RJ. O gráfico mostra a variação do número de usuários no período entre março de 2017 e março de 2018.

controlador, bem como o tráfego das informações de autenticação dos usuários entre os APs e os servidores Radius/LDAP. A rede de cadastro é uma rede *sand-boxed*, que não dá acesso à Internet, mas permite aos usuários acessar os manuais de como usar a rede EDUROAM², o aplicativo de configuração automática, para Android, e para os que não possuem credenciais na UFF, sistema iduff, é apresentada uma página de cadastro. Esta, ao ser preenchida, gera uma conta na rede visitantes. O nome de usuário e a senha na rede visitantes é enviado por SMS para o número de telefone que foi cadastrado. A rede visitantes dá aos usuários cadastrados acesso à Internet com segurança WPA2 *enterprise*, através de usuário e senha únicos por usuário.

As outras quatro VLANs cobrem grandes áreas da UFF. A UFF é distribuída em mais de 90 prédios em Niterói, em mais de 16 *campi* distintos. Atualmente, o número de APs e de usuários justifica duas VLANs específicas, uma para o *campus* da Engenharia (Praia Vermelha) e outra para o de Ciências Humanas (Gragoatá). Os outros *campi* e a reitoria são distribuídos em duas outras VLANs. Cada VLAN tem seu próprio servidor DNS e realiza o seu próprio NAT para o acesso à Internet. Dentro de uma área coberta por uma VLAN, o usuário pode se movimentar mantendo seu IP mesmo quando muda seu ponto de acesso. Como existe uma separação espacial entre áreas, a conexão naturalmente cai quando o usuário deixa um *campus*. Se o usuário migrar para outro *campus* coberto por APs da rede SCIFI de outra área, o dispositivo do usuário recebe um novo IP referente àquela área.

Vale ressaltar que por ser uma rede com propósito múltiplo, produção e pesquisa, o monitoramento da rede é de extrema importância. Os dados gerados pelo monitoramento da rede, em especial por coleta de dados SNMP (*Simple Network Management Protocol*), são armazenados para análise posterior. Os dados coletados mostram o uso de cada ponto de acesso, o número de usuários na rede, informações de consumo de recursos e, também, as associações de usuários a pontos de acesso. No entanto, o monitoramento da rede não considera o uso dos canais de cada ponto de acesso ou de pontos de acesso não gerenciados em uma mesma área.

²É um serviço de acesso a redes sem fio com conexão à Internet, desenvolvido para a comunidade internacional de educação e pesquisa, que se vale da federação de identidade para prover uma forma simples, rápida e segura de acesso à Internet. A autenticação dos usuários ocorre na instituição de origem.

4. A Coleta de Dados das Redes Sem Fio

A infraestrutura de redes sem fio SCIFI analisada cobre diversas áreas da Universidade Federal Fluminense (UFF) na cidade de Niterói, Rio de Janeiro. Os *campi* da UFF são distribuídos em diversos bairros de Niterói e, portanto, geram áreas de cobertura disjuntas da rede sem fio. A metodologia de caracterização do acesso à rede sem fio e divisão do uso de canais consiste em analisar os dados coletados pelos pontos de acesso instalados nas diversas localidades. O procedimento adotado verifica as redes sem fio conhecidas por cada ponto de acesso, averiguando se na vizinhança de cada ponto de acesso é possível identificar possíveis outros pontos de acesso SciFi vizinhos na camada de enlace sem fio. Ademais, o procedimento de verificação de vizinhança ainda armazena quais são as outras redes sem fio na vizinhança de cada ponto de acesso. Tal procedimento permite realizar uma marcação de um estado global do compartilhamento espectro das redes sem fio, tanto para a rede SciFi monitorada quanto para as demais redes. Como a rede SciFi opera prioritariamente nos canais da faixa 2.4 GHz, o procedimento de marcação do estado global do compartilhamento dos canais foi realizado somente para a faixa de frequências de 2.4 GHz. A coleta de dados é composta por três etapas: a varredura de cada ponto de acesso instalado na rede, a concentração e a interpretação dos dados coletados e, por fim, a representação dos dados coletados na forma de um grafo direcionado.

A primeira etapa consiste na realização de uma varredura das redes sem fio que está no alcance de cada ponto de acesso. Para tanto, usa-se a ferramenta *iw scan*³. Como a estrutura da rede SciFi conta com um controlador de rede, a execução da varredura é orquestrada pelo controlador. Para evitar que algum possível usuário perca acesso à rede durante a varredura, a varredura é realizada através da criação de uma interface de rede sem fio virtual de acesso somente para a realização desse procedimento. Os usuários conectados no ponto de acesso que realiza a varredura continuam conectados na interface de rede sem fio inicialmente dedicada a ser o ponto de acesso através do software *hostapd*⁴. O controlador acessa cada um dos nós, pontos de acesso, da rede e executa a varredura. Por se tratar de uma rede em um ambiente de produção, a varredura de todos os pontos de acesso foi agendada para ser executada fora do período de pico de uso da rede. Em especial, a varredura foi executada durante a madrugada, quando os prédios dos *campi* da UFF estão vazios e, portanto, com menos obstáculos para a propagação dos sinais das redes sem fio. O controlador acessa cada ponto de acesso através de uma chamada por SSH (*Secure Shell*). O resultado da varredura nó a nó é então transferida para o controlador da rede. A varredura dos nós é repetida 5 vezes em turnos. A cada turno, todos os nós são varridos em uma ordem definida *a priori*. Vale ressaltar que a varredura em cada nó evidencia a interferência potencial que cada rede pode gerar, mas não a interferência real. A interferência potencial é medida através da qualidade do sinal informada no *beacon* de cada ponto de acesso identificado na varredura. Já a interferência real depende também do uso da rede. Assim, uma rede próxima com poucos usuários e uma alta relação sinal ruído pode ter uma interferência real mais baixa que uma rede mais distante, baixa relação sinal-ruído, porém com mais usuários usando intensamente a interface de rede sem fio e, conseqüentemente, gerando mais interferências.

A segunda etapa da coleta dos dados consiste na concentração e interpretação dos

³Disponível em <http://drvbp1.linux-foundation.org/mcgrof/rel-html/iw/>.

⁴Disponível em <https://w1.fi/hostapd/>.

Algoritmo 1: Representação da varredura de redes sem fio em um grafo direcionado.

Entrada: Scan; %Dicionário contendo os dados da varredura
 APs; %Lista de APs monitorados
Saída: Grafo

início

```

  Grafo = novo_grafo_vazio();
  for turno ∈ Scan do
    for ap ∈ Scan do
      if ap ∉ Grafo then
        adiciona_nó(ap, Grafo);
        Grafo.nó[ap].scan = Dicionário();
        for mac ∈ Scan[ap] do
          qualidade = Scan[ap][mac].qualidade();
          if mac ∉ Grafo.node[ap].scan OU
            Grafo.node[ap].scan[mac].qualidade > qualidade then
            | Grafo.node[ap].scan.adiciona(mac : qualidade);
          if mac ∈ APs then
            | adiciona_aresta(ap → mac, Grafo);
          end
        end
      end
    end
  end

```

fim

dados coletados. Os dados coletados em cada ponto de acesso são salvos no controlador de rede e, então, interpretados. Como todos os dados estão locais, torna-se possível a rápida correlação das informações. Assim, os dados da varredura são interpretados através um aplicativo desenvolvido em Python que consolida todos os dados em um grafo direcionado. Para tanto, utiliza-se a biblioteca NetworkX⁵ para a criação e a operação do grafo de conexão entre os pontos de acesso. Ao descrever o relacionamento dos pontos de acesso como um grafo, o problema de caracterização da rede sem fio torna-se um problema de redes complexas. Para realizar a terceira etapa da coleta de dados, a consolidação em um grafo, aplica-se o Algoritmo 1, em que a entrada é o conjunto de dados coletados e interpretados nas etapas anteriores. A rede é modelada como um grafo $G(N, A)$, em que N representa o conjunto dos nós, no caso os pontos de acesso da rede sem fio, e A representa o conjunto das arestas. Uma aresta $a_{i,j} \in A$ representa a relação de que o ponto de acesso i é capaz de receber o sinal e identificar o ponto de acesso j . Somente as relações entre dois pontos de acesso controlados pelo SciFi são considerados na formação do grafo $G(N, A)$.

A estrutura de dados utilizada para representar o grafo da rede armazena em cada nó a lista de redes obtidas na varredura realizada pelo nó. A variável *qualidade* armazena o valor da qualidade do sinal da rede retornada pelo *iw_scan*, que é uma inferência da qualidade de recepção do sinal da rede. Ao verificar se a qualidade da rede monitora é maior que uma qualidade já armazenada anteriormente, o algoritmo de construção

⁵Disponível em <https://networkx.github.io/>.

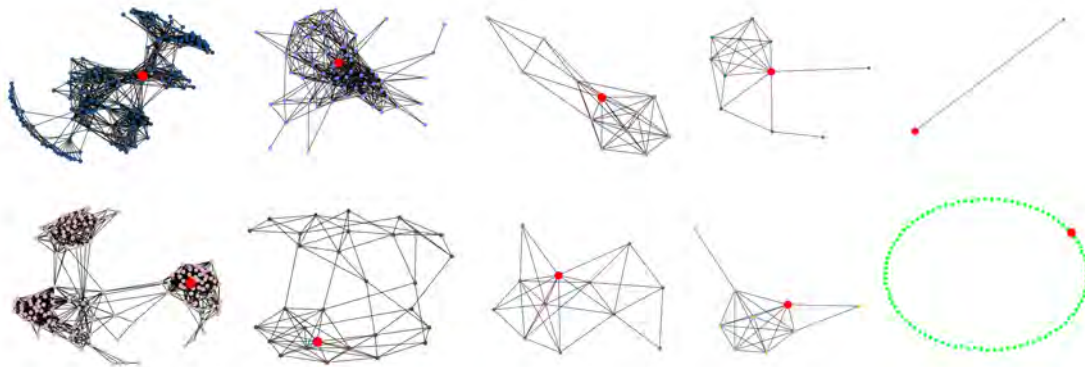


Figura 2. Grafo de conexão da rede sem fio monitorada visualizado em relação a suas componentes conexas. As componentes conexas correspondem fisicamente aos *campi* onde estão instalados os pontos de acesso. A maior componente conta com 160 nós e 92 componentes contam com apenas um nó em cada. Os nós vermelhos são os pontos de acesso com mais vizinhos dentro de uma componente.

do grafo considera apenas o melhor sinal recebido de cada rede em todos os turnos da varredura.

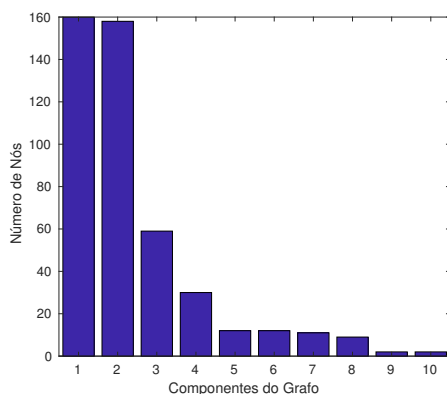
Após a coleta e o tratamento dos dados, é possível realizar uma visualização preliminar dos dados, visualizando as componentes conexas do grafo. No caso do grafo de vizinhança das redes sem fio, as componentes conexas são as partições da rede que estão isoladas na camada de enlace. A Figura 2 mostra a visualização da rede sem fio SciFi da UFF através da representação das suas componentes conexas. O grafo é composto por 547 nós, representando todos os pontos de acesso monitorados. Percebe-se que há 9 componentes conexas maiores, que são compostas por 454 pontos de acesso, e 93 pontos de acesso que não pertencem a nenhuma componente conexa. As 9 maiores componentes conexas representam os pontos de acesso distribuídos nos *campi* da UFF. Cada componente é um *campus*, ou prédio administrativo, que tem pontos de acesso da rede SciFi. Dentre os 93 pontos de acesso isolados estão pontos de acesso defeituosos que não estão com a interface sem fio ativada e, também, pontos de acesso em locais isolados nos quais não há sobreposição de cobertura da rede com outros pontos de acesso SciFi.

Vale ressaltar que a estrutura de dados utilizada para representar o grafo da rede armazena em cada nó a lista de redes obtidas na varredura realizada pelo nó. A variável *qualidade* armazena o valor da qualidade do sinal da rede retornada pelo `iw scan`, que é uma inferência da qualidade de recepção do sinal da rede. Ao verificar se a qualidade da rede monitorada é maior que uma qualidade já armazenada anteriormente, o algoritmo de construção do grafo considera apenas o melhor sinal recebido de cada rede em todos os turnos da varredura.

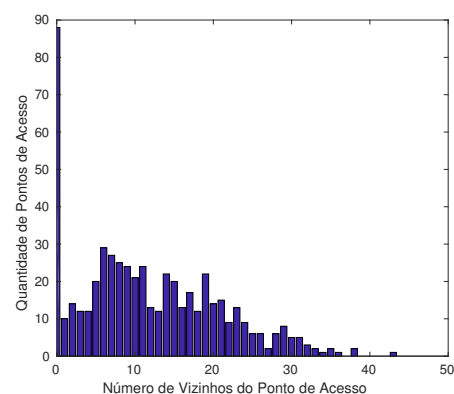
5. A Análise dos Dados

A primeira análise do grafo obtido consiste na avaliação das componentes do grafo e grau dos nós. O objetivo desse primeiro experimento é identificar as componentes do grafo e mensurar o tamanho delas. A Figura 3(a) mostra que há duas componentes maiores na rede sem fio, compostas por 160 nós cada. Essas componentes representam os dois maiores *campi* da UFF. Esse resultado mostra que os pontos de acesso dentro do mesmo *campus* formam uma rede conexa e, portanto, são capazes de fornecer cobertura

de rede sem fio na maior parte do *campus*. Já a Figura 3(b) mostra que o número de vizinhos que cada ponto de acesso tem na rede é bastante variado. Pontos de acesso sem vizinhos (zero vizinhos) são numerosos, pois nessa contagem consideram-se os pontos de acesso defeituosos e os instalados em áreas remotas. Os demais pontos de acesso têm até 50 vizinhos. Uma consequência do fato de o número de vizinhos ser alto, na média, é que os clientes que se associam a um ponto de acesso estão constantemente na visada de outros pontos de acesso e, portanto, ficam sujeitos a realizar constantemente a troca de associação entre diferentes pontos de acesso.



(a) Nós por componente conexa do grafo.



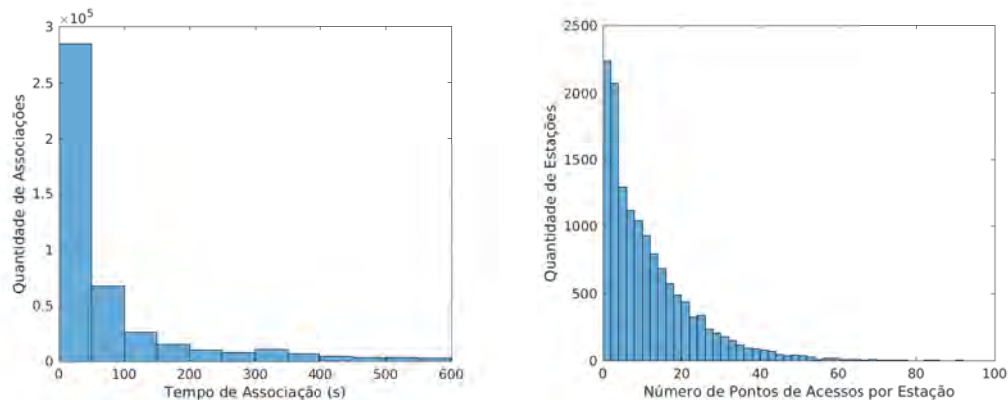
(b) Número de nós que um ponto de acesso percebe na sua vizinhança.

Figura 3. Caracterização do grafo de vizinhança entre os pontos de acesso que compõem a rede sem fio monitorada. Grafo consolidado a partir do monitoramento das redes sem fio em cada nó. a) Comparação do tamanho das 10 maiores componentes conexas formadas na rede. Cada uma das demais componentes é formada por apenas um ponto de acesso. b) Distribuição do grau de vizinhança dos nós na rede sem fio.

Para realizar análise dos dados, além de contar com as varreduras das redes sem fio em cada ponto de acesso, foram coletados também os registros de associações de clientes a cada ponto de acesso da rede. Foram considerados os registros de associação de todos os nós da rede de um único dia⁶. Nesse dia, foram identificados 13.806 dispositivos distintos que acessaram a rede sem fio. Assim, também é possível verificar o perfil de uso da rede por parte dos clientes, assim como detectar possíveis pontos de sobrecarga de acesso na rede. A Figura 4 mostra o comportamento dos clientes na rede. Pela Figura 4(a), verifica-se que o tempo de associação dos clientes aos pontos de acesso é predominantemente curto, da ordem de menos de 1 min. Esse comportamento deve-se ao fato de a rede sem fio analisada apresentar áreas de cobertura muito densas e, assim, um cliente troca constantemente o ponto de acesso a que está conectado. Como todos os pontos de acesso estão na mesma sub-rede IP, a constante reassociação de um cliente a outros pontos de acesso não gera quebra das conexões, nem exige a reconfiguração da interface de rede. Contudo, a reassociação exige a reconfiguração de tabelas de encaminhamento dos computadores da rede, o que pode gerar um grande número de pacotes ARP sendo enviados em difusão na sub-rede e atrasos na reconfiguração das tabelas de encaminhamento dos computadores. As múltiplas associações de um cliente a diversos pontos de acesso por

⁶O dia usado para o teste foi 27 de março de 2018.

estação fica evidente no histograma mostrado na Figura 4, na qual se nota que há clientes que se associam a mais de 100 pontos de acesso em um único dia. Paralelamente, o comportamento mais comum é de clientes que se associam a somente um ou a um grupo reduzido, menor que 10 pontos de acesso.



(a) Tempo de associação de estação por ponto de acesso. (b) Número de pontos de acesso que uma estação se associa por dia.

Figura 4. Comportamento da associação de estações a pontos de acesso. a) Distribuição do tempo e associação de uma estação a pontos de acesso. b) Distribuição do número de pontos de acesso únicos que uma estação se associa por dia.

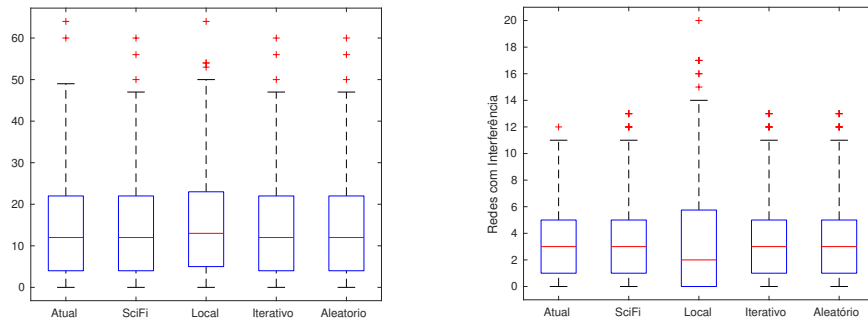
A experiência de acesso à rede sem fio pelos clientes depende da escolha de canais em que cada ponto de acesso serve a rede sem fio. Contudo, a escolha do canal de cada ponto de acesso é dependente tanto dos canais dos nós vizinhos, como das demais redes sem fio não monitoradas pelo controlador de rede SciFi. Um desafio na escolha dos canais de cada ponto de acesso é que há apenas 3 dos 11 canais⁷ da faixa de frequência de 2.4 GHz que não se interferem mutuamente. Assim, o problema de definição dos canais na rede sem fio é NP-difícil, pois pode ser reduzido a um problema de coloração de grafos. O problema é agravado pelo fato de que os nós no grafo apresentam graus altos, enquanto o número de cores disponíveis é de apenas 3. Assim, a Figura 5 mostra a distribuição do número de redes que compartilham canal com pontos de acesso SciFi, ao se considerar quatro algoritmos distintos de seleção de canal. A configuração da rede sem fio antes da aplicação de qualquer algoritmo é mostrada como *Atual*. O algoritmo de seleção de canal do controlador SciFi é mostrado como *SciFi*. Um algoritmo guloso iterativo (*Iterativo*), um algoritmo de escolha de canal usando somente conhecimento local (*Local*) e a escolha aleatória de canais (*Aleatório*) são propostas como critério de comparação.

O algoritmo iterativo consiste em executar iterações consecutivas nos nós da rede para minimizar localmente a interferência em cada nó, usando o conhecimento global das frequências. As iterações param quando o número de nós que trocam de canal em uma dada iteração é menor que 10% da rede. O algoritmo com conhecimento local realiza a troca do canal do ponto de acesso calculando qual é o canal, dentre os canais que não

⁷Canais 1, 6 e 11 não ortogonais. Os demais canais geram interferência em ao menos um desses 3 canais.

se interferem, que está menos poluído por outras redes. O algoritmo com conhecimento local simula o funcionamento de pontos de acesso que configuram automaticamente o seu canal. Por fim, a escolha aleatória consiste em selecionar uniformemente um canal entre os que não se interferem para cada ponto de acesso.

A Figura 5(a) mostra que os algoritmos com conhecimento global da rede, como o SciFi e o Iterativo, minimizam o número de redes não controladas que compartilham canal com as rede SciFi. Contudo, verifica-se também que a escolha aleatória de canal também apresenta uma boa escolha de canal que, como uma abordagem global, minimiza o número de interferência com outras redes. O comportamento da escolha de canais também se repete ao se considerar somente os nós controlados pelo SciFi, mostrado na Figura 5(b). Nota-se que o algoritmo com escolha local minimiza a mediana da interferência entre nós SciFi, mas aumenta a variância e média da interferência global. Os melhores resultados, mais uma vez, são dos algoritmos com conhecimento global da rede. Contudo, nota-se também que a solução com menor interferência é a configuração atual da rede sem a aplicação de qualquer algoritmo. Tal resultado levanta a hipótese que as demais redes sem fio, não controladas pelo SciFi, se adaptam às mudanças de canal dos pontos de acesso SciFi.



(a) Pontos de acesso não monitorados que compartilham o mesmo canal.

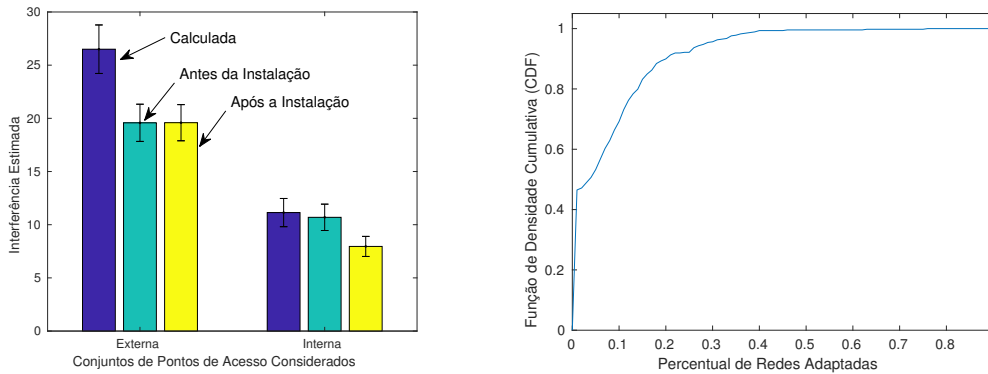
(b) Pontos de acesso monitorados e controlados que compartilham o mesmo canal.

Figura 5. Comparação entre quatro algoritmos de escolha de canal para os pontos de acesso monitorados, o algoritmo do SciFi, uma busca local do canal menos poluído, um algoritmo guloso com visão global da rede que realiza interações para diminuir a interferência interna e externa na rede e a seleção aleatória de canais. a) Número de pontos de acesso não monitorados que interferem no canal; b) Pontos de acesso monitorados que compartilham o canal.

Para verificar se a hipótese de que as redes sem fio adaptam-se às mudanças de canal dos nós SciFi, define-se uma métrica de interferência para calcular a interferência global da rede antes da instalação de novos canais, a interferência calculada para a configuração selecionada e, por fim, a interferência calculada após a nova configuração de canal ser instalada e consolidada na rede. A interferência de quanto uma rede i gera interferência ($interf_{i,j}$) em um ponto de acesso j foi modelada como sendo uma função da qualidade, normalizada entre 0 e 1, percebida pelo ponto de acesso em relação as outras redes:

$$interf_{i,j} = (1 - 0,2 * |canal_i - canal_j|) \frac{qualidade_{i,j}}{(1 - qualidade_{i,j})^3}, \quad (1)$$

para toda rede i tal que $|canal_i - canal_j| \leq 4$, em que $canal_i$ é o canal da rede i e $canal_j$, do ponto de acesso j . A Equação 1 não calcula a interferência real entre as redes, mas auxilia a mensurar o quanto os canais na faixa de 2.4 GHz estão usados. Ao usar o termo $(1 - qualidade_{i,j})^3$ pondera-se que quanto menor a qualidade do sinal de uma rede, menor é a sua interferência no ponto de acesso. Vale ressaltar que a medida da $(1 - qualidade_{i,j})$ pode ser interpretada como função proporcional à distância entre os pontos de acesso das redes i e j . Embora a métrica da Equação 1 permita inferir a interferência entre as redes, essa métrica não é capaz de considerar o uso real da rede e, portanto, a inferência é realizada somente sobre a interferência potencial.



(a) Interferência estimada em função da qualidade do sinal medida nos nós.

(b) Função de densidade cumulativa do percentual de redes sem fio não controladas que mudam de canal.

Figura 6. A interferência estimada externa foi maior que a interferência estimada após a instalação de novos canais. Os pontos de acesso não controlados mudam o canal para se adequarem aos novos canais da rede sem fio controlada. a) Comparação dos valores de interferência estimados antes e depois da instalação de novos canais. b) Função de densidade cumulativa das redes não controladas que mudam de canal para se adaptar à rede sem fio controlada.

A Figura 6(a) apresenta as interferências médias inferidas através da Equação 1, com um intervalo de confiança de 95%. Os resultados mostram que ao instalar novos canais calculados com o algoritmo do SciFi, a interferência externa que os pontos de acesso sofrem aumenta em relação à interferência calculada para a configuração da rede antes da instalação dos novos canais. O algoritmo de instalação de novos canais do SciFi reduz ligeiramente a interferência interna calculada. Contudo, ao instalar a nova configuração de canais, as demais redes sem fio se adaptam aos novos canais e mudam o seu próprio canal. Esse comportamento é verificado ao realizar uma nova varredura das redes sem fio. Na nova varredura, a interferência média das redes externas ao SciFi volta a atingir o mesmo patamar anterior e, como o algoritmo de controle do SciFi considera a potência de cada ponto de acesso no canal, a nova configuração de canais reduz ainda a interferência interna entre os nós SciFi.

A probabilidade de pontos de acesso externos ao SciFi mudarem o canal em resposta a uma mudança de canal da rede SciFi é mostrada na Figura 6(b). Verifica-se que até 75% das outras redes sem fio mudam de canal em resposta a uma mudança da configuração do SciFi. Há uma probabilidade de 0.4 de que no mínimo 10% das outras redes mudem seus canais em resposta à mudança na rede SciFi.

6. Conclusão

O número de dispositivos conectados à Internet é cada vez maior e a principal tecnologia de acesso são as redes sem fio do padrão 802.11. Com a popularização dessas redes, o número de pontos de acesso em áreas urbanas tornou-se vertiginosamente alto. Assim, uma das tarefas mais desafiadoras no gerenciamento de redes sem fio é atribuição de canais a pontos de acesso de modo que minimize tanto a interferência entre pontos de acesso gerenciados quanto a interferência com pontos de acesso não gerenciados. Contudo, esse é um problema sem uma solução exata. Esse artigo apresentou a caracterização de uma rede sem fio da Universidade Federal Fluminense, uma rede de grande porte implantada e gerenciada com *software* livre e *hardware* de baixo custo. A rede sem fio conta com picos de uso de mais de 5 mil usuários ativos e 547 pontos de acesso gerenciados espalhados por *campi* da universidade. O grafo construído a partir de uma varredura total da rede mostrou que há 9 componentes conexas no grafo, indicando que em cada *campus* em que há pontos de acesso instalado, os pontos de acesso são todos vizinhos. A distância máxima entre pontos de acesso em uma componente é de 8 saltos. A análise dos dados de acesso dos usuários revelou que, em um dia, mais de 12 mil dispositivos se associam a rede, mas que o tempo de associação entre dispositivo e ponto de acesso é muito curto devido a constantes *hand-offs*. A análise de algoritmos para a definição de canais entre os pontos de acesso mostrou que o conhecimento global do canal de cada ponto de acesso leva a heurísticas que têm melhor desempenho, mas a definição aleatória de canais também geram bons resultados. Por fim, os resultados mostraram, também, que ao executar a mudança de canal em pontos de acesso em uma rede, gera-se uma reação em cadeia na área em que até 75% dos pontos de acesso não gerenciados em uma área mudam sua frequência para diminuir a disputa pelo canal na rede sem fio.

Como trabalhos futuros, pretende-se avaliar o perfil de uso das redes por cliente e inferir uma métrica de qualidade de experiência para que se possa medir o quanto os clientes estão satisfeitos com o serviço de rede sem fio.

Agradecimentos

Agradecemos a todos os alunos, funcionários e a todos que direta ou indiretamente trabalham na instalação e manutenção da infraestrutura de rede sem fio SciFi.

Referências

- Al-Fuqaha, A., Guizani, M., Mohammadi, M., Aledhari, M. e Ayyash, M. (2015). Internet of Things: A survey on enabling technologies, protocols, and applications. *IEEE Communications Surveys Tutorials*, 17(4):2347–2376.
- Balbi, H., Fernandes, N., Souza, F., Carrano, R., Albuquerque, C., Muchaluat-Saade, D. e Magalhaes, L. (2012). Centralized channel allocation algorithm for IEEE 802.11 networks. Em *2012 Global Information Infrastructure and Networking Symposium (GIIS)*, p. 1–7.
- Balbi, H., Passos, D., Carrano, R., Magalhaes, L., e Albuquerque, C. (2016). Análise e solução para o problema da instabilidade de associação em redes IEEE 802.11 densas. Em *XXXIV Simpósio Brasileiro de Redes de Computadores e Sistemas Distribuídos - SBRC'2016*.

- Henderson, T., Kotz, D. e Abyzov, I. (2008). The changing usage of a mature campus-wide wireless network. *Computer Networks*, 52(14):2690 – 2712.
- IEEE (2016). IEEE Standard for Information technology–Telecommunications and information exchange between systems Local and metropolitan area networks–Specific requirements - Part 11: Wireless LAN Medium Access Control (MAC) and Physical Layer (PHY) Specifications. *IEEE Std 802.11-2016 (Revision of IEEE Std 802.11-2012)*, p. 1–3534.
- Leung, K. K. e Kim, B. J. (2003). Frequency assignment for IEEE 802.11 wireless networks. Em *2003 IEEE 58th Vehicular Technology Conference. VTC 2003-Fall (IEEE Cat. No.03CH37484)*, volume 3, p. 1422–1426 Vol.3.
- Li, L., Xiaoguang, H., Ke, C. e Ketai, H. (2011). The applications of WiFi-based wireless sensor network in Internet of Things and Smart Grid. Em *2011 6th IEEE Conference on Industrial Electronics and Applications*, p. 789–793.
- Lin, F. Y. S., Wen, Y. F., Fang, L. W. e Hsiao, C. H. (2017). Resource allocation and multisession routing algorithms in coordinated multipoint wireless communication networks. *IEEE Systems Journal*, PP(99):1–12.
- Luiz, T. A., Freitas, A. R. e Guimarães, F. G. (2015). A new perspective on channel allocation in wlan: Considering the total marginal utility of the connections for the users. Em *Proceedings of the 2015 Annual Conference on Genetic and Evolutionary Computation, GECCO '15*, p. 879–886, New York, NY, USA. ACM.
- Mattos, D. M. F., Velloso, P. B. e Duarte, O. C. M. B. (2018). Uma infraestrutura Ágil e efetiva de virtualização de funções de rede para a internet das coisas. Em *XXXVI Simpósio Brasileiro de Redes de Computadores e Sistemas Distribuídos - SBRC'2018*.
- Maturi, F., Gringoli, F. e Cigno, R. L. (2017). A dynamic and autonomous channel selection strategy for interference avoidance in 802.11. Em *2017 13th Annual Conference on Wireless On-demand Network Systems and Services (WONS)*, p. 1–8.
- Monteiro, A., Souto, E., Pazzi, R. e Kiljander, J. (2016). Atribuição dinâmica de canais em redes sem fio não coordenadas ieee 802.11, baseada em fatores de sobreposição e intensidade de sinal. Em *XXXIV Simpósio Brasileiro de Redes de Computadores e Sistemas Distribuídos - SBRC'2016*.
- Shin, M., Mishra, A. e Arbaugh, W. A. (2004). Improving the latency of 802.11 hand-offs using neighbor graphs. Em *Proceedings of the 2Nd International Conference on Mobile Systems, Applications, and Services, MobiSys '04*, p. 70–83, New York, NY, USA. ACM.
- Wang, T., Yang, Q., Tan, K., Zhang, J., Liew, S. C. e Zhang, S. (2018). Dcap: Improving the capacity of wifi networks with distributed cooperative access points. *IEEE Transactions on Mobile Computing*, 17(2):320–333.
- Yaqoob, I., Ahmed, E., Hashem, I. A. T., Ahmed, A. I. A., Gani, A., Imran, M. e Guizani, M. (2017). Internet of things architecture: Recent advances, taxonomy, requirements, and open challenges. *IEEE Wireless Communications*, 24(3):10–16.

Avaliação de Desempenho do IEEE 802.11 para Coleta de Grande Volume de Dados em Florestas com Auxílio de Drones

Caroline Maul de A. Lima¹, Eduardo A. da Silva¹, Pedro B. Velloso¹

¹Grupo de Teleinformática e Automação – GTA/COPPE
Universidade Federal do Rio de Janeiro – UFRJ

Abstract. *Most IoT solutions, in order to reduce power consumption, assume severe constraints on the transmission rate, which limit their use to applications with small amounts of collected data per device. Thus, these solutions are not suitable for monitoring wild animals, which requires transmitting large amounts of photos captured by cameras, installed in the middle of the forest. In addition, the forest environment also impairs the transmission capacity. Thus, this work proposes collecting data from devices in the forest with the aid of drones. Hence, this paper aims at evaluating the performance of a wireless network between the device attached to the drone and the cameras. To achieve our goal, we evaluate traditional network metrics, such as maximum range, transmission rate, and packet loss. To compare and better characterize the transmission in the forest, we also consider different scenarios. The most important result is the feasibility of using low-complexity devices for forest applications.*

Resumo. *A maioria das soluções para IoT, com intuito de reduzir o consumo de energia, assume severas restrições à taxa de comunicação limitando-as para o uso de aplicações onde os sensores captam pequenas quantidades de dados. Por isso, estas soluções não são adequadas para o monitoramento de animais silvestres, que exige a transmissão de grandes quantidades de fotos capturadas por câmeras trap instaladas no meio das florestas. Além disso o ambiente florestal também prejudica a capacidade de transmissão. Portanto, este trabalho propõe a coleta de dados de dispositivos na floresta com o auxílio de drones. Assim, este artigo visa avaliar o desempenho de uma rede sem fio para a comunicação entre o drone e os dispositivos de captura de imagem. Para tal, são avaliadas as principais métricas de rede, tais como alcance máximo, taxa de transmissão e perda de pacotes. A título de comparação e para melhor caracterizar a transmissão na floresta, são realizados testes em outros cenários. O resultado mais importante é a viabilidade do uso desses tipos de dispositivos de baixa capacidade computacional para aplicações florestais.*

1. Introdução

Tradicionalmente, o monitoramento de animais silvestres é realizado através de câmeras que são instaladas no interior das florestas, em lugares previamente definidos, a fim de registrar a presença deste animais através de fotos ou vídeos [Steenweg et al. 2017]. Para tanto, estas câmeras, comumente denominadas *camera trap* (câmera armadilha), são dotadas de um sensor de presença que dispara a câmera assim que detecta a presença de algo. Apesar de amplamente utilizado, este procedimento

de monitoramento implica um elevado custo operacional de instalação das câmeras e sobretudo de coleta dos dados. Em geral, as áreas de monitoramento se localizam em unidades de conservação, em regiões extensas e de difícil acesso. Usualmente, a coleta de dados é realizada de forma puramente manual que exige a presença de pessoas para recuperar cada uma das câmeras e retirar os dados. Além de representar um custo bastante elevado, este método não é nada eficiente, visto que não leva em conta, por exemplo, se a câmera que será visitada tirou alguma foto, se ela apresentou algum defeito, ou mesmo se ainda estava com bateria. Por fim, dependendo do tamanho do parque e da área monitorada, este procedimento pode levar muitos dias. Portanto, este trabalho propõe um sistema automatizado de coleta de dados para o monitoramento de animais silvestres em unidades de conservação.

A ideia consiste em aplicar o conceito de IoT, ou seja, dotar as câmeras com a capacidade de comunicação com dispositivos de baixo custo, de modo a permitir o envio das fotos para um *drone* que sobrevoará a floresta coletando as fotos de cada uma das câmeras. No entanto, tradicionalmente, as aplicações de IoT assumem que os dispositivos de captura de dados coletam pequenas quantidades de informação, como por exemplo, temperatura, pressão, umidade, posição e assim por diante. Assim, estes dispositivos não precisam ter uma alta capacidade de transmissão. Esta premissa vai ao encontro da limitação de consumo de energia destes dispositivos. Para comprovar este fato, muitas das propostas mais recentes para IoT, como Lora¹ e SigFox², preveem a duração da bateria da ordem de anos, com uma taxa de transmissão bastante reduzida. Claramente, este tipo de tecnologia de comunicação não é adequada para aplicações de grande quantidade de dados em florestas.

Existem alguns trabalhos que propõem o uso de *drones* para coleta de pequenas quantidades de dados em uma rede de sensores [Gu et al. 2013, Abdulla et al. 2014, Jeong et al. 2015, Say et al. 2016]. A principal inovação deste trabalho é usar os *drones* para a coleta de grande quantidade de dados, capturados e transmitidos por dispositivos de baixo poder computacional dentro de uma floresta tropical. Por este motivo, existem dois grandes desafios para a coleta das fotos de animais silvestres. O primeiro é a transmissão de uma quantidade significativa de dados, com restrição de consumo de energia. A segunda é a transmissão de dados em um ambiente de floresta densa e fechada onde o desempenho de redes sem fio pode ficar prejudicado. Portanto, o objetivo principal deste artigo é caracterizar a transmissão de dados de dispositivos IEEE 802.11, de baixo custo, poder computacional e consumo de energia, em ambientes de florestas tropicais. Assim, o módulo 802.11 é avaliado na transmissão de dados do interior da floresta para um *drone* situado acima da copa das árvores. Além disso, para efeitos de comparação e melhor entendimento do efeito da floresta fechada no desempenho da comunicação, também é avaliado o desempenho do módulo em cenários internos, externos e dentro da floresta.

Apesar de alguns trabalhos já terem avaliado a transmissão do padrão IEEE 802.11 em florestas, nenhum deles avaliou dispositivos de baixo poder computacional, que tem um impacto significativo no desempenho da comunicação, como mostrado nos resultados deste trabalho. Além disso, a maioria deles realizam os testes em outros tipos de florestas, que possuem características distintas. Portanto, não foi encontrado na literatura uma

¹www.lora-alliance.org

²www.sigfox.com

avaliação da comunicação entre um dispositivo IEEE 802.11 para aplicações de IoT e um *drone*, através da copa das árvores de uma floresta tropical.

O restante deste artigo está organizado da seguinte forma. Na Seção 2, são apresentados os principais trabalhos relacionados à avaliação do desempenho do IEEE 802.11, bem como à coleta de dados por *drones*. A ideia principal do sistema de coleta de dados de animais silvestres é apresentada brevemente na Seção 3. A Seção 4 detalha os experimentos para a avaliação de desempenho do IEEE 802.11. Os principais resultados são apresentados e analisados na Seção 5. Por fim, a Seção 6 conclui o trabalho e apresenta os trabalhos futuros.

2. Trabalhos relacionados

Exaustivos estudos sobre redes sem fio podem ser encontrados na literatura. Seja como em [Petajajarvi et al. 2015] onde é avaliado o desempenho e alcance de uma rede LPWAN (*Low-Power Wide-Area Network*) em cidade, ou em [Kriara et al. 2016] que realiza testes em um ambiente de escritório de uma rede IEEE 802.11ac, avaliando a justiça na transmissão ao serem alteradas as configurações do protocolo, e ainda em [Juang et al. 2008], que avalia a capacidade do enlace de uma rede IEEE 802.11b/g construída em um campo de futebol americano universitário. Diversas tecnologias foram avaliadas, nas mais variadas situações e com enfoques totalmente diferentes.

Há ainda propostas de avaliações, que como nesse artigo, enfatizam o ambiente florestal. Como em [Heng Siong Lim et al. 2012] e [Mutschlechner et al. 2013] onde, no primeiro, os autores investigam o efeito da vegetação sobre uma rede IEEE 802.11n em uma floresta tropical na Malásia. Usando antenas direcionais com ganho de 19 dBi, 3 dispositivos foram dispostos na floresta à 40 m, 108 m e 174 m de distância do ponto de acesso. O objetivo desse estudo era avaliar a viabilidade de construir uma rede para levar o acesso a Internet a áreas rurais da região. No segundo trabalho, os autores avaliaram o desempenho de uma rede IEEE 802.11a/b/g entre sensores, para fins de monitoramento de morcegos. Houve a preocupação de se avaliar em diversos cenários de floresta, a saber, com pouca, nenhuma e densa vegetação. Nesse artigo, a distância de 90 m apresentou 100% de perdas de pacotes devido à densidade da vegetação. Foi concluído, também, que a vegetação tem um efeito muito maior no desempenho da rede do que a distância. Wang *et al.* [Wang et al. 2008] avaliam o desempenho de uma rede *mesh* que envia dados de sensores, que monitoram uma bacia hidrográfica na floresta, a um *campus* universitário, com uma distância total de aproximadamente 8 km, usando as tecnologias IEEE 802.11a/b/g e IEEE 802.15.4 ao longo da rede. A rede é dividida em rede de trânsito - que leva os dados até o *campus* - e rede de sensores na floresta. Já Ding *et al.* [Ding et al. 2016] avaliam uma rede IEEE 802.15.4 em três tipos diferentes de florestas, mostrando o efeito deste ambiente na qualidade dos enlaces.

No âmbito de monitoramento usando dispositivos de Internet das Coisas, como esse artigo, o autor em [Nikhade 2015] propõe uma aplicação de monitoramento de baixo custo que usa *RaspberryPi* como concentrador de dados dos nós sensores, entretanto usando comunicação baseada no padrão IEEE 802.15.4. Em outro sistema, proposto em [Huu-Quoc Nguyen et al. 2015], também é utilizado um *RaspberryPi* para monitoramento, porém ele é associado à uma câmera que transmite em tempo real as imagens do sistema de vigilância para a Internet.

Outro trabalho interessante é a aplicação proposta em [Vasisht et al. 2017], onde é possível ver o uso de veículos não-tripulados para o auxílio na monitoração e predição de dados para cultivo e criação de gado em fazendas. Porém esta aplicação considera pequenas quantidades de dados coletadas pelos sensores. Além disso, o *drone* não é utilizado para coletar os dados dos sensores, mas sim para filmar a fazenda. A coleta de dados é realizada por estações base IoT, proposta no trabalho.

Não obstante os trabalhos citados acima, os autores deste artigo desconhecem o uso de *drones* para coleta de grande volume de dados de monitoramento em floresta utilizando dispositivos IoT - de baixo custo e baixo consumo de energia, e assim com severas limitações de processamento e capacidade de transmissão. Em consequência, nenhuma avaliação e caracterização proposta neste artigo foi encontrada.

3. IoTWild: solução IoT para coleta de fotos de animais silvestres em florestas

A pesquisa de animais silvestres é um tema bastante importante na área de ecologia, não somente para conhecer e entender o comportamento destes animais, mas sobretudo para preservar a sua existência. Dentro deste contexto, o monitoramento da vida silvestre nestes ambientes é fundamental para viabilizar o estudo e acompanhamento da vida destes animais. Diversos trabalhos e projetos utilizam informações para registrar a ocorrência de espécies, fazer estimativas de tamanhos populacionais ou mesmo acompanhar indivíduos específicos de espécies prioritárias, gerando conhecimento sobre hábitos, horários de atividade, áreas de vida e outros que subsidiam decisões sobre o manejo e a conservação das espécies [Steenweg et al. 2017].

Basicamente, esse tipo de monitoramento se dá por meio da instalação de *cameras trap* (câmeras armadilha) espalhadas pela área monitorada. Como explicado anteriormente, este sistema impõe um elevado custo para a coleta das informações capturadas. Assim, com foco em suprir esta demanda, o projeto IoTWild³ visa contribuir com a criação de uma aplicação experimental para automatizar o trabalho de coleta de dados de monitoramento de vida selvagem em floresta com o auxílio de veículos aéreos não-tripulados. O intuito é onerar o mínimo possível o sistema de monitoramento já existente, tanto em termos financeiros quanto em relação ao consumo de energia. Por isso são escolhidos equipamentos de baixo custo, no vasto universo de dispositivos de Internet das Coisas, para extrair as imagens da câmera e as enviarem, por meio de uma rede sem fio, para outro dispositivo acoplado a um *drone*, como ilustrado na Figura 1.

De maneira geral, a aplicação funcionará da seguinte forma: o *drone* sobrevoará próximo a copa das árvores, até que algum dispositivo o identifique e se conecte a sua rede trocando as informações de suas credenciais. Uma vez que o veículo aéreo não-tripulado identifique uma nova conexão ele se estabilizará no ar, e começará a receber os dados; ele permanecerá parado até que: o dispositivo na câmera informe que não há mais dados a serem enviados ou que sua bateria sinalize baixa carga, o que ocorrer primeiro. Para que esta aplicação funcione, possíveis limitações devem ser estudadas e superadas, a saber:

- o consumo energético do dispositivo instalado para processar e enviar as imagens, de maneira que esse não eleve o número necessário de visitas de manutenção para troca de baterias;

³www.gta.ufrj.br/~velloso/projects/iot-wild/

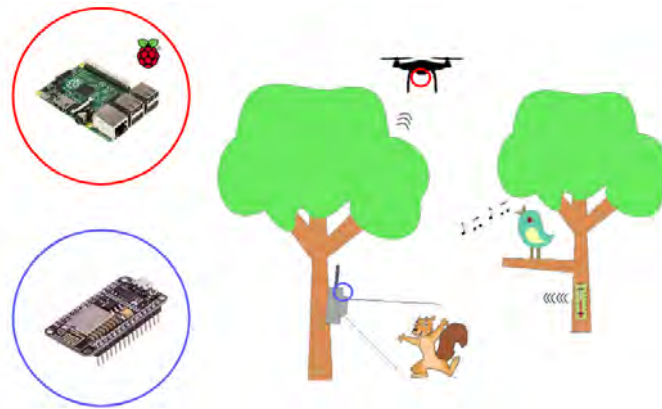


Figura 1. Cenário base da aplicação IoTWild

- a autonomia da bateria do *drone*, que em dispositivos profissionais é de aproximadamente 15 a 20 minutos de voo, de forma que é importante transmitir o máximo possível de imagens, a fim de minimizar as recargas e os retornos do *drone*;
- eficiência da rede sem fio em um ambiente florestal, sendo esta construída com dispositivos de baixo consumo de energia e, por conseguinte, baixa capacidade de processamento e taxa de transmissão.

Neste artigo é avaliado esse último item, analisando assim, o desempenho de uma rede sem fio entre um dispositivo na floresta e outro acima das árvores. Conhecendo a capacidade desta rede, é possível estimar se o tempo de voo do *drone* será suficiente para a aplicação ser minimamente eficiente. Além disso, a caracterização da rede pode permitir a proposição de novos mecanismos e protocolos para aumentar a eficiência da transmissão.

4. Avaliação de desempenho do módulo IoT IEEE 802.11

O padrão IEEE 802.11 foi escolhido por oferecer uma taxa superior às outras tecnologias para IoT, e possuir dispositivos de baixo consumo de energia, como exigido pela aplicação em foco. Vale ressaltar que as fotos capturadas variam de 500 kB à 1 MB de tamanho. A Figura 2(a) ilustra um exemplo de uma dessas fotos. Portanto, para avaliá-lo, foram realizados 5 tipos de testes divididos em 3 cenários distintos para fins de caracterização e compreensão do desempenho dos dispositivos. O objetivo é avaliar os efeitos factuais que o ambiente, sobretudo de floresta, tem sobre a rede IEEE 802.11. Nos experimentos foram observadas as métricas padrões de rede sem fio - taxa de transmissão, alcance máximo, taxa de perda de pacotes e RSSI (*Received Signal Strength Indication* - força do sinal recebido).

4.1. Cenário: Floresta

O cenário utilizado para o teste do equipamento foi a floresta do Parque Nacional da Tijuca, que constitui a maior floresta urbana do mundo ⁴, localizada no Rio de Janeiro, devido a demanda local por um serviço que facilitasse a recuperação dos dados das câmeras que monitoram a sua fauna. O parque corresponde a uma área de 3.200 hectares que é coberta por floresta de mata atlântica. A principal diferença para outros tipos

⁴www.parquedatijuca.com.br



(a) Foto capturada na floresta com as câmeras de monitoramento, tamanho aproximado 600 kB (b) Trilha onde foram realizados os testes

Figura 2. Floresta da Tijuca no Rio de Janeiro caracterizada por densa vegetação e rica fauna

de florestas é que as árvores formam uma cobertura densa e contínua de vegetação que caracteriza uma mata fechada [Freitas et al. 2006], prejudicando a comunicação com o *drone*.

Na Figura 2(b), é possível ver parte da trilha que leva às câmeras de monitoramento. Nesse espaço foram realizados tanto os testes de transmissão com o veículo aéreo não-tripulado sobre a copa das árvores, quanto um segundo teste com ambos dispositivos da rede dentro da floresta.

4.2. Cenários: externo e interno

Os testes internos foram realizados no *campus* da Universidade Federal do Rio de Janeiro, na Ilha do Fundão. O primeiro foi realizado em um corredor de 170 m de comprimento cercado por salas de aula e laboratórios. O teste todo transcorreu em visada direta, ou seja, sem nenhuma obstrução na linha de transmissão dos dois dispositivos. O segundo formato de teste foi realizado entre duas salas de aula, de 20 m de comprimento cada. Cada sala mobiliada com cadeiras, mesas e outros aparatos comuns a esse tipo de ambiente. Inicialmente, os dois dispositivos foram posicionados no mesmo ambiente, em seguida um dispositivo foi movido para a segunda sala, sendo colocado no ponto mais distante em relação ao outro dispositivo. Neste segundo teste, além das barreiras comuns da sala, havia uma parede entre os dispositivos.

O cenário externo foi uma rua pouco movimentada na cidade de Petrópolis, Rio de Janeiro. Ampla e aberta, esta rua também não sofre interferência de redes sem fio da vizinhança, uma vez que é afastada de moradias. A Tabela 1 resume os cenários dos experimentos.

4.3. Configurações/montagem do equipamento

A intenção do projeto é ser capaz de, aproveitando os equipamentos existentes de monitoramento local, coletar imagens capturadas pelas câmeras armadilhas. Os três principais requisitos para a escolha do dispositivo de comunicação são: (i) alta taxa de transmissão; (ii) baixo consumo de energia e (iii) baixo custo. O baixo custo é importante, pois o Parque Nacional da Tijuca, por exemplo, conta com 50 câmeras armadilha

Experimento	Descrição
1	Externo Aberto
2	Interno Corredor
3	Interno com Barreira
4	Externo dentro da Floresta
5	Dentro da Floresta com <i>Drone</i>

Tabela 1. Resumo dos Experimentos Realizados

espalhadas pela floresta; usar dispositivos mais caros nessa situação agregaria elevado custo a verba destinada ao monitoramento do parque, pois teria que ser instalado um dispositivo por câmera. Assim, foram escolhidos para os testes 3 versões do módulo *WiFi* ESP8266, devido ao seu custo-benefício, a saber: ESP-01, ESP-201 e ESP-12E. Os resultados apresentados nesse artigo dizem respeito aos testes realizados com esse último módulo, pois apesar de terem arquiteturas muito semelhantes ele apresentou melhores resultados em relação as outras duas versões.

O módulo de desenvolvimento ESP-12E NodeMCU, da marca DOIT, possui suporte aos padrões IEEE 802.11b/n/g e ainda com pilhas TCP/IP e UDP integradas. Tem processador de 32 bits e *clock* de até 160 MHz⁵. Diferente dos outros, esse módulo ainda possui um *chip* para realizar interface USB-serial e, ainda, regulador de tensão, facilitando sua programação e podendo ser alimentado por tensões entre 4,5 a 9 V. O módulo ESP-12E nos experimentos funcionou como cliente, se conectando e enviando os dados ao ponto de acesso, este último acoplado ao *drone*.

Inicialmente, tanto para o ponto de acesso quanto para o cliente foram usados módulos ESP-12E, entretanto obteve-se baixíssimo desempenho, com vazão aproximada de 100 bps, e alta taxa de retransmissões. Dessa forma, optou-se por usar o microcomputador *Raspberry Pi 3* como ponto de acesso, devido a sua superior capacidade de processamento e armazenamento, uma vez que o projeto visa recolher imagens de diversas câmeras. Além disso, o *Raspberry* não representa um gargalo em termos de consumo de energia, visto que o servidor é acoplado ao *drone* que possui autonomia muito inferior a do *Raspberry*. O microcomputador *Raspberry Pi 3*, suporta o padrão IEEE 802.11n. Tem CPU 1.2 GHz 64-bit quad-core ARMv8⁶.

Cada rodada dos experimentos consiste em enviar 250 pacotes com tamanhos fixos usando os protocolos UDP e TCP do módulo ESP8266 para o *Raspberry* (ponto de acesso no *drone*). As rodadas foram repetidas diversas vezes a fim de se obter uma média para cada métrica medida. A implementação dos protocolos TCP e UDP no módulo é limitada. Observou-se que usando o protocolo TCP só era possível enviar pacotes de tamanho máximo de 2 vezes o MTU, ou seja, aproximadamente 3.000 bytes; caso fosse maior ocorria um reinício espontâneo do módulo sem fio. Semelhante limitação ocorria com o protocolo UDP, porém mais severa; pacotes maiores que o MTU (1.500 bytes) geravam a mesma falha, não havia a possibilidade de fragmentação de pacote. Outra falha que causava o reinício espontâneo no módulo era, no caso do protocolo UDP, a necessidade de um intervalo de espera para processamento de cada pacote antes do envio

⁵<http://wiki.ai-thinker.com/esp8266>

⁶<https://www.raspberrypi.org/learning/hardware-guide/>

do próximo. Isto significa que foi necessário acrescentar este intervalo mínimo entre a transmissão de pacotes consecutivos. Diversos tempos de espera foram testados, o menor tempo necessário encontrado foi de 10 ms por pacote; esse valor independe do tamanho do pacote.

Em todos os testes, para evitar a influência do solo, o módulo sem fio foi posicionado a 1,30 m do chão [Mutschlechner et al. 2013]. Nos testes em terra o *Raspberry Pi* também foi posicionado a mesma distância do chão. Os tamanhos de pacote usados nos testes com o protocolo TCP foram: 128, 512, 1.024, 1.460, 2.000 e 2.920 bytes. Os mesmos tamanhos foram usados para os envios com o protocolo UDP, porém devido as limitações do módulo, não foram enviados pacotes com tamanho 2.000 e 2.920 bytes. E, ainda, como o veículo aéreo não-tripulado não faria voos em dias de chuva ou com forte vento, os testes foram realizados todos em condições parecidas, ou seja, em dias ensolarados e/ou sem fortes ventos e chuvas. No entanto, como boa parte do parque fica em uma montanha do Rio de Janeiro, não foi possível evitar totalmente rajadas de ventos momentâneas, que deslocavam o *drone* da sua posição de coleta, acarretando diferenças nos resultados, mesmo em testes realizados durante o mesmo dia.

Para cada teste, exceto com o *drone*, o procedimento adotado foi: (i) ir afastando os dispositivos até que estes, devido à distância, não conseguissem se comunicar; (ii) registrar a maior distância que houve comunicação entre eles e iniciar o envio de pacotes; (iii) aproximar os módulos, variando a distância, para verificar os efeitos na comunicação. Em todos os testes são observadas as métricas: RSSI, taxa de transmissão e as perdas na transmissão.

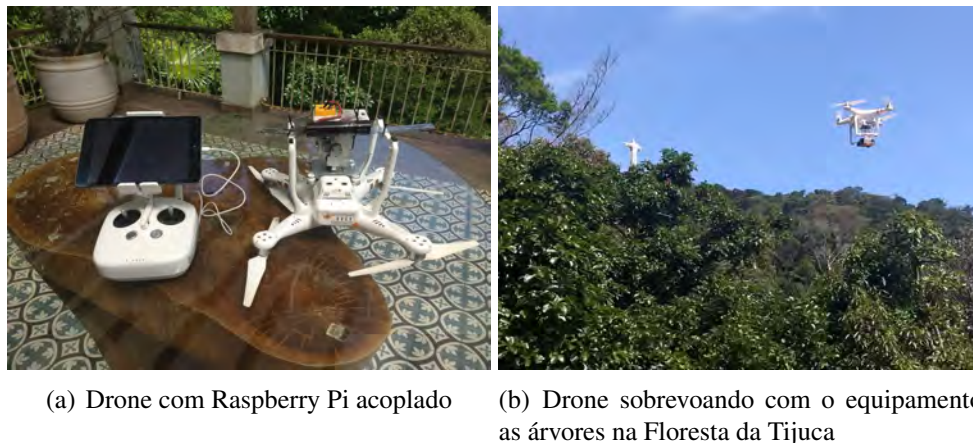


Figura 3. Montagem do equipamento para testes

O parque conta com um *drone* modelo *Phantom 3 Professional* com autonomia de 15 a 20 minutos completamente carregado. Nele foi acoplado o *Raspberry Pi*, como mostrado na Figura 3(a), para realizar os testes de coleta sobre a copa das árvores. Diferentes dos outros experimentos, o quinto não foi até o alcance máximo de comunicação. O veículo aéreo não-tripulado sobrevoou próximo a copa das árvores, com uma distância de segurança, simulando o ambiente proposto para a aplicação IoTWild, e estabilizando ao iniciar a transmissão dos dados.

5. Resultados

Nesta seção são apresentados os principais resultados referentes a cada um dos cenários avaliados. Todas as medidas representam médias com um intervalo de confiança de 95%.

5.1. Alcance máximo

Primeiramente é avaliado o valor da relação sinal ruído a fim de determinar o alcance de cada um dos cenários, exceto o cenário com o *drone*.

A Figura 4 apresenta os resultados de dois dos cenários avaliados. No cenário externo é possível ver a queda do valor do *RSSI* conforme o aumento da distância como esperado. Já no cenário interno de corredor o valor do *RSSI* apresenta um comportamento diferente. Isto porque os corredores conduzem a energia propagada e funcionam como uma guia de onda, melhorando a propagação neste cenário.

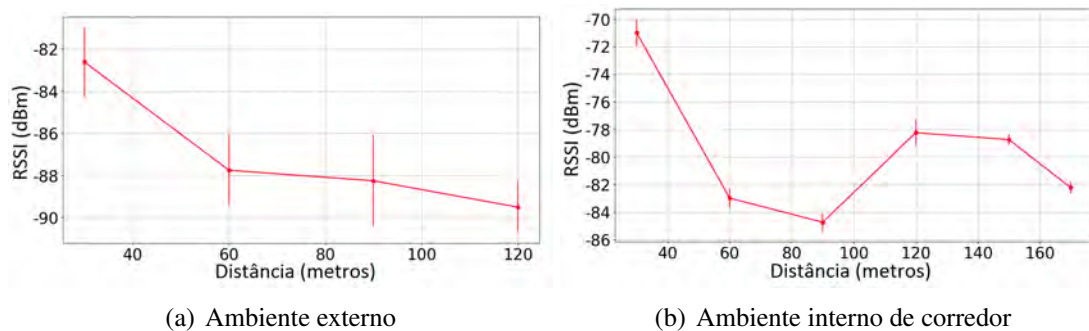


Figura 4. RSSI x Alcance

A Tabela 2 mostra os alcances de cada um dos cenários avaliados. Pode-se perceber que a rede 802.11 dentro da floresta possui alcance semelhante a uma rede interna.

Experimento	Descrição	Alcance (metros)
1	Externo Aberto	120
2	Interno Corredor	170
3	Interno com Barreira	40
4	Externo dentro da Floresta	40

Tabela 2. Alcance máximo

Na floresta com o *drone*, o RSSI médio foi de $-80,75$ dBm com desvio padrão de 2,82, a uma distância de 30 metros. Como mencionado anteriormente, neste cenário não foi medido o alcance máximo, pois não fazia sentido para a aplicação em questão.

5.2. Vazão da rede

Nestes experimentos, buscou-se avaliar a vazão da rede utilizando os protocolos UDP e TCP. Os testes com o protocolo UDP são importantes para mostrar um limite superior da vazão, além de servir como referência para outros trabalhos que avaliam redes IEEE 802.11. Além disso, é possível caracterizar as perdas de pacotes. No entanto, a transmissão das fotos capturadas pela câmera exige confiabilidade na transmissão e

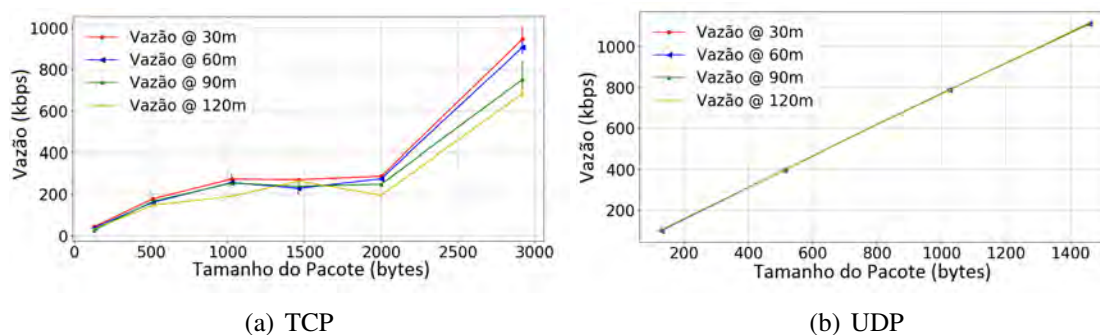


Figura 5. Vazão x tamanho de pacote em ambiente externo

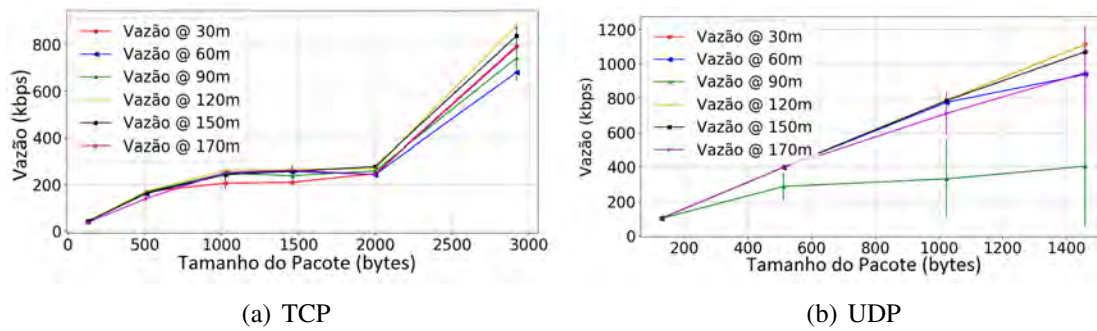
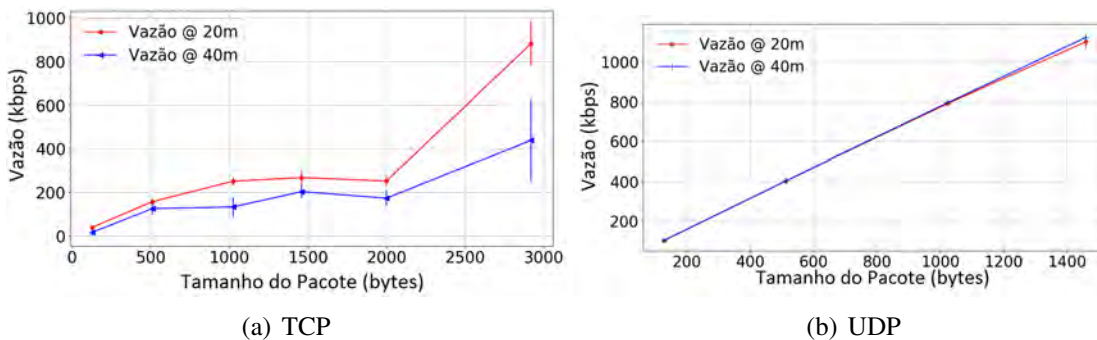
portanto, o uso do protocolo TCP. Assim, o desempenho do protocolo TCP é fundamental para estimar a vazão que o módulo sem fio neste cenário proporcionará.

A Figura 5 ilustra o desempenho do módulo no cenário externo, onde já é possível notar uma grande diferença em relação ao desempenho de dispositivos tradicionais IEEE 802.11, isto é, sem restrições de processamento e consumo de energia. Tanto a vazão alcançada quanto o alcance atingido são inferiores aos resultados obtidos usualmente [Juang et al. 2008]. Um segundo ponto importante é verificar o efeito da restrição do intervalo de tempo mínimo entre dois pacotes, exigido pelo módulo. No UDP (Figura 5(b)) é mais nítido o efeito, pois o acréscimo de 10 ms a cada transmissão de pacote torna o seu tempo de transmissão praticamente desprezível, e por isso, a curva é uma reta, proporcional ao tamanho do pacote. A Figura 5(a) apresenta o resultado para o TCP. Neste caso, é interessante observar que apesar de não haver uma restrição de configurar um intervalo mínimo entre os pacotes transmitidos, o impacto da limitação do módulo de transmitir dois pacotes consecutivos permanece. Este impacto é mais visível quando colocamos o tamanho de pacote máximo permitido pelo módulo. Neste caso, ocorre fragmentação do pacote, mas claramente não há acréscimo do intervalo mínimo entre pacotes para os dois fragmentos, e portanto, a vazão atinge seu máximo. Este fenômeno ocorre em todos os outros cenários.

Por último, pode-se perceber que não houve grandes variações na vazão alcançada para as diferentes distâncias, mesmo com a queda do valor de RSSI, como mostrada na Figura 4(a). Esta queda do RSSI normalmente acarreta uma mudança de modulação do IEEE 802.11, implicando uma redução na taxa de transmissão. No entanto, esta redução não afeta a vazão do módulo, visto que o gargalo é o próprio módulo que não consegue atingir altas taxas de transmissão.

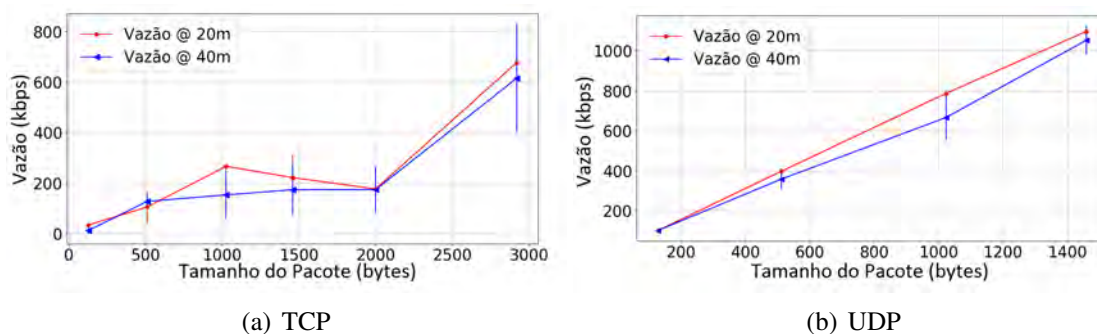
As Figuras 6 e 7 apresentam os resultados para os cenários internos. O primeiro resultado interessante é a vazão no corredor, ilustrado na Figura 6. Apesar de ser um ambiente interno, a vazão é similar ao ambiente externo, onde a transmissão sem fio é sabidamente melhor, por não estar suscetível a tantas reflexões. Como explicado na Figura 4(b), a propagação em um cenário de corredor pode ser considerada com uma onda guiada, e portanto atinge vazões iguais ou superiores a ambientes externos.

A Figura 7 mostra claramente que o resultado da vazão em um cenário interno, do tipo sala, apresenta uma vazão menor que os cenários anteriores e sobretudo que a presença de barreiras, isto é, paredes entre as salas, afeta consideravelmente a vazão do

**Figura 6. Vazão x tamanho de pacote em ambiente interno****Figura 7. Vazão x tamanho de pacote em ambiente fechado com barreiras**

TCP, que para cada perda de pacote diminui sua vazão devido às retransmissões e ao controle de congestionamento.

As Figuras 8 e 9 apresentam os resultados obtidos na floresta tropical. No primeiro experimento, onde cliente e servidor estavam dentro da floresta, a vazão e o alcance atingiram valores semelhantes ao cenário interno com barreiras. Exceto pelo desempenho do TCP que foi prejudicado pela existência de uma parede (Figura 7(a) - 40 metros). Isto significa que a transmissão em florestas densas pode ser comparado a um cenário interno.

**Figura 8. Vazão x tamanho de pacote dentro da floresta**

Finalmente, a Figura 9 mostra o desempenho do TCP e do UDP na comunicação entre o *drone* e o módulo 802.11 dentro da floresta. É importante notar que o desempenho do UDP foi bastante inferior aos outros cenários. Este resultado se deve principalmente pela ocorrência de um número maior de perdas de pacotes neste cenário, como será mos-

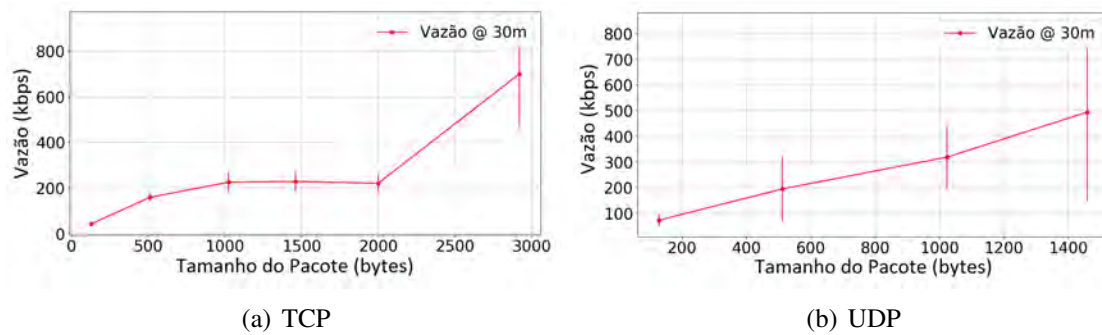


Figura 9. Vazão x tamanho de pacote experimento com drone

trado em seguida. Um dos motivos para este resultado é o fato de o *drone* não conseguir ficar estável exatamente no mesmo lugar, devido sobretudo ao vento, e o peso extra do equipamento de comunicação acoplado, que dificultava a sua estabilidade. O resultado da vazão do TCP, ilustrado na Figura 9(a), demonstra a viabilidade da comunicação acima da copa das árvores, com um desempenho similar ao cenário interno.

5.3. Taxa de perda de pacotes

A partir dos experimentos da transmissão UDP com o *drone*, observou-se que a perda de pacotes neste cenário foi significativa. Portanto, a Tabela 3 mostra a média da porcentagem de perdas para cada tamanho de pacote. É interessante notar a alta taxa de perda de pacotes para todos os tamanhos, que aumenta conforme o tamanho do pacote, como era esperado, devido à taxa de erro de bit do meio de transmissão (BER - Bit Error Rate). Exceto para o maior tamanho de pacote, que sofreu menos perdas que os pacotes de 512 e 1.024 bytes. Neste caso específico, o vento havia diminuído consideravelmente, e por consequência, o piloto conseguiu deixá-lo mais estável. Vale ressaltar que os resultados de perdas apresentaram grande variação, com rodadas com poucas perdas e outras com mais perdas. Isto se deve principalmente a variação de posição do *drone*.

Tamanho do Pacote (bytes)	Média de Perdas
128	4,8%
512	39,4%
1.024	44,8%
1.460	27,8%

Tabela 3. Experimento 5: Floresta com Drone

Para melhor caracterizar as perdas de pacotes na floresta, foram avaliadas as perdas de pacotes consecutivas. A Figura 10 apresenta a PMF (*Probability Mass Function*) e CDF (*Cumulative Distribution Function*) das perdas consecutivas para pacotes de 128 e 1460 bytes. Primeiramente, é interessante observar que a grande maioria dos pacotes perdidos não são consecutivos. Para pacotes pequenos, as perdas consecutivas não passa de três pacotes, enquanto que para pacotes maiores este valor pode chegar a nove. No entanto, menos de 14% das perdas de pacotes de 1.460 bytes são superiores a três pacotes consecutivos. A partir destes resultados é possível considerar a hipótese de que neste cenário o controle de congestionamento do TCP pode não ser eficiente, como já foi estudado para redes sem fio [Francis et al. 2012].

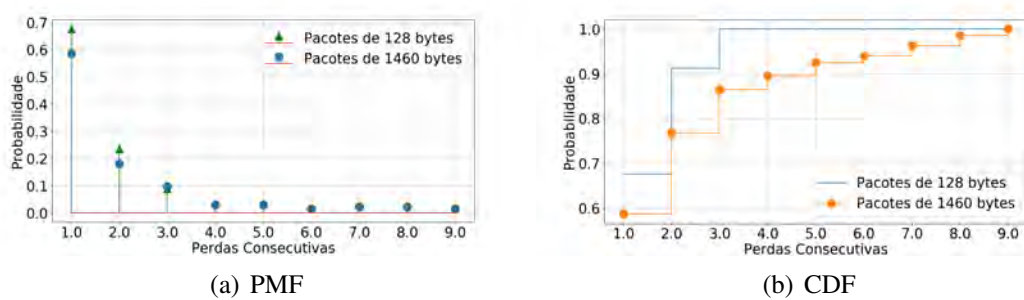


Figura 10. Perdas para o menor e maior tamanho de pacote

6. Conclusões

Este trabalho avalia o desempenho de uma rede IEEE 802.11 na floresta montada puramente com dispositivos IoT. Diferente de trabalhos anteriores, o enfoque é caracterizar este tipo de rede neste ambiente e verificar sua viabilidade para coleta de dados de câmeras com o auxílio de veículos aéreos não-tripulados. Para fins de caracterização, outros tipos de cenários além do florestal foram avaliados: interno e externo. Os testes mostraram que o cenário de floresta com o *drone* é comparável ao cenário interno com barreiras. Os resultados mostram que as limitações dos dispositivos, principalmente o tempo de processamento necessário para o módulo enviar os pacotes de dados, restringem consideravelmente o desempenho da rede sem fio. Essas limitações associadas à influência da vegetação implicaram em uma diminuição na taxa de transmissão e um aumento das perdas de pacote. Entretanto, ainda que esta rede não tenha conseguido alcançar taxas ideais, ela se mostrou minimamente eficiente para coletar imagens dentro da autonomia de voo de um *drone* padrão, que é de 15 a 20 minutos. Portanto, estes resultados demonstram a viabilidade de automatizar a coleta de grandes volumes de dados em florestas densas.

Como trabalho futuro pretende-se avaliar o consumo de energia adicionado pelo módulo 802.11 no sistema de coleta por câmeras. Além disso, seria interessante estudar simplificações no modelo de controle de congestionamento do TCP que possam ser implementados em dispositivos de baixa capacidade de processamento a fim de melhorar o desempenho da comunicação entre o módulo e o *drone*.

Agradecimentos

Os autores agradecem todo o apoio recebido pelo Parque Nacional da Tijuca. Em especial, o Ernesto de Castro, diretor do parque, que disponibilizou toda a infraestrutura necessária para realizar as medidas, a Katyucha Silva, coordenadora de manejo e pesquisa e os funcionários do ICMBIO, pilotos do *drone*, Marcelo Mendes e Guilherme Medeiros. Este trabalho teve apoio do CNPq, CAPES, FAPERJ e processo nº 15/24494-8, FAPESP.

Referências

- Abdulla, A. E. A., Fadlullah, Z. M., Nishiyama, H., Kato, N., Ono, F., e Miura, R. (2014). An Optimal Data Collection Technique for Improved Utility in UAS-aided Networks. Em *IEEE INFOCOM'14*.
- Ding, X., Sun, G., Yang, G., e Shang, X. (2016). Link Investigation of IEEE 802.15.4 Wireless Sensor Networks in Forests. *Sensors*, 16(7).

- Francis, B., Narasimhan, V., Nayak, A., e Stojmenovic, I. (2012). Techniques for Enhancing TCP Performance in Wireless Networks. Em *ICDCSW'12*.
- Freitas, S. R., Neves, C. L., e Chernicharo, P. (2006). Tijuca National Park: two pioneering restorationist initiatives in Atlantic forest in southeastern Brazil. *Brazilian Journal of Biology*, 66(4):975–982.
- Gu, Z., Hua, Q.-S., Wang, Y., e Lau, F. C. (2013). Reducing Information Gathering Latency through Mobile Aerial Sensor Network. Em *IEEE INFOCOM'13*.
- Heng Siong Lim, Ka Kien Lo, Abbas, M., Kae Hsiang Kwong, Hashim, F., Ngoh, A., e Torshizi, S. (2012). An Investigation of Vegetation Effect on the Performance of IEEE 802.11n Technology at 5.18 GHz. Em *IET ICWCA'12*.
- Huu-Quoc Nguyen, Ton Thi Kim Loan, Bui Dinh Mao, e Eui-Nam Huh (2015). Low cost real-time system monitoring using Raspberry Pi. Em *International Conference on Ubiquitous and Future Networks*.
- Jeong, H., Lee, C., Ryu, J., Choi, B.-C., e Ko, J. (2015). Communicating "in the Air-Studying the Impact of UAVs on Sensor Network Data Collection. Em *SenSys'15*.
- Juang, M., Kuang-Ching Wang, e Martin, J. (2008). A Measurement Study on Link Capacity of a High Stress IEEE 802.11b/g Network. Em *IEEE ICCCN'08*.
- Kriara, L., Molero, E. C., e Gross, T. R. (2016). Evaluating 802.11ac features in indoor WLAN. Em *ACM WiNTECH'16*.
- Mutschlechner, M., Baldemaier, P., Handle, P., e Dressler, F. (2013). Wireless in the woods: Experimental evaluation of IEEE 802.11 a/b/g in forested environments. *12. GI/ITG FACHGESPRÄCH SENSORNETZE*, 60:5.
- Nikhade, S. G. (2015). Wireless sensor network system using Raspberry Pi and zigbee for environmental monitoring applications. Em *ICSTM'15*.
- Petajajarvi, J., Mikhaylov, K., Roivainen, A., Hanninen, T., e Pettissalo, M. (2015). On the coverage of LPWANs: range evaluation and channel attenuation model for LoRa technology. Em *International Conference on ITS Telecommunications (ITST'15)*.
- Say, S., Inata, H., Liu, J., , e Shimamoto, S. (2016). Priority-Based Data Gathering Framework in UAV-Assisted Wireless Sensor Networks. *IEEE Sensors Journal*, 16(14):5785–5794.
- Steenweg, R., Hebblewhite, M., Kays, R., Ahumada, J., Fisher, J. T., Burton, C., Townsend, S. E., Carbone, C., Rowcliffe, J. M., Whittington, J., Brodie, J., Royle, J. A., Switalski, A., Clevenger, A. P., Heim, N., e Rich, L. N. (2017). Scaling-up camera traps: monitoring the planet's biodiversity with networks of remote sensors. *Frontiers in Ecology and the Environment*, 15(1):26–34.
- Vasisht, D., Kapetanovic, Z., Won, J., Jin, X., Chandra, R., Sinha, S., Kapoor, A., Sudarshan, M., e Stratman, S. (2017). FarmBeats: An IoT Platform for Data-Driven Agriculture. Em *USENIX NSDI'17*.
- Wang, K.-C., Venkatesh, G., Pradhananga, S., Lokala, S., Carter, S., Isenhower, J., e Vaughn, J. (2008). Building wireless mesh networks in forests. Em *ACM WiNTECH'08*.

Um Mecanismo de Comutação de Servidores CoAP para Aumento de Disponibilidade dos Serviços de IoT

Jasiel G. Bahia¹ e Miguel Elias M. Campista¹

¹Universidade Federal do Rio de Janeiro – GTA/PEE-COPPE/DEL-Poli

{jasiel,miguel}@gta.ufrj.br

Abstract. *The Internet of Things (IoT) challenges network scalability, given the huge number of connected devices, and also creates a fertile scenario to explore redundant resources for service robustness increasing. Consequently, new IoT protocols have been proposed, being CoAP (Constrained Application Protocol) one of the most important for the application layer. In this work, we propose a mechanism that conducts CoAP servers selection and switching at the client, aiming to increase services availability for an application. The mechanism uses CoAP capabilities to switch servers, by following a sorted list of IP addresses obtained from a central infrastructure. This list is built according to the application requirements and is stored at the client. The mechanism is evaluated in a simulator designed for IoT (Cooja) and the results obtained show that the mechanism increases the reliability and the robustness of the IoT service provisioning and, furthermore allows energy consumption balance among the servers.*

Resumo. *A Internet das Coisas (Internet of Things - IoT) desafia a escalabilidade em rede, dado o enorme número de dispositivos interconectados, e cria um cenário fértil para exploração de recursos redundantes para aumentar a robustez dos serviços. Consequentemente, novos protocolos vêm sendo propostos, sendo o CoAP (Constrained Application Protocol) um dos principais de IoT para a camada de aplicação. Este trabalho apresenta um mecanismo de seleção e comutação de servidores CoAP que visa aumentar a disponibilidade dos serviços necessários a uma aplicação. As funções do próprio CoAP são usadas para realizar a comutação dos servidores, seguindo uma lista ordenada de endereços IP obtida a partir de uma infraestrutura central. Essa lista é construída com base nos requisitos da aplicação e fica armazenada no cliente. O mecanismo é avaliado em um simulador específico de IoT (Cooja) e os resultados mostram que o mecanismo aumenta a confiabilidade e a robustez na obtenção de serviços de IoT, além de permitir o balanceamento do consumo de energia entre os servidores.*

1. Introdução

Bilhões de pessoas utilizam a Internet no mundo para aplicações que vão desde navegação *web* até interação em redes sociais [Group 2016]. Ao mesmo tempo em que o número de usuários da Internet cresce, a tecnologia embarcada em dispositivos eletrônicos evolui rapidamente, possibilitando que objetos do cotidiano também se comuniquem e processem dados. Essa evolução leva a um novo paradigma, onde os objetos também se conectam e interagem com o ambiente, transformando-se em produtores e consumidores

de dados. Esse novo paradigma, conhecido como Internet das Coisas (IoT - *Internet of Things*) [Al-Fuqaha et al. 2015], já é uma realidade na indústria, onde se busca constantemente eficiência no uso dos recursos e disponibilidade dos serviços.

A ampliação da diversidade de aplicações através da Internet das Coisas implica uma imensa quantidade de dispositivos conectados, exigindo um compromisso entre a grande quantidade de dados gerados e as limitações dos dispositivos em redes de IoT. Dessa forma, para desenvolver soluções adequadas, é necessário atentar para aspectos como: escalabilidade da rede, consumo de energia e disponibilidade dos serviços. Com relação ao consumo de energia, uma medida importante é controlar o ciclo de trabalho (*duty-cycle*) do sistema de comunicação, pois esse é o que mais demanda energia do dispositivo. Isso faz com que os protocolos empregados na Internet convencional não sejam adequados para uso em dispositivos de IoT [Al-Fuqaha et al. 2015], já que o consumo de energia não foi um pré-requisito. Como parte do desenvolvimento de novos protocolos, é importante que a análise de desempenho dos mecanismos empregados considere as limitações dos dispositivos de IoT em termos de processamento, armazenamento e energia. Além disso, a imensa quantidade de dispositivos conectados [Cisco 2017] provoca um aumento substancial no fluxo de dados e desafia o potencial de escalabilidade das redes. Os mecanismos que não consideram essas restrições podem provocar o rápido esgotamento dos recursos da rede e da energia dos dispositivos, comprometendo a disponibilidade dos serviços. Por outro lado, a multiplicidade de dispositivos conectados à Internet, contendo diversos sensores, oferece grande potencial de redundância de recursos. Essas redundâncias podem ser usadas para aumentar a robustez das redes de IoT, garantindo a disponibilidade dos serviços mesmo diante de instabilidades na rede, bastante comuns em redes LLN (*Low-power and Lossy Networks*) [Granjal et al. 2015].

Na literatura, as restrições de bateria dos dispositivos de IoT provocam uma grande busca por novos mecanismos mais eficientes energeticamente. Há propostas que reduzem o consumo de energia através do aumento da eficiência no roteamento [Barbato et al. 2013, Iova et al. 2014], outros pela redução do número de pacotes transmitidos [Jan et al. 2014] e outros ainda pelo uso de controle adaptativo do ciclo de trabalho do dispositivo [Afzal et al. 2017]. Todos esses trabalhos, porém, não aliam a eficiência energética à garantia de confiabilidade e disponibilidade dos serviços de IoT. Com relação ao provisionamento de serviços, comumente, os trabalhos consideram uma arquitetura centralizada, onde o gerenciamento dos serviços fica à cargo de um dispositivo concentrador que atua como um intermediário na comunicação entre os dispositivos [FIWARE 2011, Banks and Gupta 2014, Jeyaraman and Telfer 2012]. Como consequência, uma falha no dispositivo concentrador pode comprometer a disponibilidade dos serviços. Dessa forma, uma saída é habilitar o cliente a se comunicar diretamente com os servidores e a realizar a comutação entre eles, permitindo a seleção dos servidores e a obtenção dos serviços de acordo com os requisitos da aplicação. Dentre os principais protocolos desenvolvidos para redes LLN, o CoAP (*Constrained Application Protocol*) [Shelby et al. 2014], da camada de aplicação, destaca-se por permitir a comunicação por IP entre cliente e servidor via Internet e por possuir funcionalidades que favorecem a eficiência energética no provisionamento de serviços de IoT. A comutação de servidores é essencial sobretudo em redes LLN, onde os servidores podem ficar indisponíveis por alguma instabilidade ou mesmo devido a mecanismos de controle de demanda [Bahia and Campista 2017].

Este trabalho visa aumentar a confiabilidade e a robustez na obtenção de serviços de IoT, através do aumento da disponibilidade de servidores CoAP. O principal objetivo é propor um mecanismo que faça a seleção e comutação de servidores CoAP, baseando-se numa lista ordenada de servidores. A lista é construída e fornecida por uma infraestrutura central de acordo com os requisitos da aplicação do cliente. Cada cliente, então, armazena a lista e realiza a comutação de servidores segundo a ordem de prioridade pré-definida, de modo a permitir a obtenção dos serviços necessários à sua própria aplicação. O mecanismo proposto aumenta a confiabilidade e a robustez na obtenção de serviços de IoT, além de permitir o balanceamento do consumo de energia entre os servidores CoAP. O mecanismo proposto é avaliado através de experimentos realizados no Cooja [Thingsquare 2016], que é o simulador da plataforma de desenvolvimento do Contiki OS, desenvolvido para testes de redes LLN. Os experimentos consideram um cenário de IoT no qual os dispositivos possuem recursos limitados e interagem entre si para realizar funções de controle, podendo haver recursos redundantes que permitam o aumento da disponibilidade dos serviços necessários às aplicações dos clientes. Os resultados mostram que o mecanismo aumenta a robustez dos serviços e permite o balanceamento do consumo de energia entre os servidores, mediante a comutação de servidores.

Este artigo está organizado da seguinte forma. A Seção 2 apresenta uma arquitetura típica de IoT e o funcionamento do CoAP. Na Seção 3, o funcionamento do mecanismo de seleção e comutação de servidores é apresentado e, a seguir, na Seção 4, as condições de análise do mecanismo são descritas. A análise dos resultados é discutida na Seção 5. Por fim, a Seção 6 conclui este trabalho e sugere direções futuras.

2. Visão Geral de IoT e do Protocolo CoAP

Um dos principais objetivos da Internet das Coisas (IoT) é promover a interoperabilidade entre os dispositivos de redes LLN [Granjal et al. 2015] e as aplicações da Internet convencional. As redes LLN são compostas por dispositivos limitados em capacidade de processamento, armazenamento e energia, que podem assumir o papel de servidor ou cliente conforme a aplicação. No papel de servidor, os dispositivos disponibilizam seus recursos para clientes na rede local e na Internet. Já no papel de cliente, os dispositivos consultam os servidores para obter as informações necessárias para a sua aplicação. Considera-se que, para cada rede local, exista a figura do roteador de borda, responsável por conectar os dispositivos entre si e à Internet, divulgando os recursos disponíveis dos servidores e fazendo a tradução entre os protocolos (p. ex., HTTP/CoAP). Os dispositivos de IoT tipicamente se comunicam através de uma rede sem fio, utilizando um rádio de baixa potência, tornando-se mais susceptível a perdas.

As soluções empregadas na Internet convencional não são compatíveis com os novos cenários introduzidos pelas redes LLN, dadas as limitações dos dispositivos de IoT e o nível de escalabilidade exigido. Sendo assim, uma nova pilha de protocolos vem sendo desenvolvida para fomentar aplicações futuras de IoT [Palattella et al. 2013, Granjal et al. 2015]. Nessa nova pilha, o protocolo proposto para a camada de aplicação é o CoAP [Shelby et al. 2014], projetado para aplicações M2M (*Machine-to-Machine*). O CoAP emprega um modelo de interação pedido/resposta fim-a-fim que suporta descoberta de serviços e inclui conceitos da *web* como URI (*Uniform Resource Identifier*) e tipos de dados da Internet. Além disso, o CoAP pode ser traduzido para o HTTP, oferecendo suporte a multicast e comunicação assíncrona, com baixa sobrecarga e alta simplicidade

para ambientes com recursos limitados. O modelo de interação do CoAP é similar ao cliente/servidor do HTTP, porém interações M2M tipicamente possibilitam a um dispositivo agir simultaneamente como cliente e como servidor. Uma requisição CoAP é enviada pelo cliente solicitando uma ação sobre um recurso (identificado pelo URI) do servidor. O servidor então responde, podendo incluir uma representação do recurso, p. ex., o valor da temperatura. A descoberta de serviços é feita com um único pedido endereçado a um URI específico do servidor, conforme definido pelo protocolo CoAP. O servidor responde a esse pedido enviando uma lista com os recursos que ele oferece.

Aplicações de IoT frequentemente envolvem dispositivos desempenhando ou o papel de servidor ou o papel de cliente de um dado recurso. Nessa direção, o CoAP suporta uma funcionalidade chamada de Observação [Hartke 2014], na qual o cliente pode registrar-se como observador de um recurso do servidor através de um GET modificado. O servidor estabelece um relacionamento de observação entre o cliente e o recurso, onde cada recurso tem a sua própria lista de observadores e cada cliente só pode se registrar uma única vez na mesma lista. Um cliente registrado como observador de um recurso recebe notificações do servidor sem a necessidade de gerar pedidos adicionais, como no caso dos clientes chamados sob demanda. Contudo, a lista de observadores de um recurso tem um tamanho máximo, que depende do espaço disponível em memória no servidor para armazenamento das listas de cada recurso. A função Observação do CoAP evita que o cliente envie uma requisição ao servidor sempre que desejar um dado ou que tenha que manter uma sessão aberta, como no caso do HTTP sobre o TCP. Essa é uma das características que torna o CoAP indicado para redes LLN, já que permite que os clientes observadores acompanhem as mudanças nos recursos de seu interesse ao mesmo tempo que reduz a sobrecarga do servidor e da rede [Ludovici et al. 2012]. A Figura 1 ilustra uma arquitetura típica de IoT composta por nós servidores (nós s), nós clientes (nós c) e um roteador de borda r que interconecta a rede de IoT à Internet. Note ainda a presença de nós clientes observadores (nós o) e a presença de nós na Internet, que podem se comunicar com a rede de IoT usando HTTP/TCP ou CoAP/UDP, por exemplo.

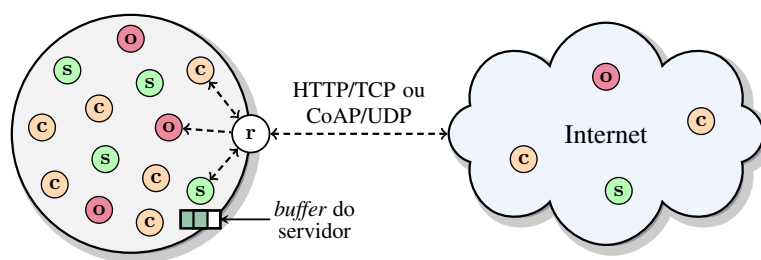


Figura 1. Arquitetura de uma rede de IoT com a presença de observadores.

3. Mecanismo de Comutação de Servidores CoAP Proposto

O principal objetivo do mecanismo proposto é aumentar a disponibilidade da rede e, consequentemente, dos serviços de IoT prestados. Para isso, o cliente realiza a comutação dos servidores a partir da consulta à sua lista de endereços de servidores que prestam o serviço desejado. Considera-se que essa lista de servidores é obtida a partir de uma infraestrutura central, papel aqui desempenhado pelo *gateway* (roteador de borda) da rede de IoT. Essa infraestrutura, dentre outras funções, gerencia os serviços disponibilizados na rede local e para a Internet, atualizando e armazenando as informações sobre

os servidores, tais como: endereços IP, recursos, status do dispositivo (carga da bateria, taxa de uso do rádio e vagas na lista de observadores). A atualização e o gerenciamento dessas informações é possível utilizando o próprio CoAP [Shelby et al. 2014]. Esta seção apresenta primeiramente a *obtenção da lista de servidores* e, em seguida, o processo de *comutação de servidores*.

Obtenção da lista de servidores: O cliente pode solicitar uma lista de servidores que ofereçam o mesmo tipo de serviço (p. ex., medição de temperatura) ou serviços de tipos diferentes (p. ex., medição de temperatura e umidade) para atender uma determinada aplicação. A lista de servidores possui uma ordem de prioridade de consulta baseada nos requisitos da aplicação do cliente. Desse modo, ao solicitar a lista ao *gateway*, o cliente deve informar os requisitos para que seja gerada uma lista ordenada da forma desejada. Esses requisitos são usados para caracterizar os servidores através de parâmetros como: número de vagas na lista de observadores, grau de confiabilidade da informação, localização do dispositivo e taxa de uso do rádio (demanda). Se o requisito para ordenação for, por exemplo, servidores com o serviço desejado que estejam mais próximos de uma determinada localização, o *gateway* deve calcular o grau de proximidade dos servidores candidatos e posicioná-los na lista de servidores por ordem de proximidade.

A obtenção da lista de servidores prevê a utilização do próprio CoAP para realizar as transações entre cliente e *gateway*. Dessa forma, o cliente envia ao *gateway* um pedido solicitando uma lista de servidores, informando os requisitos no campo de opções da mensagem CoAP. O *gateway*, por sua vez, age como um servidor CoAP que possui um recurso que aciona um seletor de servidores. Esse recurso é identificado pelo seu próprio URI que pode ser endereçado pelos clientes. O *gateway*, ao receber pedidos para esse recurso, inicia a elaboração da lista de servidores baseando-se nos requisitos do cliente presentes no campo de opções da mensagem recebida. O *gateway* inclui nessa lista os endereços IP dos servidores, contidos na sua base de dados, que melhor atendem ao pedido e encaminha a lista na mensagem de resposta do recurso CoAP. Por padrão, o *gateway* monta e atualiza a sua base de dados, conforme recebe as solicitações dos clientes, buscando os servidores que oferecem os serviços desejados. O *gateway* obtém a lista de recursos de cada servidor utilizando o mecanismo de descoberta de serviços do próprio CoAP, fazendo consultas sob demanda sempre que necessitar dessa informação. Finalmente, o *gateway* envia a lista já com a ordem requerida, mas o cliente pode modificar essa ordem localmente a qualquer momento, conforme a sua conveniência.

Comutação de servidores: O mecanismo de comutação de servidores proposto assume que uma lista de endereços de servidores ordenada de acordo com a aplicação já exista. Sendo assim, o cliente realiza a consulta aos servidores seguindo a ordem da lista recebida. A cada consulta, caso o servidor responda, o cliente atualiza o status do servidor para disponível; caso contrário, o servidor é considerado indisponível. Se o cliente quiser se tornar um observador, ele deve fazer o pedido de inscrição. Caso a inscrição seja aceita, o serviço é fornecido em forma de notificações; caso contrário, o cliente ainda pode solicitar o serviço sob demanda. Assim, o modo de provimento de serviços depende dos critérios de seleção de observadores e do número de vagas disponíveis em cada servidor. Se a preferência do cliente for por servidores que aceitem novos observadores, ele muda para o próximo endereço da lista e tenta se inscrever no novo servidor. Ao final da lista, caso nenhum servidor tenha aceito a sua inscrição, o cliente poderá solicitar o serviço sob

demanda dos servidores disponíveis, conforme a ordem de prioridade da lista. Se todos os servidores da lista estiverem indisponíveis, o cliente solicita ao *gateway* uma nova lista de servidores. Caso a lista fornecida pelo *gateway* seja igual a anterior, o cliente aguarda um tempo de espera e reinicia o ciclo de consulta. Se houver mudança na lista de servidores, o mecanismo reinicia o ciclo imediatamente.

No mecanismo proposto, o cliente pode finalizar as suas consultas assim que obtiver a resposta desejada ou continuar consultando os demais servidores da lista até completar o ciclo, caso a lista seja composta por servidores que oferecem o mesmo tipo de serviço. Dessa forma, o cliente pode obter a informação de um único servidor ou combinar as informações dos vários servidores. Caso a lista seja constituída de servidores com recursos de tipos diferentes, o cliente deve consultar todos os servidores da lista, obtendo os serviços requeridos pela aplicação. O mecanismo de comutação de servidores é bastante útil para tirar proveito das redundâncias de recursos, permitindo a recomposição dos serviços em caso de indisponibilidade de algum servidor da lista. Ao flexibilizar a obtenção dos serviços pelo próprio cliente, o mecanismo confere maior robustez no provimento de serviços de IoT. Destaca-se que as arquiteturas propostas para IoT, em muitos casos, consideram que a disponibilização dos recursos dos servidores é feita por um servidor central ou um *broker*. Porém, o mecanismo de comutação de servidores, em conjunto com o CoAP, possibilita o acesso aos recursos pelo próprio cliente, uma vez que ele possui os endereços dos servidores e pode comunicar-se com eles diretamente. Se os servidores da lista possuírem um mecanismo de controle de demanda baseado no ciclo de trabalho, como proposto em trabalho anterior [Bahia and Campista 2017], o mecanismo de comutação de servidores permite ao cliente comutar para um servidor que possa garantir o provimento do serviço (p. ex., servidores com mais energia ou com menor demanda). Isso permite um balanceamento de carga e de consumo de energia entre os servidores.

Em caso de indisponibilidade do servidor, o cliente pode perder pacotes (pedidos e notificações) até identificar o problema. Nesse caso, um cliente sob demanda, ao não receber resposta a um pedido, realiza retransmissões até atingir o número máximo previsto na configuração do CoAP. Se nenhum dos pedidos for atendido, o cliente aguarda o tempo de estouro do CoAP para considerar o servidor indisponível. No mecanismo proposto, o cliente pode efetuar a comutação para outro servidor da lista. No caso do cliente observador de um recurso, o servidor passa a condição de indisponível quando o cliente deixa de receber as notificações esperadas. O critério utilizado no mecanismo foi um tempo de tolerância para o atraso nas notificações, acima do qual o cliente considera que o recurso ou o servidor está indisponível. Mesmo já tendo comutado para outro servidor, o mecanismo pode verificar periodicamente se o servidor preferido está disponível novamente ou ainda atualizar a lista de servidores, perguntando ao *gateway* se há servidores melhores para recomposição da lista.

A configuração do mecanismo de comutação de servidores requer a definição dos seguintes parâmetros: tipo de atendimento preferido (*SERVICE*), tolerância de atraso para notificações (*tol*) e tempo de espera para reinício do ciclo de comutação ($t_{reiniciar}$). O parâmetro *SERVICE* determina se o mecanismo prioriza a busca por uma vaga de observador (*SERVICE* = 2) ou se prioriza o serviço do primeiro servidor que responder (*SERVICE* = 1), mesmo que o serviço disponível seja oferecido apenas sob demanda. O parâmetro *tol* especifica o número de notificações sucessivas não recebidas para que o

cliente considere o servidor indisponível. Finalmente, o parâmetro $t_{reiniciar}$ estabelece o tempo de espera para reinício do mecanismo após todos os servidores serem considerados indisponíveis. Esses parâmetros devem ser definidos de acordo com a aplicação, considerando as restrições de tolerância a atraso dos serviços. Quanto maior o tol , maior é o tempo de espera para atualizar uma informação após sucessivas notificações não recebidas. Uma tolerância excessiva pode comprometer a confiabilidade da informação, pois ela pode ficar muito defasada.

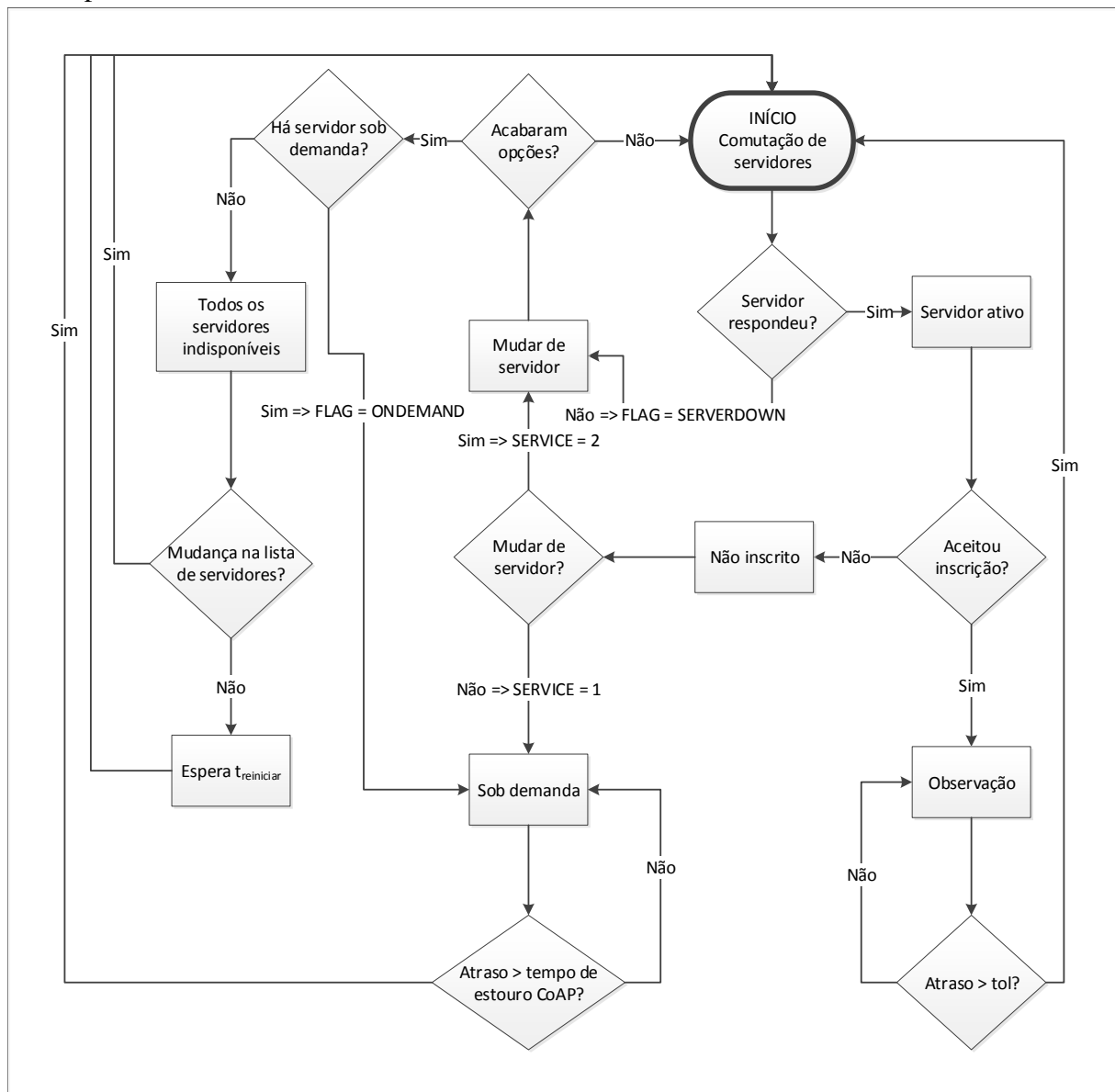


Figura 2. Funcionamento do mecanismo de comutação de servidores.

O mecanismo proposto é útil, em particular, para aplicações que dependam de informações de diversos servidores, tirando proveito inclusive da redundância de recursos (p. ex., monitoramento ambiental). Dadas as limitações dos dispositivos de LLN, embora o cliente não consiga usar uma lista grande de servidores, o mecanismo proposto confere maior flexibilidade no provimento de serviços de IoT. Além disso, a comutação de servidores CoAP promove o aumento da disponibilidade dos serviços de IoT. A Figura 2 mostra o fluxograma do funcionamento do mecanismo de comutação de servidores.

4. Análise do Mecanismo de Comutação de Servidores

Esta seção apresenta as condições de análise do mecanismo proposto.

4.1. Ambiente de simulação

O simulador Cooja, desenvolvido para análise de redes LLN, foi utilizado nos experimentos. O Cooja pertence à plataforma de desenvolvimento do Contiki OS [Thingsquare 2016] e foi usado para aumentar o realismo das simulações. O Contiki OS é um sistema operacional de código aberto desenvolvido especialmente para dispositivos com recursos limitados, usados tipicamente em Redes de Sensores Sem Fio e em redes de IoT. Os dispositivos usados na simulação são emulados a partir dos seus correspondentes comerciais, o que aumenta a proximidade do comportamento simulado com aquele que se espera encontrar em experimentos com dispositivos reais.

4.2. Experimentos

Os experimentos consideram uma rede de IoT LLN, semelhante a da Figura 1, composta por dez clientes CoAP, um número variável de servidores CoAP e um roteador de borda (*gateway*). Os servidores possuem recursos redundantes e o *gateway* executa o protocolo RPL. A comunicação entre os clientes e o servidor ocorre sempre com a ajuda do roteador de borda, tornando indiferente ao servidor se o pedido veio de um cliente interno ou externo. Para cada configuração, a duração da simulação foi de 10 minutos. O posicionamento dos dispositivos na rede muda aleatoriamente a cada rodada de simulação.

As simulações foram executadas em uma VM rodando o Contiki OS 3.0. A tecnologia utilizada em todos os dispositivos da rede foi o módulo TMote Sky (Mote IV), com tecnologia de rede IEEE 802.15.4, CPU de 16 bits, 10kB de RAM e 48kB de memória Flash. Para reduzir o uso de memória dos dispositivos, foi utilizada a configuração NullRDC (mantém rádio sempre ligado para recepção), NullMAC (sem funcionalidades de MAC) e tamanho padrão de buffer uIP de 256 bytes. O tamanho da fila do servidor (*buffer*) deve ser definido previamente na configuração do dispositivo, de acordo com a sua capacidade de memória. Neste trabalho, o dispositivo foi configurado para permitir, no máximo, 4 transações CoAP na fila do servidor. O tamanho da lista de observadores pode ser no máximo igual ao número máximo de transações CoAP permitidas simultaneamente. Nos experimentos, optou-se por deixar sempre no mínimo uma transação reservada para clientes sob demanda. Os recursos de cada servidor devem ser definidos antes da compilação do programa, incluindo o período de notificação para recursos periódicos.

Nos experimentos, os clientes CoAP podem ser clientes observadores (notificados periodicamente) ou clientes sob demanda. Para cada configuração, a periodicidade de notificações do servidor para os observadores foi mantida em $t_p = 10\text{ s}$, intervalo razoável para a atualização da informação em aplicações de monitoramento. Ressalta-se que esse valor deve ser configurado previamente para cada recurso que oferece a opção de Observação, não podendo ser alterado após a sua disponibilização, a menos que seja feita uma nova compilação do código do servidor. Os parâmetros variáveis foram: número de observadores (p) e tamanho da lista de servidores (n_s). Embora o simulador permita a inclusão de mais dispositivos na rede, esses parâmetros foram considerados suficientes para realizar as observações desejadas, evitando-se forçar o dispositivo servidor a operar no limite de sua capacidade.

A análise do servidor CoAP usou as seguintes métricas: quantidade de pacotes CoAP gerados, taxa de perdas, carga na entrada do servidor e consumo de energia (taxa de uso do rádio). Os intervalos de confiança mostrados nos gráficos são de 95%, representados por barras verticais. O número de perdas (L) é calculado somando-se o número de retransmissões de pacotes CoAP. Para obter a taxa de perdas (l), divide-se o número de perdas (L) pelo número total de pacotes CoAP ($N_{pacotes}$), incluindo as retransmissões. A carga no servidor é a quantidade de dados recebidos e processados pelo servidor dividida por um intervalo de tempo (Δt), que, neste trabalho, é o tempo decorrido entre o primeiro pacote CoAP gerado na rede e o último. Finalmente, a taxa de uso do rádio é calculada dividindo-se o tempo total de uso do rádio pelo tempo total de operação do dispositivo.

5. Análise dos Resultados

Esta seção apresenta os resultados dos experimentos utilizando o mecanismo de comutação de servidores proposto. A parte relativa à obtenção da lista de servidores não foi simulada, já que o foco é a comutação.

5.1. Desempenho do mecanismo de comutação de servidores

Os testes com o mecanismo de comutação de servidores foram realizados considerando um cenário com um total de 10 clientes e intervalo entre pedidos sob demanda de 5 segundos ($n_c = 10; t_n = 5 s$). Essa configuração corresponde àquela de maior demanda utilizada na análise do CoAP feita em trabalho anterior [Bahia and Campista 2017] e foi empregada neste trabalho para fins de comparação de desempenho do mecanismo de comutação quando se considera servidores sem controle de demanda e com controle de demanda. O mecanismo de controle de demanda proposto em [Bahia and Campista 2017] entra em modo de controle de demanda sempre que o servidor estiver operando em ciclo de trabalho maior que o limiar $Tx_{max} = 0,2\%$. Nesse modo, o servidor deixa de atender clientes sob demanda, enviando apenas notificações a clientes observadores. Após um intervalo de tempo $t_{espera} = 10 s$, o servidor volta a verificar o ciclo de trabalho e assim o faz enquanto o nível de bateria estiver acima de um limiar $Q_{critico}$. Se o nível estiver abaixo de $Q_{critico}$, apenas os clientes sob demanda são atendidos e o processo não se repete mais. O tamanho da lista de servidores variou entre 1 e 4 ($1 \leq n_s \leq 4$). Para cada cenário, o tamanho da lista de observadores por servidor variou entre 0 e 3 ($0 \leq p \leq 3$). Os parâmetros de configuração para a comutação foram o tipo de atendimento preferido (*SERVICE*), a tolerância de atraso para notificações (*tol*) e o tempo de espera para reinício do ciclo de comutação ($t_{reiniciar}$). Os valores dos parâmetros de comutação utilizados nos experimentos foram *SERVICE* = 2 (preferência por servidores com vagas para observadores), *tol* = 6 (tolera esperar o equivalente a 3 períodos de notificação) e $t_{reiniciar} = 20 s$ (espera 20 segundos para reiniciar mecanismo de comutação quando chega ao fim do processo). A configuração utilizada no mecanismo de controle de demanda foi aquela que apresentou maior redução de uso do rádio nos casos testados em [Bahia and Campista 2017].

Carga nos servidores: Avaliando a carga total no conjunto de servidores (Figura 3), observa-se que, quando somente um servidor presta o serviço, há uma forte redução na carga com a ativação do controle de demanda. O grau de redução da carga é menor quanto maior for o tamanho da lista de servidores (n_s) e o número de observadores (p). O aumento do número de observadores diminui o número de pedidos processados pelos servidores e, conseqüentemente, a carga. Dessa forma, quanto mais servidores recebem novos

observadores, menor é a carga total sobre o conjunto de servidores e menor a frequência com que os servidores entram no modo de controle de demanda. Isso é possível graças a ação do mecanismo de comutação, que permite que o cliente obtenha o mesmo serviço de outro servidor. Como observado, o aumento no número de observadores (p) reduz o fluxo de dados na entrada dos servidores, já que o cliente observador não gera requisições adicionais para o recurso em observação. Poder-se-ia concluir que uma boa prática é aumentar o tamanho da lista de observadores ao máximo possível, porém cabe lembrar que cada recurso de um servidor tem a sua lista de observadores e essas listas compartilham o mesmo espaço de memória, altamente restrito. Portanto, no projeto dos servidores CoAP deve-se levar em consideração quais os recursos a serem disponibilizados e o dimensionamento adequado de cada lista de observadores de acordo com a previsão de demanda.

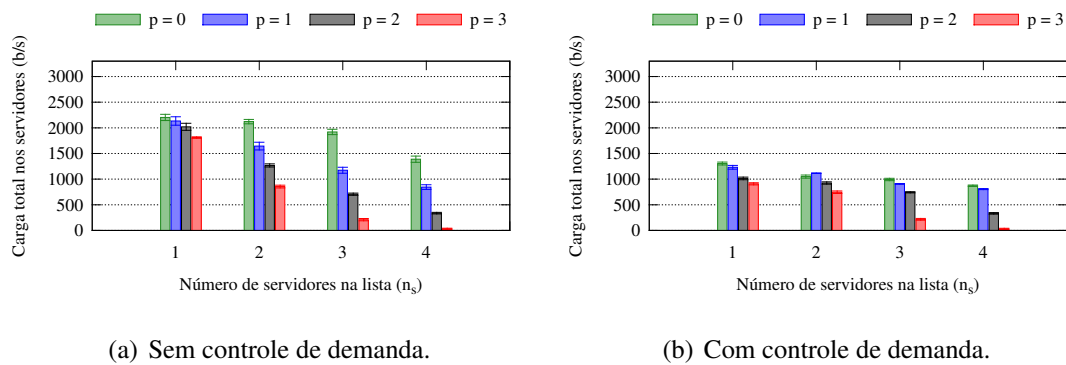


Figura 3. Carga total nos servidores.

Taxa de perdas: A Figura 4(a) mostra que o mecanismo de comutação de servidores não impacta severamente na taxa de perdas da rede, mesmo quando há somente 1 servidor ($s = 1$) sendo consultado. Observa-se também que, quando o cliente pode consultar até 4 servidores ($n_s = 4$) e estes oferecem 3 vagas para observadores ($p = 3$), há um ligeiro aumento na taxa de perdas. Isso acontece pois, nessa configuração, todos os clientes conseguem se tornar observadores (3 servidores com 3 observadores e 1 servidor com 1 observador), reduzindo a quantidade de pedidos gerados. Essa redução faz com que o impacto de uma perda seja maior no valor final da taxa de perdas. Já na Figura 4(b), observa-se que a ativação do controle de demanda pode provocar a perda de mais de 40% dos pacotes quando há somente 1 servidor ($n_s = 1$). Entretanto, o aumento na quantidade de servidores redundantes e de clientes observadores compensa a indisponibilidade temporária de um servidor causada pelo mecanismo de controle de demanda.

Número de pacotes CoAP: O aumento do número de observadores (p) reduz o número de pedidos sob demanda na rede. No caso considerado, o intervalo entre pedidos sob demanda ($t_n = 5$ s) é menor que o intervalo entre notificações ($t_p = 10$ s). Consequentemente, o aumento no número de observadores reduz também o número de pacotes enviados pelos servidores (Figura 5). Nota-se também que o aumento da lista de servidores (n_s) aumenta a oferta de vagas para observadores (cada servidor possui tamanho igual de lista de observadores), reduzindo ainda mais a quantidade de pedidos sob demanda. Quando o servidor possui o mecanismo de controle de demanda (Figura 5(b)), a frequência de ativação do controle de demanda é maior quanto maior for o número de clientes sob demanda ($n = n_c - p$). Por essa razão, o aumento do número de servidores

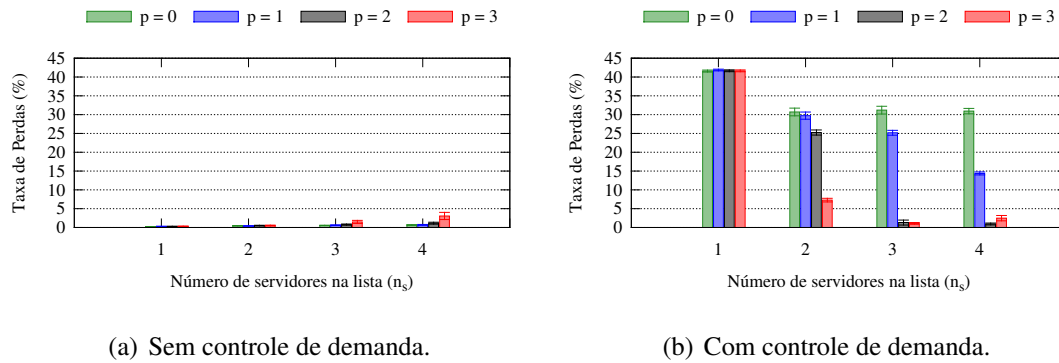


Figura 4. Taxa de perdas.

na lista e do número de clientes observadores reduz o impacto do uso do mecanismo de controle de demanda no atendimento aos clientes.

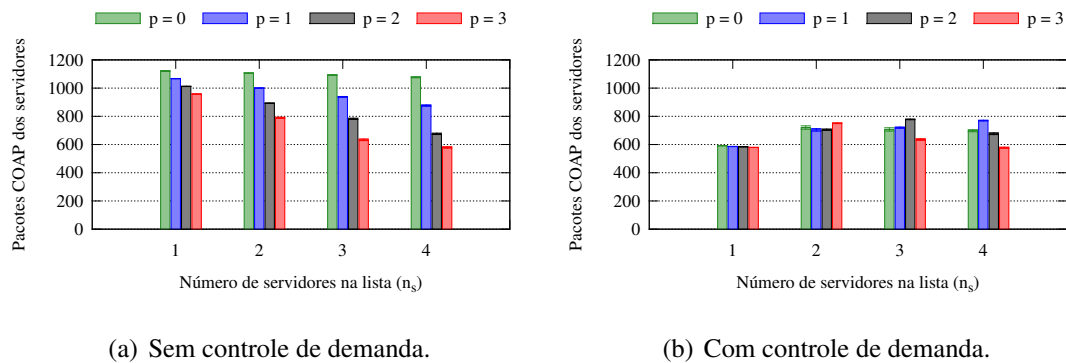


Figura 5. Total de pacotes CoAP dos servidores.

Consumo de energia: Com relação ao consumo médio de energia no conjunto de servidores, a Figura 6 mostra um comportamento semelhante ao observado na Figura 5. Quando o controle de demanda é utilizado, quanto menos servidores, maior a frequência de ativação do controle para regular o consumo e, consequentemente, maior a redução da taxa de uso do rádio. À medida que os clientes comutam entre mais servidores e conseguem se tornar observadores, o ciclo de trabalho médio dos servidores passa a sofrer menos redução com a ativação do controle de demanda. Isso permite que os servidores tenham maior disponibilidade para atender a clientes não observadores.

5.2. Balanceamento de Carga e de Consumo de Energia

Esta seção mostra a distribuição de carga e de consumo de energia entre os servidores diante da aplicação do mecanismo proposto.

A lista de servidores de cada cliente possui uma ordem de prioridade. Como a lista de servidores e a configuração utilizada no mecanismo de comutação de servidores foi a mesma para todos os clientes, o primeiro servidor da lista foi o mais demandado. Através da comutação de servidores, é possível distribuir melhor a demanda e o consumo de energia entre os servidores. Por exemplo, se o cliente monitorar o nível de carga na bateria dos servidores, ele pode comutar para outro servidor depois de consumir certa

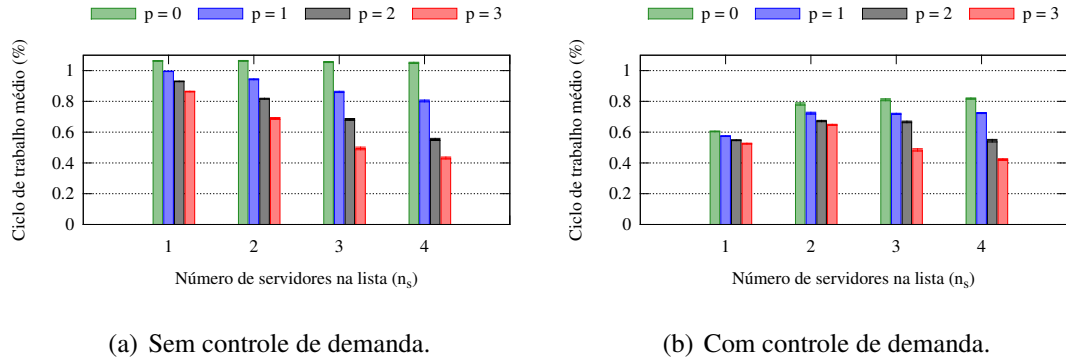


Figura 6. Ciclo de trabalho médio do rádio.

quantidade de energia do servidor atual. Para avaliar o grau de distribuição de carga e de consumo de energia entre os servidores, ou seja, o índice de equidade, utilizou-se o índice de Jain, calculado como $f(x_1, x_2, \dots, x_n) = (\sum_{i=1}^N x_i)^2 / (n \sum_{i=1}^N x_i^2)$. O índice de Jain pode assumir valores entre 0 e 1, sendo que quanto mais próximo de 1, melhor é o balanceamento entre os servidores, ou seja, melhor o índice de equidade.

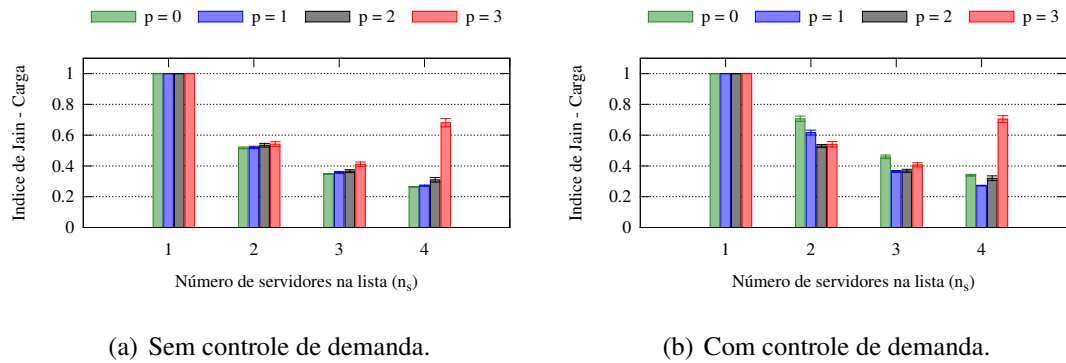


Figura 7. Índice de Jain da carga nos servidores.

Carga nos servidores: A Figura 7 mostra como o índice de Jain calculado para a carga varia com o tamanho da lista de servidores (n_s) e com o número de clientes observadores (p). Observa-se que, à medida que o tamanho da lista de servidores aumenta, o valor do índice de Jain decresce. Isso acontece porque, à medida que os clientes comutam para outros servidores e se tornam observadores, apenas parte do fluxo de dados é redistribuído. O primeiro servidor da lista absorve, num primeiro momento, todos os pedidos dos clientes sob demanda, exceto quando o servidor ativa o controle de demanda, forçando o cliente a comutar para outro servidor. Dessa forma, o aumento na quantidade de vagas para observadores ajuda a melhorar a distribuição de carga e, quando todos os clientes se tornam observadores ($n_s = 4; p = 3$), obtém-se um índice de equidade próximo de 0,7. Essa distribuição promove um balanceamento de carga entre os servidores, favorecendo a escalabilidade da rede. Note que a ativação do controle de demanda resulta num valor do índice de Jain ainda maior, pois o controle de demanda força a comutação de servidores, promovendo maior equidade em termos de carga.

Consumo de energia nos servidores: Utilizando o índice de Jain para o consumo de energia, observa-se na Figura 8 que, quanto mais clientes se tornam observadores, me-

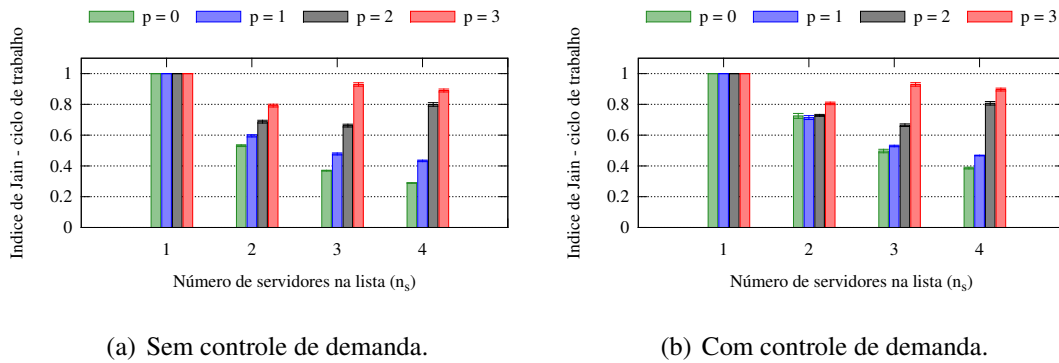


Figura 8. Índice de Jain do ciclo de trabalho dos servidores.

lhor é a distribuição do consumo de energia. Esse resultado evidencia que a comutação de servidores pode promover um balanceamento de consumo, obtendo-se um índice de equidade próximo de 0,9 quando há 3 ou mais servidores com 3 vagas para observadores cada um ($n_s \geq 3; p = 3$). O controle de demanda limita a taxa de uso do rádio do servidor e, apesar do servidor ficar indisponível para clientes não observadores durante a ativação do controle, esses clientes podem garantir a obtenção do serviço comutando para outro servidor redundante. Os resultados mostram que o mecanismo proposto aumenta a confiabilidade dos serviços de IoT ao distribuir as demandas. Consequentemente, o consumo de energia torna-se balanceado entre os servidores, aumentando a disponibilidade dos serviços na rede.

6. Conclusões e Trabalhos Futuros

Este trabalho propõe um novo mecanismo de seleção e comutação de servidores para clientes CoAP. O mecanismo proposto melhora o desempenho e a escalabilidade dos servidores CoAP, distribuindo e aumentando a disponibilidade dos serviços. Até onde se sabe, este é o primeiro trabalho a propor um mecanismo que permita a obtenção de serviços no próprio cliente CoAP através da comutação de servidores. Os resultados mostraram que quanto maior a lista de servidores e o número de vagas para observadores, menor é a carga nos servidores, tendo em vista a distribuição do fluxo de dados entre os servidores. O aumento na taxa de perdas promovido pela indisponibilidade de um servidor é reduzido à medida que a lista de servidores e a quantidade de observadores aumentam, mostrando que o mecanismo de comutação consegue compensar a indisponibilidade de servidores. Observa-se ainda que a funcionalidade de observação e a comutação de servidores promovem a redução, bem como o balanceamento do consumo energético.

Como trabalhos futuros, sugere-se implementar a proposta do mecanismo de obtenção da lista de servidores, ampliar o mecanismo proposto para interagir com *proxies* ou servidores auxiliares, visando o aumento na quantidade de observadores, e validar os resultados simulados com dispositivos reais.

Referências

Afzal, B., Alvi, S. A., Shah, G. A., and Mahmood, W. (2017). Energy efficient context aware traffic scheduling for iot applications. *Ad Hoc Networks*, 62:101 – 115.

- Al-Fuqaha, A., Guizani, M., Mohammadi, M., Aledhari, M., and Ayyash, M. (2015). Internet of Things: A Survey on Enabling Technologies, Protocols, and Applications. *IEEE Communications Surveys Tutorials*, 17(4):2347–2376.
- Bahia, J. G. and Campista, M. E. M. (2017). Um Mecanismo de Controle de Demanda no Provedimento de Serviços de IoT Usando CoAP. In *XXII Workshop de Gerência e Operação de Redes e Serviços (WGRS'2017)*, pages 123–136, Belem, Brasil.
- Banks, A. and Gupta, R. (2014). MQTT Version 3.1.1. *OASIS Standard*, pages 1–81.
- Barbato, A., Barrano, M., Capone, A., and Figiani, N. (2013). Resource Oriented and Energy Efficient Routing Protocol for IPv6 Wireless Sensor Networks. In *Online Conference on Green Communications (GreenCom), 2013 IEEE*, pages 163–168, Piscataway, USA.
- Cisco (2017). Cisco Visual Networking Index: Global Mobile Data Traffic Forecast. [Online: 18-December-2017].
- FIWARE (2011). Context Broker (Orion). [Online: 20-January-2018].
- Granjal, J., Monteiro, E., and Sa Silva, J. (2015). Security for the Internet of Things: A Survey of Existing Protocols and Open Research Issues. *Communications Surveys Tutorials, IEEE*, 17(3):1294–1312.
- Group, M. M. (2016). Internet World Stats. [Online: 18-November-2016].
- Hartke, K. (2014). Observing Resources in CoAP. *Internet Engineering Task Force (IETF)*, RFC 7641.
- Iova, O., Theoleyre, F., and Noel, T. (2014). Improving the Network Lifetime with Energy-Balancing Routing: Application to RPL. In *Wireless and Mobile Networking Conference (WMNC), 2014 7th IFIP*, pages 1–8, Vilamoura, Portugal.
- Jan, M., Nanda, P., He, X., Tan, Z., and Liu, R. P. (2014). A Robust Authentication Scheme for Observing Resources in the Internet of Things Environment. In *Trust, Security and Privacy in Computing and Communications (TrustCom), 2014 IEEE 13th International Conference on*, pages 205–211, Beijing, China.
- Jeyaraman, R. and Telfer, A. (2012). AMQP Version 1.0. *OASIS Standard*.
- Ludovici, A., Garcia, E., Gimeno, X., and Auge, A. C. (2012). Adding QoS support for Timeliness to the Observe Extension of CoAP. In *2012 IEEE 8th International Conference on Wireless and Mobile Computing, Networking and Communications (WiMob)*, pages 195–202, Barcelona, Spain.
- Palattella, M., Accettura, N., Vilajosana, X., Watteyne, T., Grieco, L., Boggia, G., and Dohler, M. (2013). Standardized Protocol Stack for the Internet of (Important) Things. *Communications Surveys Tutorials, IEEE*, 15(3):1389–1406.
- Shelby, Z., Hartke, K., and Bormann, C. (2014). The Constrained Application Protocol (CoAP). *Internet Engineering Task Force (IETF)*, RFC 7252.
- Thingsquare (2016). Contiki: The Open Source OS for the Internet of Things. [Online: 14-November-2016].

Análise de Algoritmos de Clusterização para Experimentos Randomizados em Redes Sociais de Larga Escala*

Bruno Demattos Nogueira, Francisco Galuppo Azevedo,
Fabricio Murai, Ana Paula Couto da Silva

¹Departamento de Ciência da Computação
Universidade Federal de Minas Gerais (UFMG) – Belo Horizonte, MG – Brazil

{bruno.demattos, franciscogaluppo, murai, ana.coutosilva}@dcc.ufmg.br

Abstract. *Large companies conduct A/B tests to estimate the effect of changes in their websites. In these tests, users are randomly redirected to one of two versions of the site. However, in social networks, users that access different versions can influence each other if they are linked, making estimation more difficult. To minimize this interference, graph partitioning algorithms were proposed to find clusters of well-connected users (e.g. ϵ -net and FENNEL). All users within a cluster are redirected to the same version. In this work, we propose a parallel variant of ϵ -net and a new algorithm dubbed NoMAS, inspired on FENNEL. We present a theoretical analysis of the proposed algorithms' scalability complemented by empirical results on the estimation accuracy.*

Resumo. *Grandes empresas realizam testes A/B para estimar o efeito de mudanças nos seus websites. Nestes testes, usuários são redirecionados aleatoriamente para uma de duas versões do site. Porém, em redes sociais, usuários que acessam diferentes versões podem influenciar uns aos outros se estiverem relacionados, dificultando a estimação. Para minimizar esta interferência, foram propostos algoritmos para particionar a rede em clusters de usuários bem conectados (ϵ -net e FENNEL). Todos os usuários dentro de um cluster são redirecionados para uma mesma versão. Neste trabalho, propomos uma versão paralela do ϵ -net e um novo algoritmo chamado NoMAS, inspirado no FENNEL. Apresentamos uma análise teórica da escalabilidade dos algoritmos complementada por resultados empíricos sobre a acurácia da estimação.*

1. Introdução

Conduzir pesquisas de experiência dos usuários é peça-chave na tomada de decisões para grandes companhias que oferecem serviços na Internet (como Amazon, e-Bay, Facebook, Google e LinkedIn) [Kohavi et al. 2013]. Desde simples mudanças no *layout* das páginas até alterações substanciais em algoritmos de ranqueamento de conteúdo devem ser avaliadas de forma objetiva para determinar se estas são de fato melhores que o *status quo*. Algumas métricas relevantes são: tempo de carregamento das páginas, nível de engajamento dos usuários, número médio de cliques por usuário, lucro obtido através de anúncios etc.

*The authors' work has been partially funded by the EUBra-BIGSEA project by the European Commission under the Cooperation Programme (MCTI/RNP 3rd Coordinated Call), Horizon 2020 grant agreement 690116, CAPES, CNPq and FAPEMIG.

A avaliação do efeito de mudanças em um serviço oferecido pela Internet precisa ser feita através de experimentos controlados. Em particular, é importante considerar o mesmo período de medição ao comparar estatísticas obtidas para os dois tratamentos, pois podem existir fatores não-controlados relacionados ao tempo (ex.: o site aparece em manchete de jornal entre experimentos). Além disso, as populações expostas a cada uma das versões precisam ser equivalentes. Por exemplo, uma mudança bem recebida por um grupo de usuários que trabalham com TI pode não ser positiva para o restante.

Uma técnica clássica para realizar experimentos controlados são os testes A/B, que consistem em experimentos randomizados com duas variantes de um tratamento [Kohavi et al. 2009]. Em testes clínicos, parte dos indivíduos são selecionados aleatoriamente para receber um placebo, enquanto o restante recebe o medicamento cujo efeito deseja-se avaliar. No contexto da Internet, usuários que visitam um website são aleatoriamente redirecionados à versão atual ou à versão sendo testada.

Os testes A/B têm como premissa a *Stable Unit Treatment Value Assumption* (SUTVA), que assume que o comportamento de um indivíduo irá depender apenas do tratamento que lhe foi atribuído, ou seja, não há interferência entre os indivíduos. Enquanto esta suposição é válida para testes clínicos, experimentos em redes sociais normalmente a violam (efeito conhecido como *spillover*) [Xu et al. 2015]. Considere, por exemplo, um experimento para avaliar a criação de um chat para troca de mensagens entre usuários. Mesmo que as diferenças entre os grupos de controle e de tratamento não sejam significativas quando apenas 20% dos usuários tem acesso à funcionalidade, não podemos afirmar que essa conclusão continuaria válida caso 100% dos usuários pudessem usar o chat.

Recentemente, muitas abordagens foram propostas para estimar o efeito médio do tratamento (ATE) quando a SUTVA não é válida [Middleton and Aronow 2011, Backstrom and Kleinberg 2011, Katzir et al. 2012, Ugander et al. 2013, Gui et al. 2015]. A maioria delas utilizam algoritmos de clusterização para encontrar pequenos conjuntos de usuários bem conectados entre si. Nesse caso, a randomização é feita a nível de cluster: todos os usuários do mesmo cluster recebem o mesmo tratamento. O objetivo é emular dentro dos clusters os cenários em que toda a população está no controle ou no tratamento.

Algoritmos de clusterização para experimentos randomizados em redes sociais devem atender dois requisitos principais. Primeiramente, devem ser escaláveis. Uma forma de alcançar este objetivo é através da paralelização, utilizando, por exemplo, frameworks baseados em Map-Reduce como Hadoop ou Spark. Apesar disso, não existe na literatura uma descrição paralela do ϵ -net, um dos algoritmos que compõem o estado-da-arte nesta área [Ugander et al. 2013]. Como segundo requisito, devem resultar em pequenos erros na estimação do ATE. Para controlar o erro causado por diferenças nos tamanhos dos clusters, algumas abordagens, tais como a proposta em [Saveski et al. 2017], usam algoritmos que resultam em clusters de tamanhos semelhantes [Nishimura and Ugander 2013]. Esta restrição pode acabar aumentando o número de arestas entre clusters. É crucial entender em que cenários é vantajoso o uso de tais algoritmos.

Este trabalho apresenta três contribuições principais. A primeira é a proposta de uma versão paralela do algoritmo de clusterização ϵ -net. Como segunda contribuição, propomos o NoMAS, um algoritmo que lê o grafo que representa a rede como um *stream* de dados e, analisando cada nó uma única vez, determina um particionamento em clusters de

tamanho aproximadamente igual. Finalmente, avaliamos os algoritmos propostos quanto à escalabilidade e impacto na erros de estimação do ATE. O estudo de escalabilidade consiste na derivação de resultados analíticos complementados por resultados empíricos. O impacto dos algoritmos na qualidade das estimativas é feito através de simulações de modelos de comportamento de usuário sobre diferentes redes¹.

2. Trabalhos Relacionados

O estudo da resposta a um tratamento na presença de interações sociais é formalizado em [Manski 2013]. O autor considera a existência de pequenos grupos de indivíduos conectados, cujas respostas são fortemente dependentes. Neste caso, a randomização completa (i.e., a nível individual), não tem o poder estatístico para identificar o efeito do tratamento. Uma alternativa é particionar a população usando técnicas de detecção de comunidades. Este é um problema bem estudado, para o qual uma miríade de técnicas já foram propostas [Fortunato 2010]. Recentemente, os autores de [Eckles et al. 2014] mostraram que técnicas baseadas na otimização de uma métrica global, como a modularidade de Newman [Newman 2006], não são capazes de distinguir comunidades pequenas. Por esta razão, os autores argumentam que elas não são adequadas para testes A/B em redes.

Em [Backstrom and Kleinberg 2011] os autores assumem que um indivíduo é considerado “exposto” ao tratamento se ele e pelo menos τ de seus vizinhos recebem o tratamento. Para aumentar o tamanho da amostra efetiva dada uma porcentagem fixa da população a receber o tratamento, o método proposto realiza, a partir de um conjunto inicial de nodos, passeios aleatórios que tendem a fechar triângulos. Os nodos iniciais e visitados formam o grupo de tratamento. Infelizmente, este método não é escalável. Um abordagem similar é proposta em [Katzir et al. 2012], cuja diferença é permitir que o tratamento seja atribuído probabilisticamente. Neste trabalho, assim como outros em que o tratamento de um nodo é uma variável aleatória [Aronow and Samii 2012], a estimação é baseada nos estimadores Horvitz-Thompson e/ou Hajek.

Em [Gui et al. 2015], os autores propõem um modelo linear para a resposta de um nodo em função do tratamento dele e de seus vizinhos, permitindo o uso de todas as amostras. Para reduzir o impacto de supor incorretamente tal relação linear, os autores usam randomização a nível de cluster. Em particular, eles propõem um algoritmo que resulta em clusters de mesmo tamanho. A ideia do algoritmo é analisar cada par de vértices e trocar a etiqueta deles se houver uma redução no número de arestas entre partições. Portanto, devem ser analisados $|\mathcal{V}|^2$ pares e, para cada um deles, deve se descobrir quantos vizinhos há em cada partição para quantificar o ganho (ou perda), o que inviabiliza a aplicação desse algoritmo em redes com muitos nodos, como as descritas na introdução.

O algoritmo ϵ -net proposto em [Ugander et al. 2013] encontra um particionamento da rede onde cada nodo está a menos que ϵ saltos do centro de sua partição. Além disso, os centros das partições são nodos que distam pelo menos ϵ saltos entre si. Todos os nodos do grafo são inicializados como não-marcados. Um nodo v é escolhido uniformemente dentre os não-marcados para ser o centro de uma partição. Todos os nodos a uma distância menor que ϵ de v são marcados. Os dois últimos passos são repetidos até que todos os nodos sejam marcados. Ao final do processo, cada nodo u é associado à partição cujo centro está mais próximo de u (empates são resolvidos arbitrariamente). Apesar da

¹As implementações estão disponíveis em github.com/settifoglio/Redes

simplicidade, a descrição do ϵ -net é inerentemente sequencial, inviabilizando seu uso em redes de larga escala.

Em [Nishimura and Ugander 2013], os autores propõem o FENNEL, um algoritmo para particionar um grafo lido como um *streaming* de dados. Todas as partições são inicializadas como conjuntos vazios. Ao analisar um nodo v , o método calcula uma pontuação associada a atribuir v a cada uma das partições e escolhe aquela que tem a maior pontuação. Essa pontuação é calculada descontando, do número de vizinhos do nodo que estão na partição, um termo que cresce de acordo com o tamanho da comunidade. Este desconto ou penalidade tem como objetivo fazer com que as comunidades tenham tamanhos semelhantes. Mais precisamente, o nodo v é colocado na partição $\arg \max_{i \in \{1, 2, \dots, \kappa\}} |P_i^t \cap N(v)| - \alpha \frac{\gamma}{2} (|P_i^t|)^{\gamma-1}$, onde P_i^t é o subconjunto de nodos no cluster i no passo t , $N(v)$ é o conjunto dos vizinhos do vértice v e α e γ são parâmetros dados como entrada. O único caso estudado é aquele em que $\gamma = 2$ (i.e., termo de penalidade é reduzido a $\alpha |P_i^t|$). Este algoritmo é utilizado no método proposto em [Saveski et al. 2017] para testar a hipótese de que a SUTVA é válida. Embora outro algoritmo de particionamento pudesse ter sido utilizado, o FENNEL é um dos mais escaláveis.

3. Algoritmos de clusterização propostos

A seguir descrevemos os dois algoritmos de clusterização propostos neste trabalho. Estes algoritmos visam o particionamento de uma rede para a execução de experimentos randomizados, não sendo, portanto, algoritmos gerais para detecção de comunidades.

3.1. Algoritmo ϵ -net paralelo

Nesta seção propomos uma implementação paralela escalável do ϵ -net baseada no paradigma Map-Reduce, que é o bloco fundamental de frameworks distribuídos usados em redes sociais online de escala global tais como o Facebook e o LinkedIn. O Giraph e o Spark-GraphX são os exemplos mais proeminentes dos frameworks para executar operações em grafos. Esses frameworks mapeiam nodos, arestas, além de dados associados aos nodos e arestas para hosts de um cluster de computação. Além dos *joins* e filtros típicos das aplicações baseadas em Map-Reduce, esses frameworks permitem a execução de operações baseadas em *message passing* síncrono através das arestas do grafo.

A utilização de memória e o volume de dados transmitidos são restrições fundamentais nesses frameworks. Por essa razão, no algoritmo proposto, apenas um par (*etiqueta, distancia*) será armazenado por nodo, e as mensagens transmitidas tem essa mesma estrutura. Ao final de cada passo do *message passing*, uma operação de **reduce** é usada para sumarizar as mensagens recebidas. Este sumário poderá ser usado para atualizar a informação contida no nodo.

A chave para paralelizar o ϵ -net é constatar que, uma vez que os **centros dos clusters** sejam determinados, existe uma operação de *message passing* em tempo constante que associa os nodos do grafo aos seus respectivos clusters. Portanto, iremos focar em paralelizar a busca por centros dos clusters. Em particular, é crucial verificar de maneira eficiente quais nodos de uma dada lista de nodos são clusters válidos, i.e., consistentes com o algoritmo sequencial. Por exemplo, na Figura 1(a), consideramos a lista formada pelos nodos C, E, A e D. No algoritmo sequencial com $\epsilon = 3$, a escolha de C como centro de partição inviabiliza E como possível centro. Note que não é preciso determinar de uma

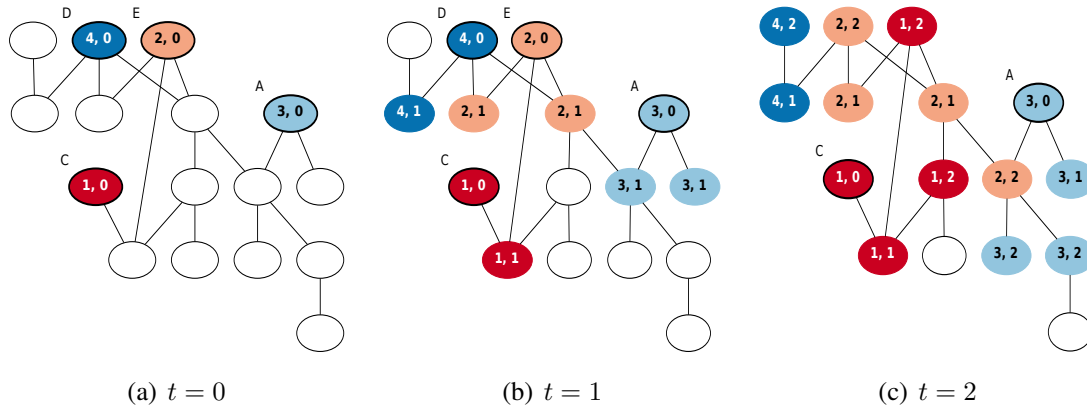


Figura 1. Exemplo de chamada à PROPAGAEtiquetas com `field=etiqueta` e $\epsilon = 3$. As sementes são C, E, A e D. No interior de cada nodo, x, y representa respectivamente a etiqueta sendo propagada e a distância em relação ao seed. No passo $t = 2$, observamos que: C é um centro válido, E foi invalidado por C, A é um centro válido e D deve ser testado novamente na próxima iteração, pois foi invalidado pela semente inválida E.

única vez uma lista de centros cujos discos irão cobrir o grafo: podemos encontrar mais centros repetidamente até que o grafo esteja completamente coberto.

Denominamos **sementes** os nodos que compõem uma lista s , a partir da qual deseja-se extrair um suconjunto $\mathcal{C}$ de centros de clusters válidos. Para isso, estabelecemos uma ordem de precedência entre as sementes usando as posições $I(s)$ em que as sementes $s \in s$ aparecem (posição anterior indica maior precedência). Desta forma, podemos propagar pelo grafo etiquetas que correspondem aos índices $I(s)$, dentro de um raio $\epsilon - 1$. Neste caso, o sumário das mensagens recebidas por um nodo é a mensagem cuja etiqueta é mínima. O conteúdo do sumário irá substituir os dados associados ao nodo se a etiqueta presente no sumário for menor. Na Figura 1(b), observamos que, após o primeiro passo da propagação, os vizinhos comuns a D e E mantiveram apenas a informação sobre E.

Ao final do processo de propagação, iremos analisar as sementes em ordem de precedência. As sementes válidas são aquelas que mantiveram a etiqueta inicial; as inválidas, possuem uma etiqueta que corresponde a uma semente válida. É possível ainda que uma semente X tenha sido invalidada por uma semente Y que foi, por sua vez, invalidada por uma semente Z . No exemplo da Figura 1(c), $X = D$, $Y = E$, $Z = C$. Ao invés de concluir que X é válida (dado que Y não é), optamos por interromper a análise da lista de sementes, pois é possível que existam outras sementes que teriam sido invalidadas por X , caso Y não estivesse em s .

O Algoritmo 1 descreve o algoritmo ϵ -net paralelo sobre um grafo $\mathcal{G} = (\mathcal{V}, \mathcal{E})$. Na linha 1, uma lista $\mathbf{v}$ é criada embaralhando-se os índices dos nodos em $\mathcal{V}$. Em seguida, inicializa-se a variável $\mathcal{D}$, que irá mapear os nodos a pares cluster-distância. Os clusters serão identificados pelos índices dos seus centros. O laço das linhas 4-18 é repetido até que todos os nodos tenham sido mapeados. Na linha 5, a função OBTEMSEMENTES varre a lista $\mathbf{v}$ e retona a lista s contendo os k primeiros nodos ainda não-mapeados (i.e., $v : \mathcal{D}(v) = (\text{NULL}, \epsilon)$). Em seguida, inicializa-se a variável I , que mapeia os nodos de s aos índices $1, \dots, |s|$. Esses índices são usados para criar $\mathcal{G}'$, uma cópia de $\mathcal{G}$, usada

Algoritmo 1: ϵ -NET PARALELO encontra um particionamento do grafo

Input: grafo $\mathcal{G} = (\mathcal{V}, \mathcal{E})$, distância ϵ , grau de paralelismo k
Output: mapa $\mathcal{D}$, onde $\mathcal{D}(v) = (\text{etiqueta do cluster ao qual } v \text{ pertence, distância ao centro do cluster})$ para todo $v \in \mathcal{V}$

```

1  $\mathbf{v} \leftarrow \text{EMBARALHAR}(\mathcal{V})$ 
2  $\mathcal{G} \leftarrow (\mathcal{V}, \mathcal{E}, \mathcal{D})$  onde  $\mathcal{D}(v) = (\text{NULL}, \epsilon)$  para todo  $v \in \mathcal{V}$ 
3  $n \leftarrow 0$  // número de clusters encontrados
4 while existe  $v \in \mathcal{V}$  tal que  $\mathcal{D}(v) = (\text{NULL}, \epsilon)$  do
5    $\mathbf{s} \leftarrow \text{OBTEMSEMENTES}(\mathbf{v}, \mathcal{D}, k)$ 
6   Inicialize  $I(s_i) = n + i$  para  $\mathbf{s} = (s_1, \dots, s_{|\mathbf{s}|})$ 
7    $\mathcal{G}' \leftarrow (\mathcal{V}, \mathcal{E}, \mathcal{D}')$  onde  $\mathcal{D}'(v) = \begin{cases} (I(v), 0.0) & \text{se } v \in \mathbf{s} \\ (\text{NULL}, \epsilon) & \text{caso contrário} \end{cases}$ 
8    $\mathcal{D}'' \leftarrow \text{PROPAGAETIQUETAS}(\mathcal{G}', \epsilon, \text{etiqueta})$  // passo paralelizado
9    $\mathcal{C} \leftarrow \emptyset$  // inicializa lista de centros
10  for  $s \in \mathbf{s}$  do
11    if  $\mathcal{D}'(s) = \mathcal{D}''(s)$  then
12       $\mathcal{C} \leftarrow \mathcal{C} + \{s\}$ 
13    else if  $s[\mathcal{D}''(s).first] \notin \mathcal{C}$  then
14      break // foi invalidada por outra inválida
15   $n \leftarrow n + |\mathcal{C}|$ 
16   $\mathcal{G}'' \leftarrow (\mathcal{V}, \mathcal{E}, \mathcal{D}'')$  onde  $\mathcal{D}''(v) = \begin{cases} (v, 0.0) & \text{se } v \in \mathcal{C} \\ (\text{NULL}, \epsilon) & \text{caso contrário} \end{cases}$ 
17   $\mathcal{D}''' \leftarrow \text{PROPAGAETIQUETAS}(\mathcal{G}'', \epsilon, \text{distancia})$  // passo paralelizado
18   $\mathcal{D}(v) = \begin{cases} \mathcal{D}'''(v) & \text{se } \mathcal{D}'''(v).second < \mathcal{D}(v).second \\ \mathcal{D}(v) & \text{caso contrário} \end{cases}$ 
19 return  $\mathcal{D}$ 

```

como entrada para a primeira chamada ao Procedimento PROPAGAETIQUETAS. Este procedimento retorna $\mathcal{D}''$, o mapeamento obtido ao se propagar a informação em $\mathcal{D}'$ por uma distância $\epsilon - 1$, repassando-se apenas as mensagens cuja *etiqueta* (índice do cluster) é mínima. O laço das linhas 10-14 calcula $\mathcal{C}$, o subconjunto de sementes de $\mathbf{s}$ que serão os centros dos novos clusters. Caso a etiqueta associada à semente s em $\mathcal{D}''$ corresponda a uma semente que não está em $\mathcal{C}$ (i.e., uma semente inválida), o laço é interrompido na linha 14. Na linha 16, um novo grafo é criado para a propagação dos índices dos novos clusters, que se dá na linha 17. Desta vez, são propagadas as etiquetas dos centros encontrados até o momento, repassando-se apenas as mensagens cuja *distancia* (até o cluster) é mínima. Por fim, atualiza-se o mapeamento de nodos aos clusters na linha 18.

O Procedimento *propagaEtiquetas* é baseado no paradigma *message passing* síncrono. As linhas 1 e 2 indicam que apenas as sementes ou centros irão trocar mensagens no passo $t = 0$. As linhas 3-8 descrevem como as mensagens recebidas serão processadas, gerando, potencialmente, novas mensagens a serem tratadas no passo seguinte. O processo termina quando nenhuma mensagem é trocada.

3.2. NoMAS: novo algoritmo de clusterização baseado em streaming

O NoMAS (Normalized Intersection Minimization Algorithm for Graph Streams) é o segundo algoritmo de clusterização proposto neste trabalho. Sua principal vantagem é

Procedimento propagaEtiqetas

Input: grafo $\mathcal{G} = (\mathcal{V}, \mathcal{E}, \mathcal{D})$, distância ϵ , campo $field \in \{etiqueta, distancia\}$ usado para sumarizar mensagens

Output: mapa $\mathcal{D}$

- 1 **if** $\mathcal{D}(v).second = 0.0$ **then**
- 2 | Envie $m = (\mathcal{D}(v).first, 1.0)$ a todos os vizinhos em $\mathcal{G}$
- 3 Seja $\mathcal{M}(v)$ o conjunto de mensagens recebidas por v no passo anterior
- 4 **if** $field = etiqueta$ **then** $m^* \leftarrow \arg \min_{m \in \mathcal{M}(v)} \{m.first\}$
- 5 **else** $m^* \leftarrow \arg \min_{m \in \mathcal{M}(v)} \{m.second\}$
- 6 **if** $(field = etiqueta \text{ and } \mathcal{D}(v).first > m^*.first) \text{ or } (field = distancia \text{ and } \mathcal{D}(v).second > m^*.second)$ **then**
- 7 | $\mathcal{D}(v) \leftarrow m^*$
- 8 | Envie $m = (\mathcal{D}(v).first, \mathcal{D}(v).second + 1)$ a todos os vizinhos em $\mathcal{G}$
- 9 Quando nenhuma mensagem for trocada, **return** $\mathcal{D}$

permitir controlar o compromisso entre obter partições de tamanhos semelhantes e a qualidade dessas partições, medida segundo a fração de arestas que ligam nodos em partições diferentes. Este algoritmo é baseado no modelo de *streaming* de grafos e pode ser facilmente paralelizado, características que o tornam altamente escalável.

Diferente do ϵ -net, o número de partições κ obtidas pelo NoMAS é fixo, fornecido como entrada. O modelo de streaming assume que a ordem em que os vértices são analisados é dada por um caminhamento em largura (BFS) em um grafo direcionado², assim como foi feito por [Stanton and Kliot 2012]. Neste modelo, a comunidade a que um nodo pertence deve ser determinada no momento em que a BFS o explora. Apesar do algoritmo ser sequencial, é possível fazer com que ele seja executado em um cluster computacional aplicando-se a mesma ideia usada em [Nishimura and Ugander 2013] para paralelizar os algoritmos FENNEL e LDG.

Assim como o FENNEL, o algoritmo NoMAS determina a qual partição um nodo será associado a partir de uma regra simples. Seja $P_i^t = \{P_1^t, \dots, P_\kappa^t\}$ um particionamento do grafo no passo t , onde P_i^t é o conjunto de vértices do cluster i , $N(u)$ é o conjunto de vizinhos do vértice u e α um parâmetro fornecido com entrada. O vértice u será membro da comunidade cujo índice maximiza a pontuação, i.e.

$$\arg \max_{i \in \{1, 2, \dots, \kappa\}} \frac{\max(1, |P_i^t \cap N(u)|)}{|P_i^t|^\alpha} \quad (1)$$

Em caso de empate entre comunidades, o nodo será colocado naquela de menor tamanho.

O Algoritmo 2 descreve o NoMAS aplicado a um grafo $\mathcal{G} = (\mathcal{V}, \mathcal{E})$. Na linha 1, o vetor $\mathcal{C}$ de tamanho $|\mathcal{V}|$ é inicializado. Nas linhas 2-4, k nodos uniformemente aleatórios são selecionados sem repetição e mapeados às partições 1 a k . No laço das linhas 5-12, é feita uma busca em largura pelo grafo onde cada nodo é avaliado. Na linha 6, é inicializado o vetor **Coefficiente** de tamanho k , que irá armazenar a pontuação de cada comunidade em relação ao vértice u . Nas linhas 7 e 8, incrementa-se a pontuação da comunidade a que cada vizinho de u está associado. A pontuação final de cada comunidade

²Caso o grafo seja não-direcionado, cada aresta original $(u, v) \in \mathcal{E}$ deve ser transformada em duas arestas direcionadas, (u, v) e (v, u) .

Algoritmo 2: NOMAS encontra um particionamento do grafo

Input: grafo $\mathcal{G} = (\mathcal{V}, \mathcal{E})$, distância ϵ , número de partições k
Output: mapa $\mathcal{C}$, onde $\mathcal{C}(v)$ indica o cluster ao qual v pertence, $\forall v \in \mathcal{V}$

```

1  $\mathcal{C} \leftarrow [-1, \dots, -1]$  // inicializa partição de cada nó
2  $\mathbf{v} \leftarrow \text{EMBARALHAR}(\mathcal{V})$ 
3 for  $i \in 1, \dots, k$  do
4    $\mathcal{C}[\mathbf{v}[i]] \leftarrow i$  // associa nó aleatório a cada partição
5 for  $u \in \text{BFS}(\mathcal{G}, \mathbf{v}[1])$  do
6   Coefficiente  $\leftarrow [0, 0, \dots, 0]$  // pontuação de cada partição
7   for  $s \in N(u)$  do
8     Coefficiente $[\mathcal{C}[s]] \leftarrow \text{Coefficiente}[\mathcal{C}[s]] + 1$ 
9   for  $i \in 1, \dots, k$  do
10    Coefficiente $[i] \leftarrow \max(1, \text{Coefficiente}[i])$ 
11    Coefficiente $[i] \leftarrow \text{Coefficiente}[i] / \text{TAMANHOPARTIÇÃO}(i)^\alpha$ 
12    $\mathcal{C}[u] \leftarrow \arg \max_{i \in 1, \dots, k} \text{Coefficiente}[i]$  // Equação (1)
13 return  $\mathcal{C}$ 
```

é calculada nas linhas 9-11. Finalmente, a partição do nodo u é escolhida na linha 12.

O motivo pelo qual o algoritmo funciona é simples e similar ao que explica o êxito do LDG e do FENNEL. Por um lado, as comunidades que possuem mais vizinhos com o vértice sendo analisado terão um numerador maior. Por outro, comunidades que já possuem muitos nodos são penalizadas, o que favorece o aparecimento de comunidades com tamanho similar. O parâmetro α determina se a divisão será feita priorizando comunidades de tamanho similares ou poucas arestas cortadas.

4. Avaliação dos algoritmos de clusterização

Nesta seção iremos avaliar os algoritmos ϵ -net paralelo e NoMAS em relação à escalabilidade e a acurácia na estimação do ATE usando três datasets de redes reais.

4.1. Datasets

Utilizamos três redes sociais (nodos representam indivíduos) disponíveis na coleção de datasets SNAP, de Stanford³. A Tabela 1 mostra as estatísticas básicas delas:

- **Bitcoin OTC:** Rede de pessoas que fizeram transações utilizando o Bitcoin. Uma aresta direcionada (i, j) indica que i confia em j (*who-trusts-whom*).
- **Enron Email:** Rede formada a partir de cerca 500 mil emails trocados dentro da companhia Enron. Nodos são endereços de email e arestas não-direcionadas (i, j) indicam que pelo menos um email foi enviado de i para j ou de j para i .
- **Wiki-Vote:** Rede que representa a votação entre colaboradores da Wikipedia para a escolha de administradores das páginas. Nodos na rede representam colaboradores e uma aresta direcionada (i, j) indica que o usuário i votou no usuário j .

4.2. Análise de escalabilidade

Nesta seção iremos avaliar a escalabilidade dos algoritmos de clusterização propostos segundo os requisitos de memória, custo de transmissão de mensagens e tempo de execução. Para o ϵ -net, estudamos o caso em que $\epsilon = 3$, pois este é o valor utilizado na literatura.⁴

³<http://snap.stanford.edu/>

⁴Ademais, $\epsilon = 2$ considera apenas vizinhos imediatos e $\epsilon > 3$ tende a gerar clusters de tamanho $O(n)$.

Dataset	Total de Nodos	Total de Arestas
Bitcoin OTC	5.881	35.592
Enron Email	36.692	183.831
Wiki-vote	7.115	103.689

Tabela 1. Total de nodos e arestas das redes em cada dataset.

Algoritmo ϵ -net paralelo com $\epsilon = 3$

Custo de memória. O ϵ -net paralelo é projetado para frameworks distribuídos onde os vértices são mapeados para hosts de um cluster $\mathcal{H}$. Cada host $h \in \mathcal{H}$ é responsável por um subconjunto $\mathcal{V}_h \subset \mathcal{V}$ e pelo conjunto de arestas $\mathcal{E}_h$ adjacentes a vértices em $\mathcal{V}_h$. Para a execução do algoritmo, é necessário armazenar um par (etiqueta,distancia) para cada vértice. Portanto, o requisito mais estrito de memória é dado pelo número de arestas armazenadas em um host. Havendo memória suficiente para carregar o grafo no cluster $\mathcal{H}$, haverá memória suficiente para executar o ϵ -net paralelo.

Custo de transmissão. Derivar resultados analíticos para o custo de transmissão no caso de grafos gerais é muito complicado. Por essa razão, faremos a análise apenas para o caso de grafos regulares, que será complementada com resultados empíricos para os datasets utilizados neste artigo. Suponha um grafo $\mathcal{G} = (\mathcal{V}, \mathcal{E})$ escolhido uniformemente dentre todos os grafos d -regulares com $|\mathcal{V}|$ vértices. Na primeira chamada à PROPAGAE-TIQUETAS, são enviadas mensagens de cada semente $s \in \mathcal{S}$ para seus vizinhos. Seja $\mathcal{N}_i$ a variável aleatória que denota a cardinalidade do conjunto de vizinhos dos nodos em $\mathcal{S}$ quando $|\mathcal{S}| = i$. Em particular, sabemos que $\mathcal{N}_1 = d$ e que $\mathcal{N}_i \leq di$ com probabilidade 1. No caso geral, o valor esperado dessa v.a. é dado pela relação de recorrência homogênea

$$\mathbb{E}[\mathcal{N}_i] = \mathbb{E}[\mathcal{N}_{i-1}] + d - d \frac{\mathbb{E}[\mathcal{N}_{i-1}]}{|\mathcal{V}|}. \quad (2)$$

Note que estamos apenas acrescentando d e descontando os vizinhos da nova semente que são adjacentes aos nodos em $\mathcal{N}_{i-1}$. A solução da recorrência em (2) é dada por

$$\mathbb{E}[\mathcal{N}_i] = |\mathcal{V}| \left(1 - \left(1 - \frac{d}{|\mathcal{V}|} \right)^i \right), \quad (3)$$

Quando $d \ll |\mathcal{V}|$ e $i \ll |\mathcal{V}|$, podemos usar a aproximação de primeira ordem $(1 - d/|\mathcal{V}|)^i \approx 1 - di/|\mathcal{V}|$ e assim obter $\mathbb{E}[\mathcal{N}_i] \approx di$. Essencialmente, esse resultado diz que quando d e $|\mathcal{S}|$ são pequenos, analisar o caso médio é equivalente a analisar o pior caso. Na segunda chamada à PROPAGAE-TIQUETAS, cada um dos $\mathcal{N}_{|\mathcal{S}|}$ nodos irá enviar mensagens para os seus d vizinhos. As mensagens trocadas entre nodos armazenados no mesmo host não consomem recursos da rede de comunicação. Portanto, o custo de transmissão é dado pelo número de mensagens trocadas entre hosts. A probabilidade de dois nodos estarem no mesmo host do cluster $\mathcal{H}$ é $1/|\mathcal{H}|$. Logo, o custo de transmissão por iteração é aproximadamente $d^2|\mathcal{S}|(|\mathcal{H}| - 1)/|\mathcal{H}|$.

Para complementar os resultados teóricos obtidos para o caso de grafos d -regulares, calculamos o número de mensagens trocadas em cada iteração a partir de 1000 execuções do algoritmo sobre os três datasets de redes, quando $|\mathcal{S}| \in \{2, 4, 8\}$. A Figura 2 mostra os boxplots do número de mensagens trocadas nas iterações 1, 2, 5, 10, 20

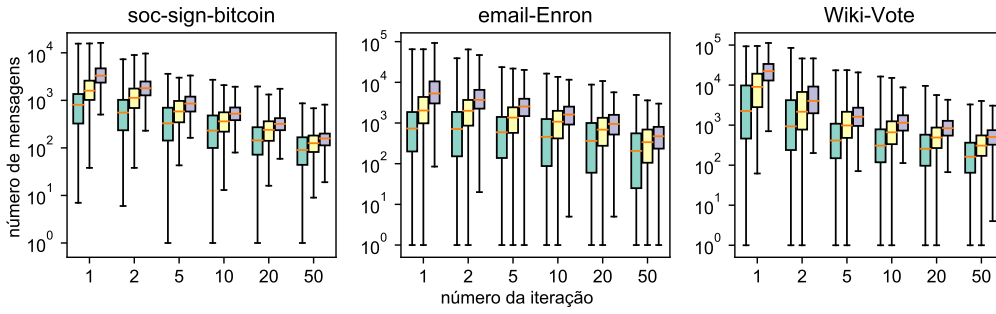


Figura 2. Boxplots do custo de transmissão (número de mensagens) do ϵ -net paralelo para certas iterações, com $|s| = 2$ (verde), 4 (amarelo) e 8 (rosa). A relação entre o número de mensagens e $|s|$ é aproximadamente linear.

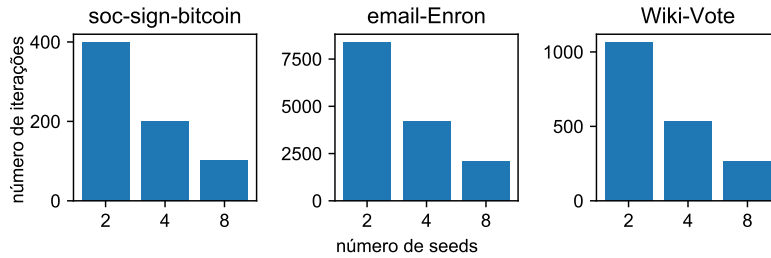


Figura 3. Número médio de iterações do ϵ -net paralelo até a parada, variando-se o número de seeds por iteração $|s| \in \{2, 4, 8\}$.

e 50. Cada boxplot indica, de cima para baixo, máximo, 3º quartil, mediana, 1º quartil e mínimo. Verificamos que a mediana varia de maneira aproximadamente linear com $|s|$. Além disso, o número de mensagens transmitidas diminui com o número de iterações, pois os vértices de maior grau tendem a ser associados às partições primeiro.

Tempo de execução. O número de iterações realizadas até que o algoritmo termine é aproximadamente a razão entre o total de sementes utilizadas e $|s|$. Contudo, não é trivial calcular esse total, mesmo no caso de um grafo d -regular. Assim, incluímos resultados empíricos que relacionam o número de iterações com a escolha de $|s|$. A Figura 3 mostra o número médio de iterações em cada dataset baseado em 1000 execuções. Como o desvio padrão nesses casos foi muito pequeno, não incluímos intervalos de confiança. O número médio de iterações decresce linearmente com o número de seeds.

Algoritmo NoMAS

Custo de memória. Na versão serial do NoMAS o custo de memória consiste no espaço para armazenar o grafo, o tamanho atual de cada partição e, para cada vértice: sua etiqueta e um vetor de tamanho κ contendo o número de vizinhos em cada partição. Em um grafo $\mathcal{G} = (\mathcal{V}, \mathcal{E})$, a complexidade assintótica de espaço é $O(\max\{|\mathcal{V}| + |\mathcal{E}|, \kappa|\mathcal{V}|\})$. Quando o grau médio $\bar{d} > \kappa$, o custo dominante é o custo de se armazenar o grafo. Logo, se a máquina puder comportá-lo, é possível executar o NoMAS.

Já na versão paralela, em um *setup* com $|\mathcal{H}|$ hosts, cada um armazena: (i) a lista de vértices pelos quais ela é responsável, (ii) o conjunto de vizinhos de cada um deles e

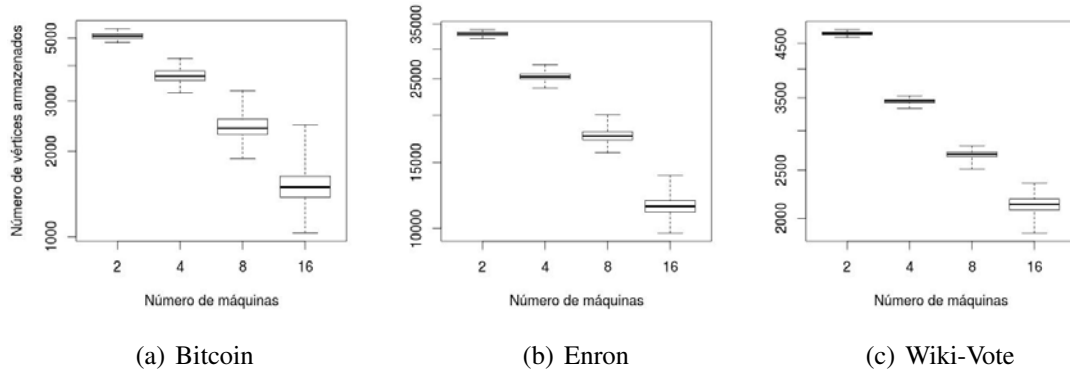


Figura 4. Custo de memória por máquina do NoMAS (soma do número de nodos com o tamanho da lista de vizinhos), variando-se o número de máquinas.

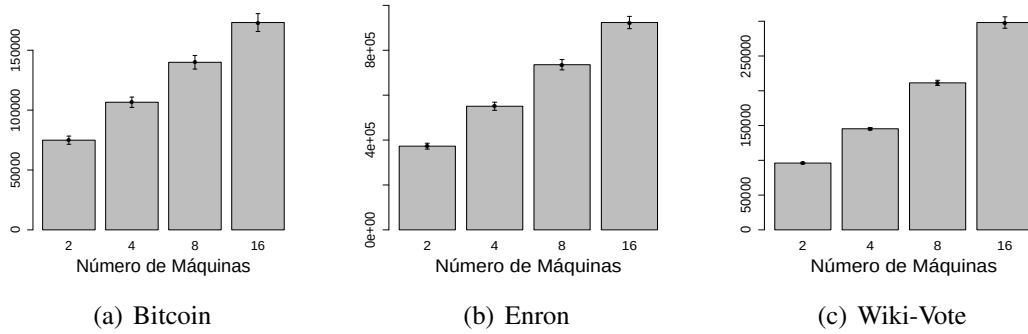


Figura 5. Número total de mensagens trocadas para cada dataset variando-se o número de máquinas no cluster.

(iii) a partição a qual cada um dos vizinhos pertence. Dessa forma, cada máquina guarda uma lista de tamanho $|\mathcal{V}|/|\mathcal{H}|$, que representa os nodos que são gerenciados por ela, além dos vizinhos de cada um desses vértices e suas respectivas partições, que tem tamanho aproximado de $\bar{d} \cdot |\mathcal{V}|/|\mathcal{H}|$.

Custo de transmissão. Assim como no caso do ϵ -net, faremos uma análise para um caso simples e depois apresentaremos resultados empíricos usando grafos reais. Novamente, assumamos um grafo $\mathcal{G} = (\mathcal{V}, \mathcal{E})$ escolhido uniformemente dentre todos os grafos d -regulares com $|\mathcal{V}|$ vértices. Cada um dos hosts do cluster $\mathcal{H}$ será responsável por $|\mathcal{V}|/|\mathcal{H}|$ nodos. No pior caso, em que os vizinhos desses vértices são distintos, serão armazenados os dados de $d \cdot |\mathcal{V}|/|\mathcal{H}|$ vértices, como dito na seção anterior. A cada *stream* do algoritmo, a partição a que pertencem os vizinhos dos nós armazenados em cada máquina deve ser atualizada. Para cada uma, a etiqueta dos seus $|\mathcal{V}|/|\mathcal{H}|$ nodos terá que ser transmitida para até d outros computadores do *cluster*, enquanto ela receberá a informação dos $d \cdot |\mathcal{V}|/|\mathcal{H}|$ vértices vizinhos. Dessa maneira, o custo de transmissão total será $|\mathcal{V}| \cdot d$, no pior caso.

Tempo de execução. A análise do tempo de execução deve levar em conta o caminhoamento feito no grafo para visitar todos os vértices, que é $O(|\mathcal{V}| + |\mathcal{E}|)$ e que, para cada nodo, deve ser analisada a pontuação de cada comunidade, o que custa $O(\kappa)$, onde κ é o número de comunidades. Logo, a complexidade total de tempo é dada por $O(\kappa(|\mathcal{V}| + |\mathcal{E}|))$.

Datasets	Corte			Carga máxima normalizada ρ		
	ϵ -net	FENNEL	NoMAS	ϵ -net	FENNEL	NoMAS
Bitcoin OTC	0.745	0.554	0.580	106.0	1.002	1.020
Enron Email	0.884	0.563	0.400	197.8	1.000	1.000
Wiki-Vote	0.882	0.813	0.798	464.6	1.001	1.071

Tabela 2. Avaliação da qualidade do particionamento em termos de fração de arestas entre comunidades (Corte) e o tamanho normalizado da maior comunidade (ρ). Para ambas as métricas, quanto menor, melhor.

4.3. Avaliação da qualidade do particionamento

Nesta seção avaliamos a qualidade do particionamento encontrado pelos algoritmos ϵ -net paralelo (com $\epsilon = 3$) e NoMAS (com $\kappa = 10$ partições e $\alpha = 2$). Comparamos os resultados com aqueles do FENNEL. O ϵ -net original não foi incluído na comparação, pois sua saída é idêntica a da versão paralela proposta neste artigo. As métricas validadas são as mesmas utilizadas em [Tsourakakis et al. 2014] para a avaliação do FENNEL:

- **Corte:** fração das arestas do grafo que conecta nodos em diferentes partições.
- **Carga máxima normalizada ρ :** razão entre o tamanho da maior partição e $|\mathcal{V}|/\kappa$. Quando $\rho = 1$, todas as partições tem o mesmo tamanho.

A Tabela 2 mostra as médias obtidas para cada métrica para os algoritmos em cada um dos três datasets baseados em 100 execuções. Observa-se que o ϵ -net resultou em valores maiores de **Corte**. Isso acontece porque o número de partições encontradas é uma variável aleatória, tipicamente muito maior que $\kappa = 10$: o número médio de partições encontradas foi 1326,3 para o Bitcoin, 5505,3 para o Enron Email e 4131,6 para o Wiki-Vote. Também conforme esperado, o ϵ -net apresenta variabilidade muito maior no tamanho das partições (i.e., maior ρ) porque não leva esse critério em consideração.

Comparando o NoMAS com o FENNEL, observamos um dataset onde o NoMAS é superior (Enron Email), um dataset onde o FENNEL é superior (Bitcoin OTC), e um dataset onde o NoMAS tem **Corte** menor, mas ao custo de uma variabilidade ρ maior. Em termos quantitativos, o NoMAS chegou a reduzir **Corte** em 29.0% e aumentou-o em 4.7% no pior caso. Já a carga máxima normalizada ρ foi até 7% maior para o NoMAS.

4.4. Acurácia na estimação do ATE

Nesta seção, comparamos os algoritmos propostos com o FENNEL em relação à acurácia na estimação do *average treatment effect* (ATE). Para isto, simulamos a resposta (ou comportamento) Y_i de cada indivíduo i na rede a partir do tratamento $Z_i \in \{0, 1\}$ atribuído por randomização a nível de cluster (i.e., todos os nodos j no mesmo cluster tem mesmo Z_j). O modelo utilizado para geração de respostas é o modelo linear

$$Y_i = \beta_0 + \beta_1 Z_i + \beta_2 \sigma_i + \omega_i, \quad (4)$$

onde $\beta = (\beta_0, \beta_1, \beta_2)$ são parâmetros do modelo, σ_i é a fração de vizinhos j do nodo i com $Z_j = 1$, e $\omega_i \sim \text{Normal}(0, 1)$.

Usamos os cinco vetores de parâmetros β sugeridos em [Azevedo et al. 2018] para contemplar os casos mais representativos. Dado um vetor β e um algoritmo, existe

	Algoritmo	Vetor de parâmetros β e MSE_{Linear}				
		$\beta = (0, 0, 1)$	$\beta = (0, 1, 0, 5)$	$\beta = (0, 1, 0)$	$\beta = (0, 1, 1)$	$\beta = (0, 1, 2)$
bitcoin	FENNEL	0,00189	0,00174	0,00192	0,00194	0,00214
	NoMAS	0,00212	0,00190	0,00200	0,00190	0,00188
	ϵ -net	0,00224	0,00214	0,00204	0,00208	0,00205
Enron	FENNEL	0,00032	0,00034	0,00036	0,00034	0,00033
	NoMAS	0,00030	0,00032	0,00032	0,00031	0,00031
	ϵ -net	0,00027	0,00025	0,00028	0,00027	0,00030
Wiki-Vote	FENNEL	0,00198	0,00181	0,00188	0,00187	0,00186
	NoMAS	0,00187	0,00189	0,00204	0,00208	0,00198
	ϵ -net	0,00201	0,00217	0,00204	0,00203	0,00209

Tabela 3. Avaliação dos algoritmos quanto ao erro de estimação do ATE, medido em termos do MSE. Modelo de resposta linear usado para geração dos dados.

aleatoriedade na geração das partições, no mapeamento de partições para os grupos de controle e tratamento (que resultam no vetor $\mathbf{Z}$), e na geração das respostas (que resultam no vetor $\mathbf{Y}$). Para cada ponto de aleatoriedade realizamos 10 execuções, resultando em 5 vetores $\beta \times 3$ algoritmos $\times 10$ vetores $\mathbf{Z} \times 10$ vetores $\mathbf{Y}$, totalizando 15.000 simulações.

Os resultados são avaliados em termos do erro médio quadrático (MSE), dado por

$$MSE(\text{algoritmo}, \beta) = \frac{1}{100} \sum_{\mathbf{Z}, \mathbf{Y}} (\widehat{ATE}(\mathbf{Z}, \mathbf{Y}) - ATE(\beta))^2, \quad (5)$$

onde $\widehat{ATE}(\mathbf{Z}, \mathbf{Y})$ é o ATE estimado por um modelo de regressão linear para os vetores $\mathbf{Z}$ e $\mathbf{Y}$ produzidos pelo algoritmo e $ATE(\beta)$ é o valor real do ATE dado β . Consulte [Azevedo et al. 2018] para mais informações sobre como calcular estas quantidades.

A Tabela 3 sumariza os resultados obtidos para cada dataset e vetor de parâmetros. Observamos que o NoMAS resulta em menor MSE que o FENNEL em 7 de 15 casos. Em particular, NoMAS supera o FENNEL no dataset Email Enron para todos valores de β testados. É interessante notar que o ϵ -net obtém os melhores resultados naquele dataset.

5. Conclusões

Neste trabalho propusemos e analisamos dois algoritmos escaláveis de particionamento de redes para a realização de experimentos randomizados. O primeiro é uma versão paralela do algoritmo ϵ -net. O segundo, NoMAS, é inspirado no FENNEL, mas usa uma função diferente para controlar o compromisso entre número de arestas conectando vértices em diferentes partições e balanceamento do tamanho das partições. Comparando os resultados obtidos para NoMAS e FENNEL em relação à qualidade do particionamento e à acurácia na estimação do ATE, concluímos que as duas formas de penalizar partições grandes geram resultados semelhantes para o modelo de resposta linear. Além disso, a qualidade das partições não exibiu uma correlação direta com a acurácia da estimação: embora o ϵ -net tenha levado aos piores resultados de particionamento, o MSE resultante em um dos datasets foi menor que dos outros algoritmos. É possível que diferenças mais acentuadas entre os algoritmos seriam observadas quando o modelo de estimação é diferente do modelo usado para gerar as respostas. Isto será investigado em trabalhos futuros.

Referências

- Aronow, P. M. and Samii, C. (2012). Estimating average causal effects under general interference. In *Summer Meeting of the Society for Political Methodology*, pages 19–21.
- Azevedo, F. G., Nogueira, B. D., Murai, F., and da Silva, A. P. C. (2018). Modelos de resposta para experimentos randomizados em redes sociais de larga escala. arXiv:1803.03497.
- Backstrom, L. and Kleinberg, J. M. (2011). Network bucket testing. In *WWW*, pages 615–624.
- Eckles, D., Karrer, B., and Ugander, J. (2014). Design and analysis of experiments in networks: Reducing bias from interference. *CoRR abs/1507.00803*, stat.ME.
- Fortunato, S. (2010). Community detection in graphs. *Physics Reports-Review Section of Physics Letters*, 486(3-5):75–174.
- Gui, H., Xu, Y., Bhasin, A., and Han, J. (2015). Network A/B Testing: From Sampling to Estimation. *WWW*, pages 399–409.
- Katzir, L., Liberty, E., and Somekh, O. (2012). Framework and algorithms for network bucket testing. In *WWW*, pages 1029–1036.
- Kohavi, R., Deng, A., Frasca, B., Walker, T., Xu, Y., and Pohlmann, N. (2013). Online controlled experiments at large scale. In *KDD*, pages 1168–1176.
- Kohavi, R., Longbotham, R., Sommerfield, D., and Henne, R. M. (2009). Controlled experiments on the web: survey and practical guide. *DMKD*, 18(1):140–181.
- Manski, C. F. (2013). Identification of treatment response with social interactions. *Econometrics Journal*, 16(1):S1–S23.
- Middleton, J. A. and Aronow, P. M. (2011). Unbiased Estimation of the Average Treatment Effect in Cluster-Randomized Experiments. *SSRN Electronic Journal*.
- Newman, M. E. J. (2006). Modularity and community structure in networks. *Proceedings of the National Academy of Sciences*, 103(23):8577–8582.
- Nishimura, J. and Ugander, J. (2013). Restreaming graph partitioning: simple versatile algorithms for advanced balancing. In *KDD*, pages 1106–1114.
- Saveski, M., Pouget-Abadie, J., Saint-Jacques, G., Duan, W., Ghosh, S., Xu, Y., and Airolidi, E. M. (2017). Detecting Network Effects. In *KDD*, pages 1027–1035.
- Stanton, I. and Kliot, G. (2012). Streaming graph partitioning for large distributed graphs. In *KDD*, pages 1222–1230. ACM.
- Tsourakakis, C., Gkantsidis, C., Radunovic, B., and Vojnovic, M. (2014). Fennel: Streaming graph partitioning for massive scale graphs. In *WSDM*, pages 333–342. ACM.
- Ugander, J., Karrer, B., Backstrom, L., and Kleinberg, J. (2013). Graph cluster randomization: Network exposure to multiple universes. In *KDD*, pages 329–337.
- Xu, Y., Chen, N., Fernandez, A., Sinno, O., and Bhasin, A. (2015). From infrastructure to culture: A/B testing challenges in large scale social networks. In *KDD*, pages 2227–2236.

Análise e Paralelização de um algoritmo para Gridding

Carlos A. T. Aguni¹, Daniel Cordeiro²

¹Escola Politécnica, Universidade de São Paulo

²Escola de Artes, Ciências e Humanidades, Universidade de São Paulo

{carlos.aguni,daniel.cordeiro}@usp.br

Abstract. *This paper analyzes the performance of a parallel implementation for the Gridding Method Algorithm in two different approaches: Intel® Xeon Phi Coprocessor (Many Cores architecture) and Nvidia Tesla K20x (General Purpose Graphics Processing Unit - GPGPU). We study its suitability when optimizing its serial implementation into a multi/many core environment as well as profiling its resource usage on memory, CPU, and cache usage on both CPU and accelerator.*

Resumo. *Este artigo analisa o desempenho do algoritmo de Gridding implementado de forma paralelizada em duas propostas: com uma placa aceleradora Intel® Xeon Phi (arquitetura Many Cores) e uma Nvidia Tesla K20x Unidades de Processamento Gráfico de Propósito Geral (General Purpose Graphics Processing Unit - GPGPU). Estudamos sua adequabilidade quando otimizado para um ambiente multi/many core e o perfilamos em relação ao consumo de recursos como memória, processador, e uso de caches na CPU e placas aceleradoras.*

1. Introdução

O *Square Kilometre Array* (SKA [Dewdney et al. 2009]) é o maior projeto de radiotelescópios do mundo e tem como objetivo reconstruir imagens do céu a partir dos dados coletados de várias antenas. O projeto, constituído por duas fases, iniciou-se em 2011 e pretende impulsionar o desenvolvimento científico em várias áreas da ciência. Tecnologias de satélites, processamento de sinais e métodos de computação e infraestrutura foram combinados para otimizar tanto o desempenho da execução quanto armazenar os dados do *workload* [SDP 2018].

A primeira fase do projeto finalizou-se em 2017 e conta com cerca de 250 antenas que simulam um grande radiotelescópio extremamente sensível [T.J. Cornwell 2012] capaz de coletar dados observados em uma área de um quilômetro quadrado do espaço. As informações capturadas pelas antenas correspondem a uma quantidade de dados que chegam a uma razão de aproximadamente 5,2 Terabytes por segundo.

Os satélites, uma vez configurados, são então acionados para capturar imagens do céu por períodos prolongados de horas a dias gerando, portanto, uma quantidade de dados economicamente inviável de ser armazenada em qualquer infraestrutura existente. Tal problema trouxe a necessidade de tratar os dados antes de seu armazenamento. No entanto, o poder computacional necessário para se tratar essa quantidade de dados é estimado na casa do PetaFLOPs [Dewdney and et. al 2010] na primeira fase do projeto e ExaFLOPs na segunda. Ainda espera-se que a primeira fase permita a captura e o tratamento de apenas 10% da capacidade total do projeto.

O processo do tratamento dos dados, desde a sua captação bruta até o seu armazenamento final, é composto por tarefas computacionalmente intensivas como: a correlação no processamento das imagens [Jacobs 2005], Convolução (*Convolutional Gridding* [Beatty and Pauly 2005]), Transformada de Fourier (FFT) [Frigo and Johnson 2005], etc. Este trabalho devota-se a estudar a etapa de Convolução, normalmente referenciada na literatura pelo seu termo em inglês *gridding*.

A computação do *gridding* é considerada um dos maiores desafios do projeto SKA [Braam and Wortmann 2016, Cornwell 2006]. *Convolutional gridding* é uma técnica utilizada em interferometria (combinação de n antenas) para converter as medições feitas pelas n antenas em dados numa malha regular. Em seguida, somente após calculada a inversa da transformada de Fourier, é possível obter algo próximo da imagem final do espaço. Vale ressaltar que, para realizar a FFT é necessário que a malha esteja disposta em uma grade regular, por isso a importância da convolução. O processo de convolução seguida da transformada de Fourier é, além de bastante estudado por astrônomos, também, muito utilizado nas áreas de ciências médicas, especialmente associado ao uso de máquinas de ressonância magnética, tomografia, etc. [O'Sullivan 1985, A. Eklund and LaConte 2013].

Vistos o ecossistema, a importância, relevância e fatores de gargalo do algoritmo *gridding* o presente artigo propõe um estudo de otimização e análise do algoritmo em arquiteturas de computadores orientadas ao total de trabalho realizado por unidade de tempo (*throughput-oriented architecture*). Uma abordagem diferente à computação convencional nas CPUs onde a importância está no tempo em que cada tarefa é executada (*latency-oriented architecture*).

O objetivo deste trabalho é otimizar a execução e análise do algoritmo de *convolutional gridding*, desenvolvido e utilizado pelo projeto SKA, utilizando métodos de programação paralela apresentando uma caracterização detalhada do consumo de recursos nos sistemas em que foram executados. A implementação proposta foi estudada em termos de suas relações carga/memória, carga/CPU e tempos de execução, e sua implementação baseia-se em uma divisão eficiente da carga das tarefas entre os recursos computacionais nas diferentes arquiteturas avaliadas.

Os *hardwares* orientado à tarefas (*throughput-oriented programming*) utilizados foram dois: Intel Xeon Phi [Fang et al. 2013] e Nvidia Tesla [Huang et al. 2008]. A utilização de tais placas aceleradoras não pode ser feita de forma independente, ou seja, elas trabalham em conjunto com a CPU (*latency-oriented programming*), portanto, as versões paralelas implementadas para ambas arquiteturas contemplam, também, otimizações paralelas realizadas na CPU. Por fim, a partir da versão serial foram criadas duas versões paralelas do algoritmo, uma para cada arquitetura de placa aceleradora:

1. $Máquina_{hospedeira}^1 + Máquina_{coprocessador}^1$
2. $Máquina_{hospedeira}^2 + Máquina_{coprocessador}^2$

Logo, além da versão serial já existente, o artigo concentra-se em descrever os métodos de paralelização, escolha de parâmetros (quantidade de *threads*, memória, cache) e *profiling* para as versões que envolvem a placa aceleradora e sua respectiva máquina hospedeira. O artigo encontra-se organizado da seguinte forma. A seção 2 explica com maiores detalhes o algoritmo de *convolution gridding*. A seção 3 introduz sobre o ambiente utilizado e as seções 4 e 5 apresentam de que forma o algoritmo fora paralelizado e

quais foram os resultados obtidos respectivamente e, por fim, as conclusões discutidas na seção 6.

2. Contexto

De forma sintética, o algoritmo analisado sintetiza as imagens do céu recuperadas pelas antenas e as combinam em uma única imagem final. Tal processo pode ser descrito e separado em três partes principais: a inicialização, o *gridding* e a finalização.

A inicialização envolve todo o processo de pré-configuração da malha final onde serão armazenados os dados referentes ao espaço. Cada antena realiza as medições em uma sistema de coordenadas UVW representada na Figura 1. Nessa fase também são geradas as funções de convolução que serão utilizadas na fase do *gridding*. Na etapa seguinte, é realizada a re-amostragem convolucional, ou *gridding*, cujo objetivo é juntar em uma só malha os dados obtidos pelas antenas; é esta etapa que o presente trabalho se propõe a otimizar. Por fim, durante a etapa de finalização é executada a inversa da transformada de Fourier e, em seguida, é feito o armazenamento da imagem final no *storage*. Tais passos podem ser vistos na Figura 2 onde a área azul representa o algoritmo de *gridding*.

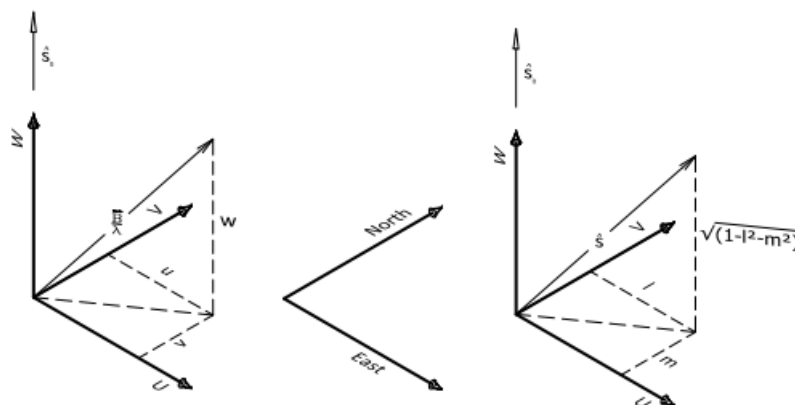


Figura 1. Representação do sistema de coordenadas UVW. A esquerda a sua representação em componentes (u, v, w) e a direita quando apontado para um objeto celeste. Imagem retirada de [Muscat 2014].

Note que há vários outros métodos [Muscat 2014, Sørensen et al. 2008] que permitem a transformação dos sinais recebidos pelos telescópios em dados coerentes para serem analisados. *Convolution gridding* [Humphreys and Cornwell 2011] foi o método escolhido pelo projeto SKA.

2.1. Descrição do algoritmo de *Gridding*

Transformadas de Fourier podem ser rapidamente calculadas através do algoritmo conhecido como *Fast Fourier Transform* (FFT). Mas seu uso impõe como pré-condição que os dados estejam dispostos em intervalos regulares. Quando os dados de entrada não são regulares, faz-se necessário realizar um pré-processamento para torná-los regular.

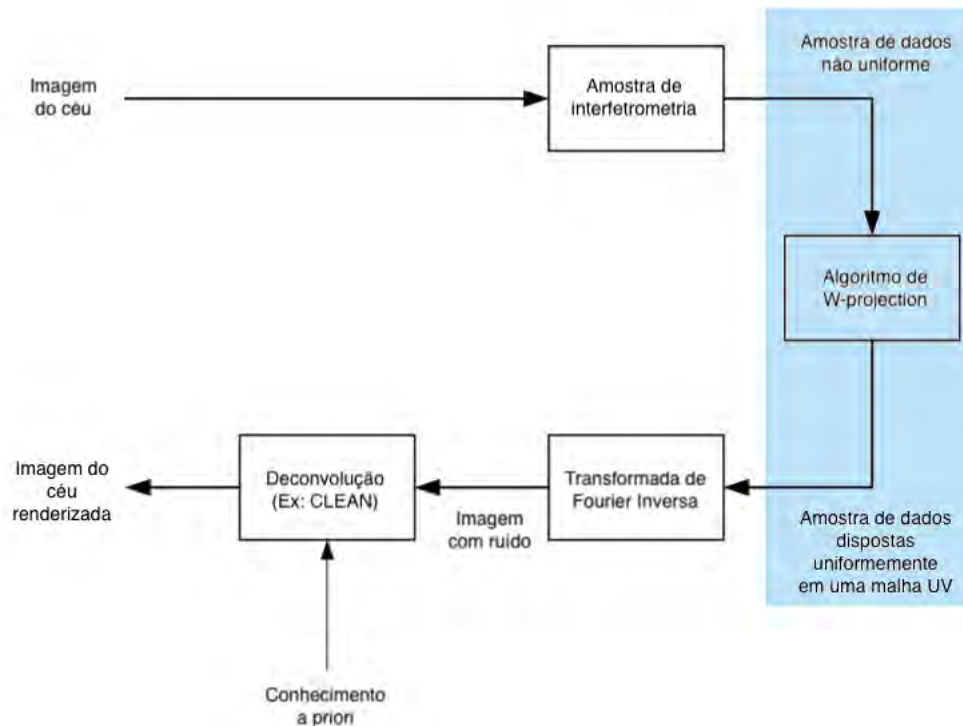


Figura 2. Procedimento de captura, processamento e armazenamento das imagens capturadas do espaço. Adaptada de [Muscat 2014].

Convolution gridding é um dos métodos utilizados para regularizar os dados. Através dele é possível realizar a convolução de espaços irregulares de dados para regular. Realizar a convolução significa passar a malha de dados por um filtro (função de convolução representado na Figura 3). Funções de convolução são bastante utilizadas na área da física quando a intenção é de se atenuar ou remover ruído como por exemplo de um áudio, ou imagem, etc. No SKA é utilizado o método conhecido como *W-projection* [Cornwell 2004], capaz de juntar n malhas em uma única [Muscat 2014].

O algoritmo de *gridding* pode ser expresso de forma relativamente simples, como demonstrado no pseudocódigo 1. O algoritmo descrito dessa forma, entretanto, possui muitas dependências de dados. Por exemplo, existe uma dependência de dados no vetor *grid*, onde uma tentativa ingênua de paralelização do cálculo dos *samples* acarretaria em uma situação de concorrência no acesso aos dados caso o *d_index* fosse modificado por duas ou mais *threads*.

2.2. Descrição do workload

Utilizamos como base da implementação paralela o código serial disponibilizado pela *Australia Telescope National Facility* (projeto CSIRO) em sua conta no GitHub [Humphreys et al. 2013]. O código compreende também um conjunto de testes automatizados que utilizamos com a finalidade de verificar a corretude da versão paralela.

O *workload* utilizado, disponibilizado junto com o código, contém 3.200.000 amostras, onde cada amostra possui 536MB no formato de uma malha 2D de tamanho

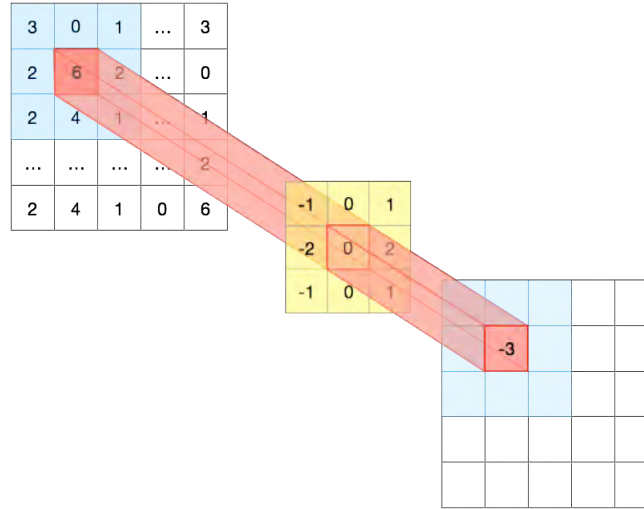


Figura 3. Representação da utilização de uma função de convolução.

Pseudocódigo 1. Pseudocódigo do algoritmo de *gridding* onde cada *sample* seria a saída de uma observação de cada telescópio. O *d_index* armazena sua posição da malha única e *c_index* armazena a posição que iniciará os valores da função de convolução.

```

1  entrada: malha , função de convolução, amostras
2  saida: malha atualizada
3  inicio
4      para cada: amostra em amostras (índice d_index itera sobre amostras)
5          g_index ← Posição inicial na malha da amostra (u, v, w)
6          c_index ← Posição inicial na função de convolução
7
8          para: suppv até comprimentoSup
9              peso ← Peso da amostra da função de convolução
10             para: suppu até comprimentoSup
11                 malha[g_index++] += peso * cfunc[c_index++]
12             g_index += comprimentoMalha
13         fim
14     retorna malha
15 fim

```

$4096 \times 4096 = 16.777.216$ elementos que servirão de entrada do algoritmo de *Gridding*. O projeto também disponibiliza um programa chamado *verify* que computa o erro entre os resultados de diferentes versões utilizando a norma L^1 do vetor a partir da fórmula $S = \sum_{i=1}^n |y_i - f(x_i)|$, onde o resultado (**S**) é a soma das diferenças absolutas entre o resultado (Y_i) e o valor estimado ($f(x_i)$). Este trabalho utiliza-se do *verify* para assegurar a corretude da implementação.

3. Características do Ambiente de Testes

Estudamos a paralelização do algoritmo usando tanto GPUs como Intel Xeon PHIs. A análise experimental foi feita utilizando os seguintes recursos computacionais:

- AMD Opteron(tm) Processor 6380 2,5GHz + 1 Placa Nvidia Tesla K20x.
- Intel Xeon E5-2650 2,6 Ghz + 1 Placa Intel Xeon PHI 5110P 60 Cores

A Tabela 1 apresenta uma comparação ente as placas utilizadas neste trabalho.

Para fins de desambiguação, iremos utilizar os termos:

	K20x	Phi 5110P
# Processadores	2688	60
Proc. Velocidade	732 MHz	1,05GHz
Memória RAM Total	6 GB	8 GB
Chipset	GK110	Knights Corner

Tabela 1. Comparação entre as placas Nvidia K20x e Xeon Phi 5110P utilizadas.

1. *GPU* para se referir às placas aceleradores Nvidia
2. *MIC* (Intel® *Many Integrated Cores*) para se referir à Xeon Phi.
3. $HOST_{GPU}$ para se referir ao nó computacional que inicia o código a ser executado na *GPU*
4. $HOST_{MIC}$ para se referir ao nó computacional que inicia o código a ser executado na *MIC*
5. “Versão híbrida”: $HOST_X + X$, $X \in \{MIC, GPU\}$ onde X refere-se à arquitetura sendo discutida naquela seção ou subseção salvo exceções onde a máquina é então especificada ou quando faz-se menção à ambas as máquinas.

A arquitetura Xeon Phi oferece dois modelos de programação paralela: *native* e *offloading*. No primeiro, o código é executado nativamente no coprocessador e suas dependências precisam ser previamente carregadas na MIC. No segundo, a aplicação é executada no HOST e partes do código são carregadas no coprocessador.

O presente trabalho opta pelo uso do modo *offloading* para a otimização da execução do algoritmo devido à maior flexibilidade para organizar os dados do workload, o que permite melhor controle de sincronização de dados entre HOST e MIC como também transferência de trechos de código pré-escritos para serem executados na MIC sob demanda. Note que os recursos de uma MIC são limitados, então a ideia é aproveitá-la o máximo possível tanto em termo de processamento, quanto em termos de memória, e não usá-la para E/S de dados. Utilizamos OpenMP (paralelização por *threads*) para habilitar paralelismo na máquina.

Uma GPU permite apenas o modo *offloading*. O modo de acesso e execução em uma placa GPU é muito semelhante à MIC, uma vez que também é necessário explicitar quais componentes do código serão executados na GPU e é necessário gerenciar manualmente quais dados serão transferidos para/das placas.

No HOST existem dois modos principais de obter-se paralelismo: com o uso de *threads* ou com o uso de processos (via *fork*). Esses modos são facilitados com o uso de bibliotecas que implementam um conjunto de funcionalidades de controle e de troca de mensagens de mais alto nível, tais como: OpenMP, que oferece um conjunto de diretivas (*pragmas*) e programação paralela com *threads*, e MPI (*Message Parsing Interface*) que oferece uma biblioteca com um conjunto de funções para controle de processos. Os prós e contras em se utilizar um ou outro dependerá sempre da natureza do problema em questão. Para a otimização do algoritmo de *gridding*, visto a necessidade de haver um controle maior sobre o uso de memória *intra-node*, optou-se pela utilização do OpenMP.

4. Estratégias de paralelização

Após um estudo do algoritmo serial de *Gridding* implementado pela *Australia Telescope National Facility*, e a partir da utilização de ferramentas de *profiling*, observamos que

um único trecho de código, responsável pela soma e multiplicação de números complexos, estava ocupando quase todo o tempo de processamento da aplicação. Sendo assim, decidimos focar os esforços em diminuir o tempo computacional neste trecho.

A versão sequencial precisou ser adaptada para poder ser executada nas aceleradoras. A versão original, escrita em C++, utiliza de uma grande quantidade de estruturas de dados complexas oferecidas pela linguagem (listas ligadas, objetos, etc.), enquanto que as aceleradoras só permitem o uso de tipos de dados primitivos do C. A adaptação não só facilitou a manipulação e transferência de dados, como também trouxe um significativo ganho de desempenho.

A versão híbrida do algoritmo leva em conta a possibilidade de execução de forma assíncrona de determinados trechos do código, que podem ser executados nas aceleradoras. Antes da execução, porém, os dados devem ser tratados e, então, copiados para as aceleradoras. Após a execução, os resultados devem então ser convertidos para as estruturas de dados principais do programa.

O processo de paralelização híbrida segue três etapas. Na primeira etapa é necessário aplicar uma solução de paralelização no HOST, seguida por uma solução na MIC/GPU para só depois mesclar as duas. Tais fases facilitam a organização do código, depurações, execuções de testes, etc.

A seção a seguir descreve o perfil de desempenho da versão sequencial e mostra os fatores que guiaram o processo de paralelização para execução no HOST, Xeon PHI e GPU.

4.1. Paralelização no HOST

Através da ferramenta Intel VTune Amplifier¹ foi possível encontrar pontos de gargalo na execução serial da função de convolução. Dos resultados obtidos destacamos que do total do tempo de execução da aplicação, de 508 segundos, o código gastou 273 segundos na linha 11 do pseudocódigo 1 mais interna do “for” aninhado responsável pelo processo de convolução da malha.

A versão paralela para execução no HOST demonstrou possuir uma grande concorrência de dados nos valores da malha quando várias *threads* são executadas ao mesmo tempo. Detectou-se, por exemplo, um cenário onde dois *ginds* (índice da malha) eram escolhidos no mesmo instante, acarretando a atualização dos valores no mesmo local por duas *threads* concorrentes.

Para resolver o problema da concorrência, a solução foi criar uma cópia da malha para cada nova *thread* criada. Desse modo, cada *thread* teria sua própria malha temporária para trabalhar sequencialmente e a atualização das malhas temporárias na malha principal seria feita apenas no fim. Sendo assim, a complexidade de espaço utilizado pela aplicação seria de modo linear para cada *thread* criada igual a $\mathcal{O}(n)$. Dado que cada ponto na malha necessita de um tipo complexo, temos que o uso total de RAM na máquina pode ser calculada pela fórmula: $RAM_{utilizada} = nThreads * tamanhoMalha * 2 * 8$.

Com a criação de cópias das malhas para novas *threads* foi possível dividir a carga de trabalho do número de amostras para ser executada de forma paralela entre as

¹<https://software.intel.com/intel-vtune-amplifier-xe>

n threads. Como podemos ver na Figura 4 a quantidade de amostras da esquerda agora pode ser paralelizada.

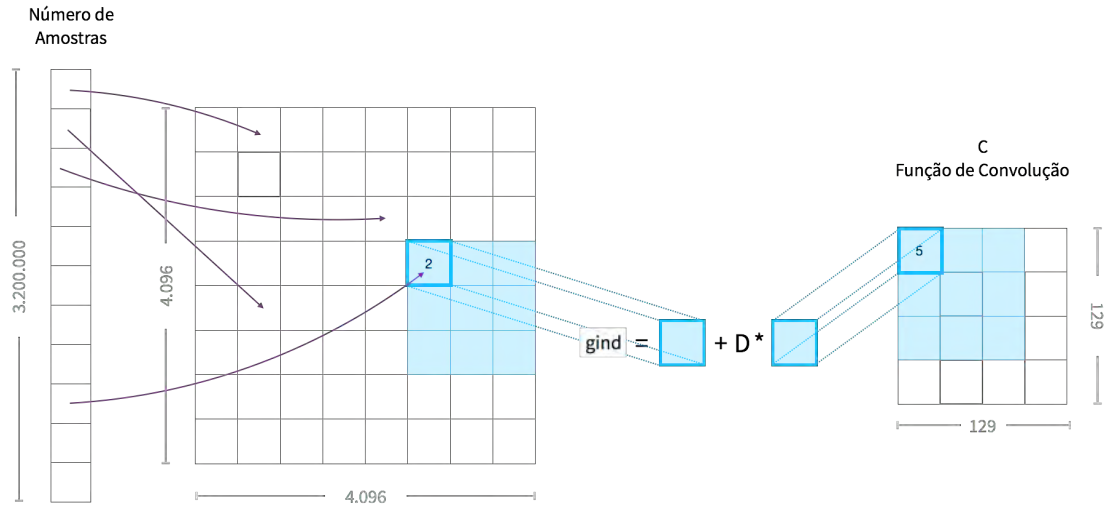


Figura 4. Funcionamento do algoritmo de *gridding*. O número de amostras à esquerda é endereçado pelo d_index 1 que representa sua posição inicial na malha.

4.2. Paralelização na Xeon PHI e GPU

A estratégia de paralelismo para a Xeon PHI e GPU foi a mesma, o uso do modo *of-floating*. A diferença foi o arcabouço de programação paralela usado: OpenMP para a Xeon Phi e CUDA para a GPU. O método de cópia da malha feito nos *HOSTs* foi mantida mas, percebe-se que as placas aceleradoras possuem muito menos espaço em RAM disponível (7GB) quando comparado ao *HOST*, ou seja, se optássemos por utilizar o mesmo método realizado no *HOST* para a *MIC/GPU*, levando-se em conta a possibilidade de criar 240 threads ($\#processadores * 4(hyperthread)$) na *MIC* e 28.672 ($\#threadsporSM(2048) * SMs(14)$) seria preciso $RAM_{utilizada} = 240 * 4096^2 * 2 * 8 = 64GB$ de espaço disponível na *MIC* e muito maior na *GPU*.

Sabendo disso já é possível se calcular os limites de cópias que preenchem cada placa. Para a *MIC* o número de cópias/threads não pode passar de $nThreads * 4096^2 * 2 * 8 \leq 8GB$, ou seja, $nThreads$ para *MIC* não pode passar de 22. E para a *GPU* o número de cópias não pode passar de $nThreads * 4096^2 * 2 * 8 \leq 6GB$, ou seja, $nThreads$ não pode passar de 18.

Agora nota-se que o número máximo de threads possíveis de se utilizar sem que ocorra concorrência de dados é muito menor que o número máximo de threads capaz de serem executadas nas placas. Se executado o algoritmo dessa forma, mesmo o número de threads sendo maior que aquelas criadas no *HOST* percebe-se que o resultado será muito pior que o *HOST* visto os processadores das placas aceleradoras dotadas de menos instruções e velocidade de processamento.

Agora como atingir a máxima ocupação de threads na *MIC* e *GPU*? A solução encontrada foi dividir as multiplicações das linhas e colunas do array da função de convolução em tiras para um número novo de threads criadas, ou seja, o número de máximo de amostras a serem computadas dentro das placas nunca será maior que o

número de cópias da malha. No entanto, é possível controlar o número de *threads* que irão processar cada amostra. Por exemplo, consideremos a *GPU*, o número máximo de amostras calculadas por vez é 18 ($nThreads * 4096^2 * 2 * 8 \leq 6GB$). No entanto é possível, por exemplo, criar mais 10 *threads* e torná-las cada uma responsáveis por calcular $\frac{1}{10}$ das linhas da função de convolução. O que pode ser visto na figura 5.

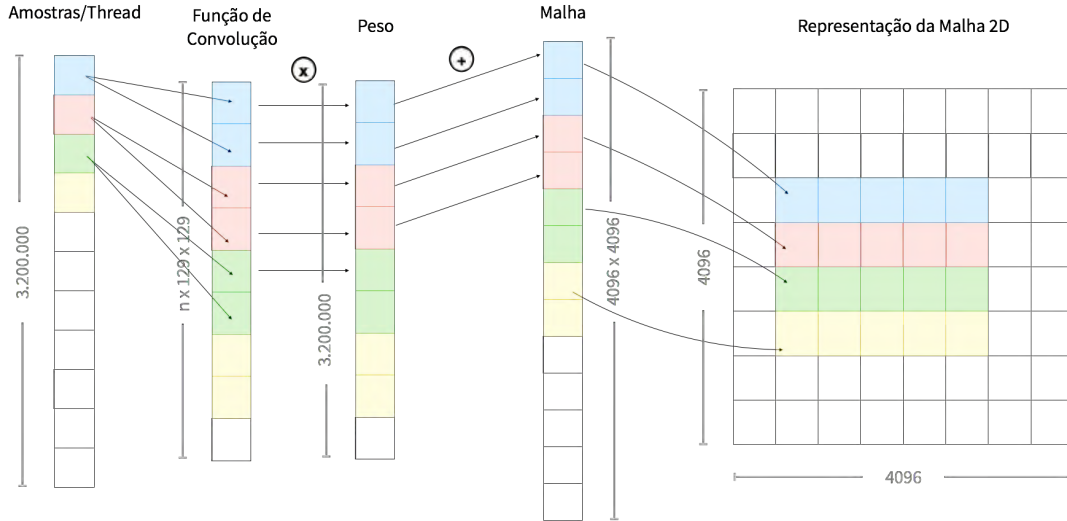


Figura 5. Esquema do fluxo percorrido pela *thread* refatorado para se explorar o *cache*

Outro fator importante comentar foi que a adição de *threads* para dividir a carga de trabalho realizada na função de convolução ajudou a granularizar o problema possibilitando o acesso de cada *thread* a pequenos trechos da memória. Dessa forma o objetivo é encontrar parâmetros de granularidade para que o nível de ocupação na *MIC* e *GPU* se aproximem da máxima ao mesmo tempo que se pudesse aumentar a frequência de utilização dos *caches L1* e *L2* mais próximos do processador. Na figura 6 é possível ver a representação do algoritmo *gridding* que se despreocupa com a utilização eficiente do *cache* o que acaba levando a perdas de *cache* e uma sobrecarga devido a coerência de *caches*.

Comparando-se as figura 5 com a figura 6 é possível apontar algumas vantagens que a versão *cache-friendly* tem da versão que podemos chamar de ingênua: há uma melhor divisão de trabalho para cada *thread*, os *caches* são melhores explorados, é possível controlar cada tira a ser realizada pela sua *thread* respectiva. O próximo passo, agora, seria encontrar um valor $nTiras_x$ e $nTiras_y$ onde cada variável é responsável por calcular a tira, supondo o vector da função de convolução sendo C , $largura_x$ sua largura e $largura_y$ sua altura, a tira responsável por essa *thread* pode ser representada da seguinte forma $C[largura_x : nTiras_x][largura_y : nTiras_y]$ onde o id que representa a *thread* pode ser usado para calcular o deslocamento na linha e/ou coluna. Lembrando que cada valor diferente dado leva a um molde totalmente do problema como pode se ver na figura 7.

Dessa forma, cada *thread* fica responsável por sua respectiva tira da função de convolução o que resulta num maior aproveitamento dos *caches*. Por exemplo, caso a divisão de tarefas fosse feita assumindo-se $largura_x = 129$ e $largura_y = 1$ teríamos um cenário onde cada *thread* se ocuparia em obter $largura_x * largura_y * complex\_type =$

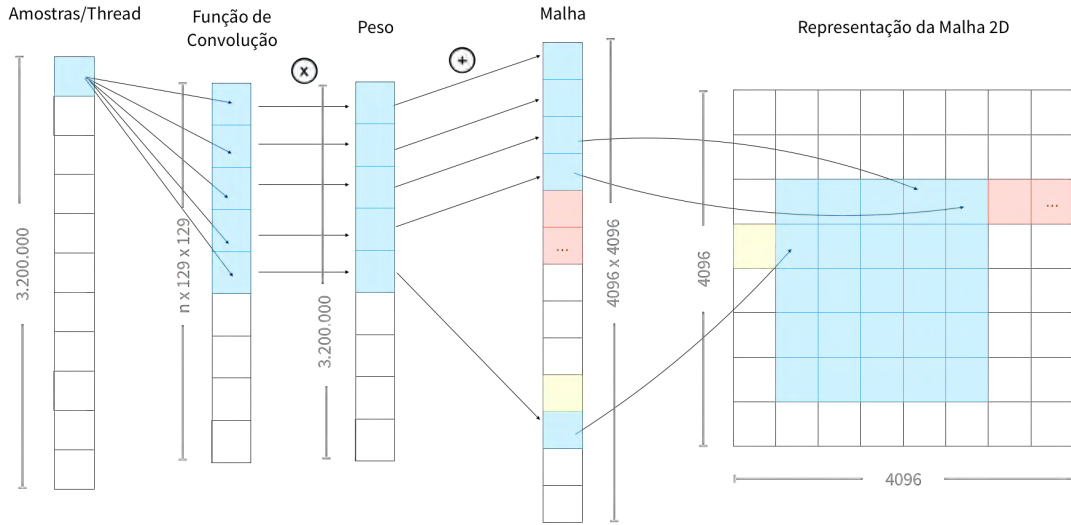


Figura 6. Imagem representando os acessos à memória realizados por uma *thread*. O acesso à próxima linha pela função de convolução não é o mesmo quando este é refletivo para a malha. Vê-se que a mesma *thread* precisa executar um grande “salto” de memória.

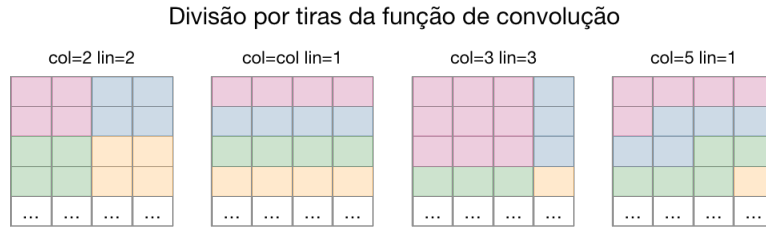


Figura 7. Imagem representa o balanço de carga para cada *thread* variando-se a granularidade da divisão de linhas e colunas.

$129 * 1 * 16 = 2K\text{Bytes}$. Somando-se à essa *thread* outras constantes de dados e controle, acesso ao vetor de pesos e à malha podemos supor que cada *thread* possui acesso a um tamanho B . A partir das especificações técnicas retiradas da *MIC* e *GPU* temos que a *MIC* possui 32KB de cache L1 e 512KB de cache L2 dividido entre as 4 *hardware threads* de cada processador. Já a *GPU* possui um cache de até 64KB (cache variável, depende da memória compartilhada) do L1 e 1.5MB para o cache L2. Logo, o que se precisa fazer é escolher um valor de $largura_x$ e $largura_y$ para que o valor de B chegue o mais próximo do cache L1.

A seguir serão apresentados os resultados obtidos a partir da variação da carga de trabalho do número de amostras entre cada uma das duas arquiteturas seguidas de uma análise de como os parâmetros escolhidos impactaram no desempenho da aplicação.

5. Resultados

As execuções nos dois ambientes de testes foram feitas de tal forma que se pudesse dividir o número de amostras a serem calculadas entre a *MIC/GPU* e seus respectivos *HOSTs*. A variação da carga de trabalho foi feita manualmente onde cada linha dos gráficos montados abaixo significa uma execução diferente.

Lembrando que foram comparados apenas as execuções realizadas na mesma

máquina, visto que especificações, arquiteturas e variáveis do ambiente (GHz, RAM, número de processadores, versão sistema operacional, marcas diferentes de processadores, versão de compiladores) tornam inviável uma comparação direta a não ser pelo tempo de execução e o quão bem a aplicação escalou comparado com sua versão serial.

Levando-se em conta a corretude dos dados, em todas as execuções, o erro calculado pela versão paralela comparando-se à versão sequencial, o resultado da taxa de erro não ultrapassou 1×10^{-15} .

5.1. Intel Xeon Phi

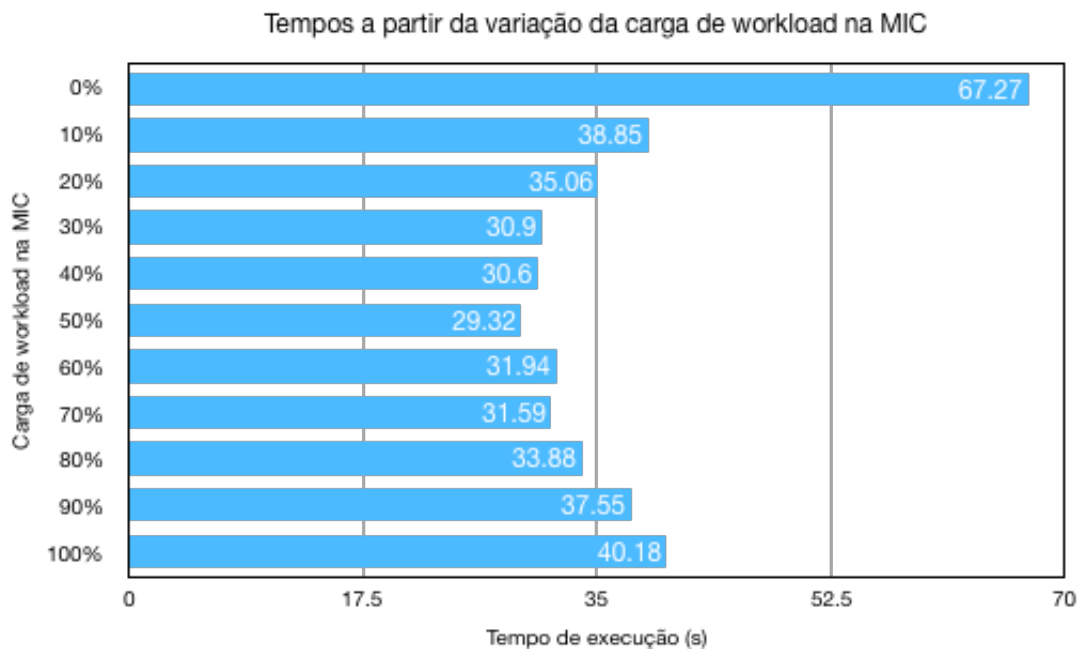


Figura 8. Tempos de execução da versão paralela no HOST (MIC com 0%) e da versão híbrida (MIC com 10% a 100%).

O código paralelo no $HOST_{MIC}$ foi executado com 16 *threads* e durou 67.27 segundos como pode ser visto na figura 8 quando a carga de trabalho realizada na *MIC* é 0%, ou seja, aproximadamente três vezes mais rápida que a versão serial de 185.12s. Essa solução paralela ocupou $RAM_{utilizada} = 16 * 4096^2 * 2 * 8 = 4\text{Gigabytes}$ de RAM

Já na versão híbrida o melhor tempo de execução foi obtido quando dividido 50% da carga das amostras entre a *MIC* e $HOST_{MIC}$. Tal execução foi encontrada após se definir o número de cópias da malha em $nThreads = 22$, ou seja, o número máximo de amostras a serem executadas na *MIC* e $largura_x = 1$ e $largura_y = 10$. Logo, o número total de *threads* geradas foi $TotalThreads = nThreads * largura_x * largura_y = 220$ threads geradas.

Logo, comparando-se o melhor tempo obtido na versão híbrida de 29.32 com a versão serial temos um ganho de 6 vezes e 2 vezes quando comparada com a execução do $HOST_{MIC}$.

5.2. Nvidia Tesla K20x GPU

Para se avaliar a implementação híbrida na GPU, primeiro implementou-se a versão paralela no $HOST_{GPU}$, que, utilizando 16 processadores foi capaz de reduzir o tempo de

363.12 segundos obtido na execução serial para 202.93 segundos, ou seja, um desempenho de aproximadamente 2 vezes da solução serial.

Já para a versão híbrida, sendo possível dividir a carga de amostras entre *GPU* e *HOST_{GPU}* o melhor tempo de execução obtido foi quando 90% da carga de trabalho das amostras foi realizado na *GPU* atingindo um tempo de execução de 30.05 segundos, ou seja, um desempenho 12 vezes mais rápido quando comparada à versão serial e aproximadamente 7 vezes mais rápido que a solução paralela implementada na *HOST_{GPU}*. As demais variações de carga podem ser vistas no gráfico da figura 9.

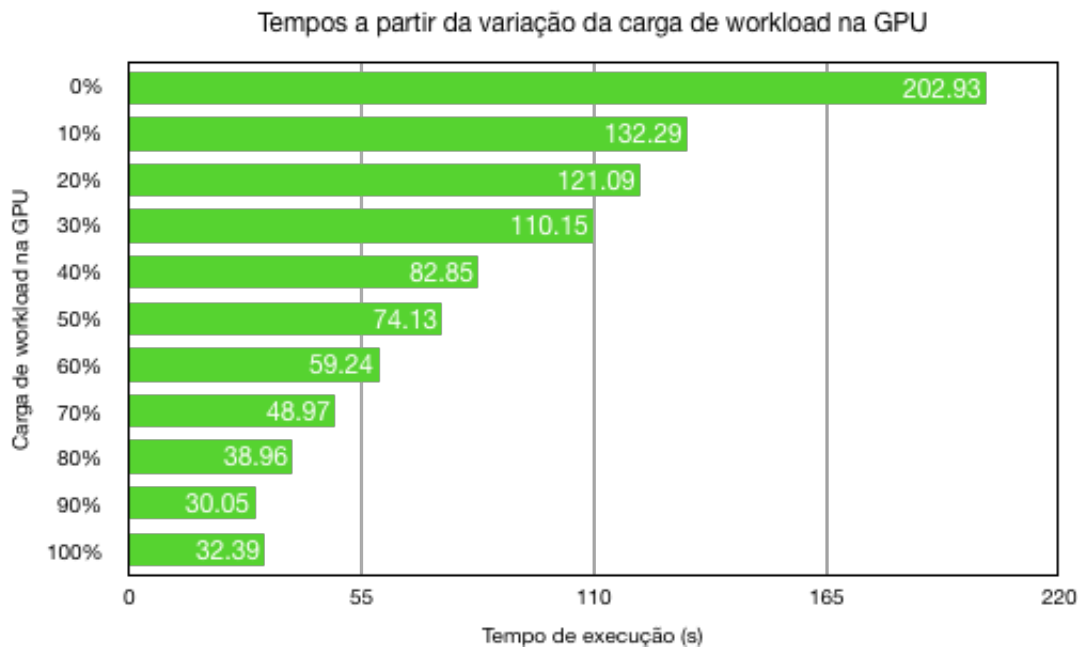


Figura 9. Gráfico do tempo de execução variando-se a carga executada na GPU.

Tal performance na *GPU* foi obtida quando definido o número de cópias da malha em 18, ou seja, $nThreads = 18$ que também reflete ao máximo de amostras sendo executadas na *GPU*. Os parâmetros que resultaram no melhor tempo de execução quando 90% da carga de trabalho foi executada na *GPU* foram $largura_x = 5$ e $largura_y = 129$, ou seja, sendo o tamanho da matriz de convolução 129×129 cada *thread* fica responsável por calcular um vetor de tamanho $\lceil \frac{129}{largura_x} \rceil \lceil \frac{129}{largura_y} \rceil = \lceil \frac{129}{5} \rceil \lceil \frac{129}{129} \rceil = [\approx 25][1]$ na malha. O número total de *threads* criadas ficou, então, $TotalTreads = nThreads * largura_x * largura_y = 11.610$ threads ou 645 threads por *threadBlock*. Executando-se o Nvidia Visual Profiler ² foi possível observar que cada *thread* utilizou uma quantidade de 31 registros por *thread* com uma taxa de ocupação de 98.4%.

6. Considerações Finais

O presente artigo mostra uma análise de desempenho e consumo de memória de diferentes implementações paralelas do algoritmo de gridding executadas em diferentes arquiteturas de computadores, no caso HOSTs, MIC e GPU.

Uma análise preliminar do desempenho mostrou a existência de contenção de dados no acesso à malha. Propomos uma cópia temporária de dados para uso local por cada

²<https://developer.nvidia.com/nvidia-visual-profiler>

thread que possibilitou a execução de n amostras em paralelo. No entanto, tal abordagem gerou um grande consumo de memória, que se mostraria inviável em sistemas que possuem pouco espaço em memória, como é o caso de placas aceleradoras. Propusemos, por isso, um novo arranjo no acesso à memória que permitiu cada *thread* acessar somente uma área da memória, o que possibilitou o uso eficiente das aceleradoras.

Nossa análise experimental mostrou que ser possível obter um *speedup* de 12 vezes ao se executar 90% da carga de trabalho nos cores da placa GPU. Já na arquitetura Xeon Phi, obtivemos um *speedup* de 6 vezes quando 50% da carga de trabalho é executada na MIC, enquanto o restante é executado no HOST.

A análise do desempenho realizada confirmou nossa hipótese inicial de que o gargalo de desempenho estava relacionado à forma como o algoritmo realiza os acessos à memória. A avaliação do desempenho forneceu indicadores importantes que nos levaram a concluir que o algoritmo paralelo proposto é eficiente para o problema, e que a utilização bem sucedida da memória *cache* foi fundamental para os bons resultados obtidos.

Referências

- [A. Eklund and LaConte 2013] A. Eklund, P. Dufort, D. F. and LaConte, S. (2013). Medical image processing on the GPU: Past, present and future. *Med. Image Anal.*, vol. 17 pp. 1073-1094.
- [Beatty and Pauly 2005] Beatty, Philip J., D. G. N. and Pauly, J. M. (2005). Rapid gridding reconstruction with a minimal oversampling ratio. *Medical Imaging, IEEE Transactions on* 24.6, 799-808.
- [Braam and Wortmann 2016] Braam, P. and Wortmann, P. (2016). Kernel Prototyping SOW. Technical Report SKA-TEL-SDP-0000083, Science Data Processor Consortium.
- [Cornwell 2004] Cornwell, T. J., G. K. . B. S. (2004). W projection: A new algorithm for wide field imaging with radio synthesis arrays. *Astronomical Data Analysis Software and Systems XIV ASP Conference Series*, v. 347.
- [Cornwell 2006] Cornwell, T. (2006). Computing costs of imaging for the xNTD. Technical Report ASKAP Memo Series 001. *ANTF*.
- [Dewdney and et. al 2010] Dewdney, P. E. and et. al (2010). SKA phase 1: Preliminary system description. SKA memo. 130.
- [Dewdney et al. 2009] Dewdney, P. E., Hall, P. J., Schilizzi, R. T., and Lazio, T. J. L. W. (2009). The square kilometre array. *Proceedings of the IEEE*, 97(8):1482–1496.
- [Fang et al. 2013] Fang, J., Varbanescu, A. L., Sips, H., Zhang, L., Che, Y., and Xu, C. (2013). An empirical study of Intel Xeon Phi. *arXiv preprint arXiv:1310.5842*.
- [Frigo and Johnson 2005] Frigo, M. and Johnson, S. G. (2005). The design and implementation of FFTW3. *Proceedings of the IEEE*, 93: 216–231.
- [Huang et al. 2008] Huang, Q., Huang, Z., Werstein, P., and Purvis, M. K. (2008). GPU as a general purpose computing resource. In *PDCAT*.
- [Humphreys and Cornwell 2011] Humphreys, B. and Cornwell, T. (2011). Analysis of convolutional resampling algorithm performance. *Memo*, 132.

- [Humphreys et al. 2013] Humphreys et al., B. (2013). SKA GitHub. <https://github.com/ATNF/askap-benchmarks>. Acesso em 17/08/2017.
- [Jacobs 2005] Jacobs, D. (2005). Correlation and convolution. *Class Notes for CMSC 426*.
- [Muscat 2014] Muscat, D. (2014). High-performance image synthesis for radio interferometry.
- [O’Sullivan 1985] O’Sullivan, J. D. (1985). A fast sinc function gridding algorithm for fourier inversion in computer tomography. *IEEE Trans. Med. Imag.*, vol. 4 pp. 200-207.
- [SDP 2018] SDP (2018). SKA Website Science Data Processor. <https://www.skatelescope.org/sdp/>. Acesso em 23/03/2018.
- [Sørensen et al. 2008] Sørensen, T. S., Schaeffter, T., Noe, K. Ø., and Hansen, M. S. (2008). Accelerating the nonequispaced fast fourier transform on commodity graphics hardware. *IEEE Transactions on Medical Imaging*, 27(4):538–547.
- [T.J. Cornwell 2012] T.J. Cornwell, M.A. Voronkov, B. H. (2012). Correlation and convolution. *Wide field imaging for the Square Kilometre Array*.

Back to the Past: When Segmentation Is More Efficient Than Paging

Lauri P Laux Jr, Roberto A Hexsel

Departamento de Informática,
Universidade Federal do Paraná

lauripaulo@gmail.com, roberto@inf.ufpr.br

Abstract. *Virtual Memory was devised in a time of scarce resources. In the coming decade we expect to see physical memory systems populated with 2^{64} bytes of RAM, a fraction of which may be non-volatile. Demand paging is inefficient in such large memories because space (Page Tables) and time (Page Table walks) overheads are too high. We collected execution traces from six applications and characterized their virtual memory behavior with respect to miss rates in references to Translation Buffers (TLBs) and Segment Buffers (SBs). Our measurements indicate that the miss rates for SBs are 2-3 orders of magnitude smaller than for TLBs. In light of these results, we discuss some of the design implications of segmented systems and of SBs.*

1. Introduction

The first edition of “Computer Architecture, A Quantitative Approach” [HP90], stated, as a rule of thumb, that *the memory needed by the average program grows from 1/2 to 1 address bit per year*. The book was published in 1990, and around 1995 we saw the introduction of 64 bit microprocessors with memory buses that were 32-36 bits wide. A decade later, memory buses widened to 40 bits, and in the mid 2010’s 50+ bit buses are available. If this trend continues, in the next decade we can expect to see processors which may reference the full 2^{64} byte address space, all of it populated with RAM, with possibly a large fraction comprised of non-volatile RAM.

The memory systems of the next decade will be rather different from those of the early 2000s, when Intel discontinued support for segmentation. We believe it might be convenient to reconsider the design of the virtual memory system, as the premises that held since the early 1960s are no longer valid. Some of the recent trends include (i) slow magnetic disks are being replaced by $\geq 100\times$ faster solid state disks (SSDs); (ii) installed RAM capacity is so large, and inexpensive, that swapping may become unnecessary; (iii) when the promise of non-volatile RAM is fulfilled, magnetic disks will become “second-class peripherals” and a complete redesign of file systems may be worthwhile, and this may entail the elimination of buffer caches, for instance [LH16].

Managing such gigantic address spaces with demand paging becomes rather expensive, both in terms of space – one huge page table per process – and time – paging traffic between the translation buffer and primary memory. For these large physical memories, we believe Multics style segmentation to be a better memory management model than demand paging [DD68].

This paper presents the characterization of the virtual memory behavior in terms of the miss rates when accessing the buffer that hold virtual-to-physical address mappings. We simulated translation buffers for demand paging and for segmented virtual memory. The simulated buffer organizations are not meant to represent what one may find in current products as we are interested in counting memory references rather than measuring the simulated performance of said buffers. As we are interested in characterizing memory behavior of systems which will be available ten years hence, we do not try to guess what will be the specific design parameters of the translation buffers, as the design space is rather large.

We performed measurements to assess the effectiveness of a segmented virtual memory system and compared that to a demand paging system, for translation buffers of similar complexity. Our results indicate that miss rates of the segmentation address translation buffer is 100 to 1000 $\times$ smaller than its demand paging equivalent. This, plus the gains achievable in handling the smaller data structures, indicate that segmentation may be a worthy alternative to demand paging when one considers physical memories with 2^{64} bytes.

Traces were collected from the execution of six real programs, and with these traces we compare the performance of address translation with Translation Lookaside Buffers (TLB) for paging, and Segmentation Buffers (SB) for segmentation. Our tracing tool builds a Segment Table for the executing process and rewrites the linear addresses into segmentID-displacement tuples. These are then used to compute the TLB and SB hit rates. In this paper we consider single processor systems.

In the next section we define both Demand Paging and Segmented virtual memory systems. Section 3 describes our trace collection tools, and in Section 4 we present the simulation data collected. In Section 5 we compare the miss rates measured for TLBs and SBs; and in Section 6 we discuss system design issues in light of our results. Section 7 discusses related work, and Section 8 presents our conclusions.

2. Paging and Segmentation

The need for virtualizing the memory system arose in the mid 1950's as the programs plus their data were becoming larger than the physical memories then available. The earlier systems made use of segmentation because it was intuitive: as a program is logically divided into three segments, text (code), data, and stack, portions of the physical memory should be allocated to these segments [Den70]. Segmentation had a serious drawback as the expensive physical memory could be underutilized because of fragmentation. The holes between the segments residing in memory would be too small for any new segment, whereas the scattered free space could accommodate one or more segments. This was called "external fragmentation" because the available spaces were external to the segments.

The computer architecture group at Manchester University designed the first demand paging system [KELS62]. Memory was divided into small chunks (pages) and the capacity of physical memory was augmented by "virtual memory", obtained by a clever combination of software with the use of secondary memory. This way, each segment wasted, on average, half of a page (internal fragmentation), and the

pages would be allocated onto physical memory only if the dynamics of program execution so dictated. Primary memory held only a small-ish subset of the program's address spaces. Memory utilization was greatly improved at the cost of increased traffic between primary and secondary memory. This memory management model has been successfully employed for half a century.

The Multics [DD68, BCD72] operating system was a very ambitious design, which combined the logic simplicity of segmentation while each segment was paged in or out of primary memory on demand. This design was much too complex for the hardware available in the early 1970's.

Figure 1 shows a conceptual model of a modern memory management system [LH16]. Two address streams are generated by the processor, one from the program counter (PC) and one from the address generation unit (AGU). These two units generate virtual addresses v_i and v_d to reference instructions and data. For each stream there is a “translation buffer” (TB) between the processor and primary memory. These TBs are implemented with fast associative memory and hold a small subset of the process' memory map, which is represented by the function $f()$ in the diagram. The primary memory is referenced with physical addresses p_i and p_d .

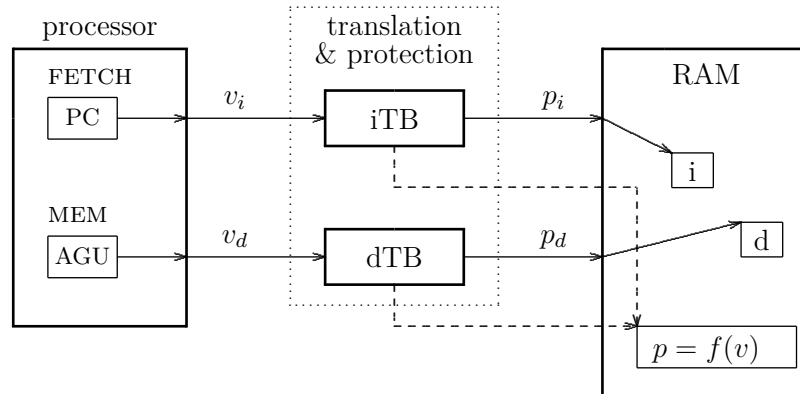


Figure 1. Conceptual model for the virtual memory system.

For a given process, its memory map holds all the valid associations between virtual and physical addresses. This map provides two important functions: (i) physical memory allocation which decouples virtual addresses from physical addresses, so that a process may use any free memory available; and (ii) security to stop a process from referencing addresses allocated to other processes. To improve performance, these two functions must be performed near the processor and the translations must be fast, hence the buffers are small.

When a process starts, its memory map is partially filled by the loader and a small subset of the instructions and data are loaded from secondary memory. As the execution progresses, a different subset of the address space may have to be loaded, and the memory map is updated accordingly. The domain of the mapping $f()$ is the whole address space, typically 2^{64} bytes in current processors, whereas the image of $f()$ is some subset of the physical address space. The main function of virtual memory is to provide the programmer with access to, essentially, infinite memory – 2^{64} is a rather large number.

2.1. Paging

The conceptual model in Figure 1 is valid for both, segmentation and demand paging. For paging, the chunks of memory are called pages, and the most popular page size is 4K bytes. Some systems provide super-pages, with sizes ranging from 64K to 1G bytes depending on processor model and operating system design. The mapping function is called “Page Table” (PT), and the size of its domain is, for pages with 2^n bytes, $|f()| = 2^{64}/2^n$. For 4K bytes, the Page Table holds up to 2^{52} elements; for 1G byte pages, the domain is a hefty 2^{34} elements. As each PT element is encoded in 8 or 16 bytes, these tables are very large. Clever hierarchical designs are used to reduce table size. Yet for large processes such as database managers, the Page Tables are proportionally large. At least some subset of the PT must always be kept in primary memory.

A virtual address is split into two fields, a virtual page number (VPN) and a displacement within the page. When the processor issues a reference, the TLB is searched for the corresponding VPN. If the translation is found (a TLB hit), the reference proceeds towards the primary memory (RAM), whence it then completes. If the address mapping is not found in the TLB (a TLB miss), then the Page Table is indexed with the VPN, and if the mapping is valid, the pair $\text{VPN} \mapsto \text{PPN}$ is loaded onto the TLB, possibly evicting some other mapping. These actions may be performed by a state machine – in Intel x86 processors – or by a short sequence of instructions – in MIPS processors.

If the mapping on the PT is not valid, it may be the case that the page is not resident in memory and has to be fetched from secondary memory (a page fault), or that the reference is to a non-mapped region of the address space (a segmentation fault), or the reference is deemed illegal (a protection fault). In any case, the operating system (OS) must take over and perform a series of operations to either recover from the fault, or to terminate the process.

2.2. Segmentation

As a minimum, a process comprises three logical segments: text, data and stack. Usually, the size of the text segment is fixed at compilation, whereas the size of the data and stack segments may vary during execution. In addition, each library dynamically linked to the process contributes with text and data segments which must be included in its address space.

A Segment Table (ST) is much smaller than a Page Table. The number of segments in a process is on the order of 100 to 1000. Even if the operating system considers open files as process segments, as in Multics, the ST would need some 2000 elements. Managing these smaller tables is far more efficient than the gargantuan Page Tables.

A virtual address is split into two fields, a virtual segment number (VSN) and a displacement into that segment. Segments can be of arbitrary sizes, hence both the VSN and the displacement are of arbitrary width. Usually, a certain number of the most significant address bits are reserved for indexing the Segment Table (or the segment buffer). The actions needed to translate a virtual address into the corresponding physical address are similar to those with paging. If the

mapping is found in the SB (a SB hit), the translated address is forwarded to the primary memory. If there is a SB miss, the segment table is searched for the physical segment number (PSN), and the mapping $VSN \mapsto PSN$ is loaded onto the SB, possibly evicting some other mapping.

If the mapping in the segment table is invalid, the three possibilities described above also apply: it may be a segment fault and the missing segment must be loaded from secondary memory, it can be a protection fault or a reference to an unmapped address. In any of these cases, the OS must take over to rectify the problem.

3. Trace Collection

We collected the execution traces on an Intel x86-64 processor, running Linux. The programs were instrumented with Valgrind 3.1.2 [NS07], and the traces generated with the Lackey tool (a Valgrind component). We wrote a program, called Trace-Tool, to read Lackey's output and store it in compressed files. Only the execution of the 'main' process is traced; the main process is the one that performs the bulk of the work for an application. Some of the applications are invoked by a shell script which sets up the environment, then starts the main process. If the main process spawns other processes, the child processes are not traced because their address spaces are disjoint from the main process'.

Our traces contain the addresses of all references to data, plus the addresses of the load-store instructions. The traces contain enough information to reproduce the memory behavior of the programs.

When the execution starts, most of the logical segments are mapped by the loader onto the process' address space: four segments for the program (text, initialized data, uninitialized data ("block started by symbol", or BSS) and the stack, plus three more segments for each library linked into the process (text, data and BSS). As the execution progresses, more segments may be added when the process requests heap space, usually via a call to `malloc`. To capture this behavior, we wrote a Python script that periodically scans the address space and records 'new' segments. The script makes use of `pmmap` and takes a new snapshot every 100ms. Trace collection can last for several hours, and taking 10 samples per second seems to be a good compromise between precision and reduced interference on the process being traced. When the Lackey process ends, the memory map is complete.

The original x86-64 traces are suitable for simulating the linear (flat) address space of paged memory systems. To generate the two-dimensional addresses for a hypothetical segmented machine, we use the process memory map to create a Segment Table (ST). Each element of the Segment Table records the segment's base address (BaseAddr), top address (TopAddr), the segment size, access rights, and a string with the segment name. Table 1 shows four lines of an ST. As a new segment is added to the ST, if the topmost 16 address bits of the segment starting address are all zero then it receives the next available segment number (VSN). If the topmost 16 address bits are non-zero, the VSN becomes the value of the topmost 16 bits. In the table, the digits are grouped into quartets to improve readability.

Once the ST is populated, the two-dimensional addresses can be generated for a linear address (LA). To find the segment to which a linear address belongs to,

Table 1. Segment table (fragment).

VSN	BaseAddr	TopAddr	Size	Rights	Name
0x0000	0x0040.0000	0x0061.0fff	2M	r-x--	lsblk
0x0001	0x0061.1000	0x0061.1fff	4K	r----	lsblk
0x0002	0x0061.2000	0x03ff.ffff	58M	rw---	lsblk
0x0003	0x0400.0000	0x0402.5fff	152K	r-x--	ld-2.23.so

the address is compared to the starting address and size of each ST element. If the address is in range of a segment ($\text{BaseAddr} \leq \text{LA} \leq \text{TopAddr}$), then the VSN replaces the topmost 16 address bits of the linear address, thus generating a segmented address. For instance, if the linear address is 0x0061.10ff, then it belongs in segment 0x0001 and the segmented address is 0x0001.0000.0061.10ff.

4. Reference Counts

We collected traces from six real applications. In what follows we briefly describe the application, the data sets, and the reference counts obtained from the traces. The programs were simulated for 10 billion instructions, and each simulation yields traces with roughly 13-15 million memory references.

MySQL SysBench [Ubu17] performs random operations on a table with 1,000,000 lines, implemented as an MySQL database. The operations are distributed across eight internal MySQL threads, and run during 480 seconds. Each thread's stack is held in a separate segment, and there is an anonymous segment below the bottom of the stack to catch references outside of the allocated area.

Firefox The web browser was traced while opening a dynamic web page pointed to by a command line argument. Most of the segments mapped are dynamic libraries but just a few of them are used to display the (relatively simple) web page. There are 91 segments labeled as 'cache', with sizes ranging from 1 to 256 pages.

Python Python was traced executing pip, to fetch and install tools used in the development of Python programs.

Tomcat Tomcat is a Java Servlet Container and web server. The server was traced while serving Tomcat's initial web page, then shutting down. There are 319 segments labeled as anon.

QEMU + FreeDOS QEMU is a hardware virtualization platform. The traces were obtained while emulating an Intel x86 processor, with 32 Mbytes of RAM, hard disk, and text console. FreeDOS was run on this virtual machine from the boot sequence until the first prompt. At the prompt, FreeDOS was shutdown from the text console. There are 242 segments labeled as anon.

LibreOffice The trace records the writer application reading a 314 Kbytes Microsoft Word document. There are 91 segments labeled as 'cache', with sizes from 1 to 256 pages, and 163 anonymous segments.

Table 2 shows the number of libraries linked into the application, the segments mapped, and the segments actually referenced. Many segments, mostly from libraries, are mapped but never referenced, as the libraries' functions are not invoked.

Table 2. Libraries, number of segments, usage ratio.

application	libs	segs	used	ratio
MySQL	75	310	151	48%
Firefox	485	979	633	64%
Python	111	162	115	71%
Tomcat	90	416	272	65%
QEMU	601	845	567	67%
LibreOffice	785	1,085	830	76%

All libraries, with two exceptions, map a 2 Mbytes segment with no ‘r,w,x’ permissions, between the executable and the read-only segment. This buffer zone is used to catch any references to addresses below the segment with initialized variables, which is copy-on-write and remains untouched unless its contents are changed by the process. The two exceptions are the loader `ld-2.23.so`, and the locale’s tables `libc.mo` – these libraries do not make use of the 2 Mbyte buffers.

Memory allocated as heap is mapped on segments labeled anon, and have r,w permissions. Some anon segments have no permission bits set, and probably are heap regions that were allocated but never touched during execution – the traces show no references to these segments.

Figure 2 shows a plot of segment frequency versus segment size for the applications, and the average for the six programs. The most popular segment is a single 4 Kbytes page; the second most popular size is 2 Mbytes. Segment sizes were rounded up to the nearest power of two. The numbers shown are for allocated segments, and not for segments actually referenced – Table 2 indicates that some 20 to 50% of the segments are allocated yet never touched during execution.

5. A Comparison of Paging and Segmentation

Three TLB/SB configurations were simulated, with 32, 64 and 128 entries. All buffers are fully associative and were simulated with perfect least-recently used (LRU) replacement. Except for the LRU replacement, the buffers are of a complexity similar to those available on current x86-64 processors. By ‘similar’ we mean the simulated buffers are of similar capacity to the last level TLB of x86-64 processors. As we are merely counting references, it is unnecessary to simulate the rather complex TLB hierarchies of existing processors.

Table 3 displays the number of references on each trace (refs). The columns under the heading TLB show the miss rates for the three TLB organizations, while those under the heading SB show the miss rates for the three SBs. Some of the applications encounter a number of misses that are less than one in 10,000 references – typically, these are buffers that accommodate all the active segments referenced during execution, and the few misses observed are compulsory misses.

The miss rates shown in Table 3 are not surprising. For the demand paging systems, the 32 entry TLB covers just $32 \times 4096 = 128$ Kbytes, and the 128 entry covers 512 Kbytes. Given that the applications have working sets larger than the TLB reach, high miss rates are to be expected.

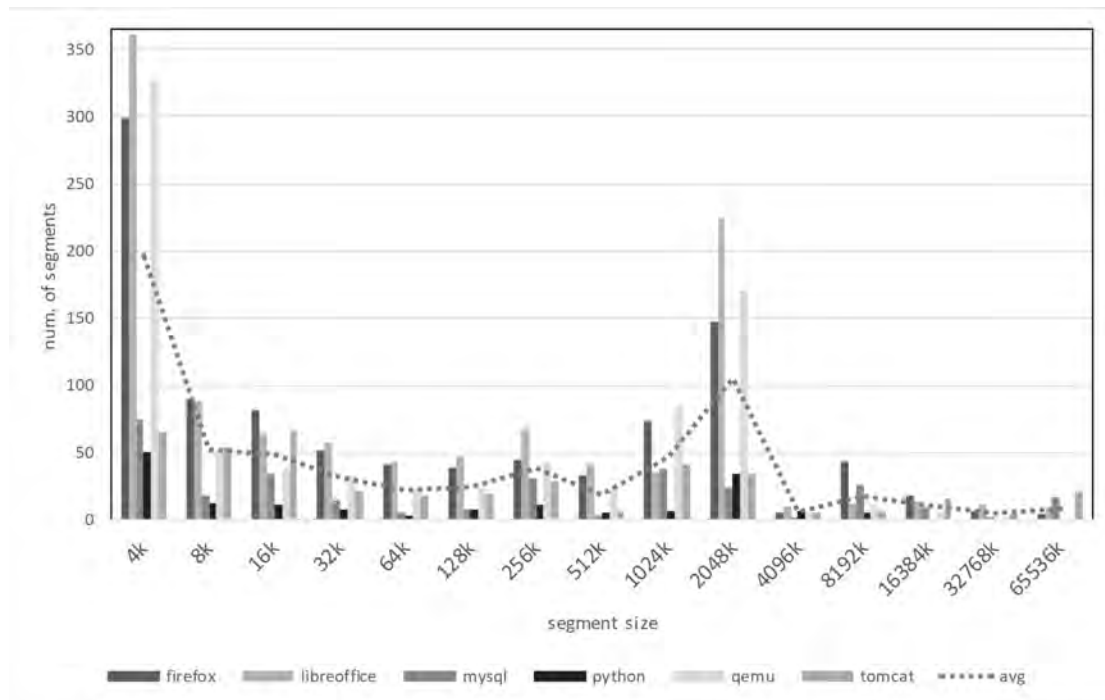


Figure 2. Segment size distribution.

Suppose that a RAM reference takes 200 cycles (this is optimistic), that the PT is organized in four levels (realistic), and that the PT lines are never present in the L2 or L3 TLBs (pessimistic). Under these assumptions, each TLB miss costs some 800 cycles. Hornyack, Basu, *et al.* [HCG⁺15, BGC⁺13] present measurements for the percentage of execution cycles spent walking the Page Table on TLB misses for large scale applications. They found that the applications spend from 2 to 10% of the execution cycles filling the TLB from the PT. One graph application with bad locality spends over 50% of its cycles on PT walks to refill the TLB.

The miss rates for the segmented memory are 100–1000× smaller than those for TLBs of similar organization, and this is also not surprising. As the number of segments is much smaller than the number of virtual pages, the miss ratios are bound to be better. Furthermore, all processes have at least one segment of 8–128 Mbytes that holds a large portion of the actively referenced addresses.

Table 3. TLB and SB reference counts and miss rates (#refs×10⁶).

applic.	refs	TLB			SB		
		32	64	128	32	64	128
MySQL	15,272	0.718	0.152	0.058	0.0033	0.0006	$1 \cdot 10^{-8}$
Firefox	14,649	0.354	0.137	0.053	0.0251	0.0178	0.0129
Python	14,499	0.855	0.193	0.032	0.0001	$2 \cdot 10^{-8}$	$8 \cdot 10^{-9}$
Tomcat	14,512	0.258	0.066	0.018	0.0011	$3 \cdot 10^{-7}$	$3 \cdot 10^{-8}$
QEMU	13,618	0.278	0.036	0.004	0.0146	0.0137	0.0033
LibreOf	13,509	0.216	0.073	0.029	0.0529	0.0353	0.0267

For some applications, such as Firefox, LibreOffice and QEMU, dozens of libraries are linked into the address map yet their segments are never referenced, thus improving locality, hence yielding better miss ratios. In a segmented system, these libraries would be mapped on the Segment Table but allocation of physical memory would be postponed until the segment is referenced, further improving performance.

6. Design Considerations

The diagram in Figure 3 shows a ‘traditional’ organization for a segment buffer [LH16]. The virtual address is split into the virtual segment number (VSN) and a displacement. The VSN is associatively compared to the virtual tags in the SB; in case of a hit, the displacement is added to the physical address of the segment base, and the physical address is posted to memory. This completes the translation phase.

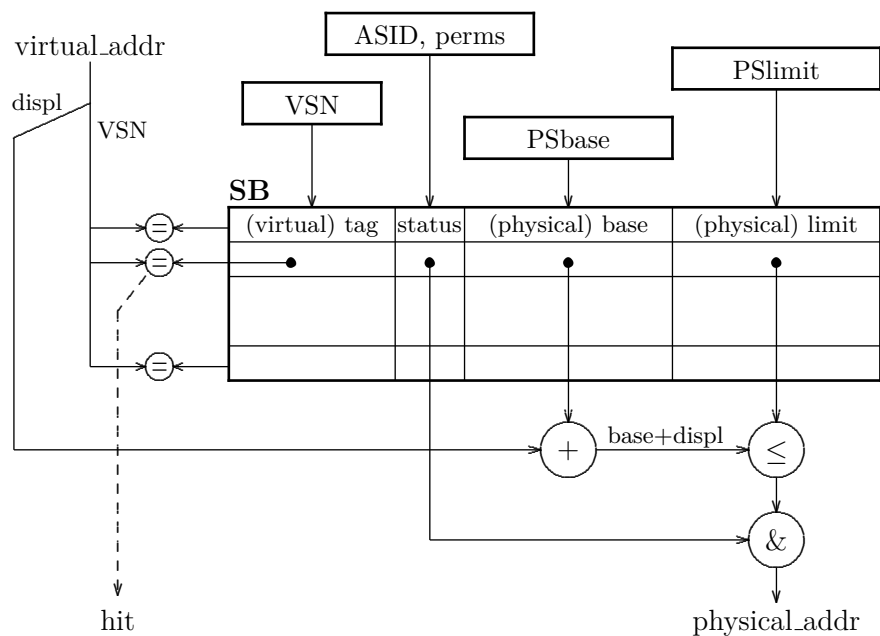


Figure 3. Segment Buffer.

The resulting physical address is then compared to the segment limit; if the comparison results in a segment violation, the exception is signaled to the processor. Some form of process, or address space, identification is kept in the ASID field, and the ASID is also checked for illegal references. This completes the reference validation phase, or the protection phase.

Notice that all addresses are 64 bits wide, hence the programming model does not need to change and the segmented memory is hidden from the programmer, as it should [LH16]. The management of logical segments is performed by the linker, the segment table is setup by the loader and is maintained by the OS. To improve physical memory utilization, the compiler could allow the programmer to give an estimate for segment size, or to declare a segment as ‘elastic’ if it is known that it can grow or shrink during execution.

The address translation ($VA \mapsto PA$) must be performed as early as possible so the memory reference may complete without delay. With demand paging, the translation is obtained by simply indexing the TLB, whereas with segmentation two additions must be performed, the first to compute the PA ($PSbase + displ$), and the second to compare the PA to the segment limit. The first addition must be performed early and the adder is on the critical path. As a first approximation, the average memory access time could be elongated by 1–2% without significant performance loss when compared to a paging system. Recall that TLB-page table walks may cost 2–10% additional execution cycles, according to Hornyack’s measurements.

Any additional delay introduced by translation may be compensated by the reduction in traffic at the processor-RAM interface: as the SB miss ratios are 100–1000× smaller, there is less demand for RAM bandwidth. Also, the smaller segment tables incur in less management overhead, which translates into thousands of instructions, and secondary memory accesses, *not* performed to manage a hierarchical memory mapping table.

The adder that compares the PA to the segment limit can be taken off the critical path because addressing/protection exceptions do not need to be signaled until the instruction is about to commit, and that is late in the pipeline. The same applies to access permissions.

There is one additional condition that must be verified. If the programming model is to remain unchanged, then the the PA must also be checked against the segment base. A programming error (or malice) could generate a ‘negative’ VA, which when added to the segment base could result in a PA that is smaller than the physical segment base, thus generating an addressing fault. This comparison can also be performed late in the pipeline and kept off the critical path. The third adder is not shown in Figure 3.

External fragmentation may be a problem, albeit not so serious because memory is ‘infinite’. Trading physical memory utilization for shorter execution time may be worthwhile for time-critical applications. Internal fragmentation is a problem in systems that use super-pages as their sizes are either too large for the smallish segments (libraries), or too small for the large heap areas or in-memory tables [HCG⁺15]. In systems which support long running applications, the cost of relocating segments in order to free physical memory may be a small fraction of the total execution time. There is a relatively inexpensive solution to the relocation dilemma: a DMA-like engine can steal bus cycles to move inactive segments onto more crowded regions in physical memory, thus freeing unused space [LH16].

7. Related Work

Single address space systems [KCE92] are an alternative to segmentation: all processes are mapped on a single virtual address space. If the physical memory is large enough, address translation may be unnecessary. Protection is enforced at the level of logical segments, or with paging. In either case, a structure similar to a TLB or an SB must be used to capture the illegal references.

Basu *et al.* [BGC⁺13] introduce the “direct segment” to support *one* arbitrarily large segment while retaining demand paging for the rest of the memory

space. Their solution is justified by the way large-scale applications use memory: one very large chunk of memory is allocated at initialization, then the application itself manages the allocation of smaller chunks for buffers, tables, etc. The direct segments imply in changes the programming model, by adding one programmer visible segment, yet is less disruptive than variable size super-pages, from the micro-architecture point of view.

Pham *et al.* [PBEL14] state that applications typically exhibit ‘contiguous’ spatial locality in which tens of consecutive virtual pages are mapped to consecutive physical pages. This behavior generates many instances of contiguous page table entries (PTE), though typically not enough for large page generation. Pham proposes a low-overhead, multi-granular TLB organization that exploits PTE clustering on a hierarchical TLB architecture with enhancements regarding replacement policies and prefetching to eliminate 46% of L2 TLB misses. Our results show that segmentation may provide the same benefits in exploiting spatial locality, with less complex hardware and software implementation.

Hornyack *et al.* [HCG⁺15] present strong evidence that several large-scale applications would perform better if the memory allocation were segment-based instead of the current paging systems. These applications spend a great deal of time handling TLB misses and the performance loss ranges from a few percent of the execution cycles to 58% for some workloads with poor locality. Hornyack also shows that server-class memory-hungry applications use “virtual memory areas” which represent an item of code (or data) in a contiguous region of memory that spans from one to several contiguous virtual pages. They claim that, rather than using fixed size pages, hence large page tables, a segmented system would substantially reduce the amount of state needed to keep the protection and mapping information. While our results are in agreement with their findings, we extend Hornyack’s work by characterizing program behavior in terms of segmentation. Our work, and theirs likewise, indicate that segmentation can bring substantial performance gains in current systems, and specially for the systems of the coming decade.

Virtual memory was devised to hide the idiosyncrasies of primary and secondary memories from the programmer, by providing a flat address space, and a programming model that abstracted away all details regarding the timing of memory references, and the capacity of the installed memory. Large pages are an interesting idea but represent an unwelcome break with the original, neat and tidy, programming model provided by virtual memory. Large pages expose the physical memory to the programmer and introduce more complexity to a rather intricate component of the operating system.

Virtual memory systems were devised in an era of scarcity, and have served us well since the late 1960s. Petabyte physical memories are becoming affordable and the first products supporting non-volatile memory (NVRAM) are reaching the market. In [LH16] we examine several changes to the way operating systems might be designed and built, when one considers the effects of solid state disks (SSDs) and NVRAM on system performance and complexity.

8. Conclusion

In the next decade we may see systems equipped with 2^{64} bytes of physical memory, possibly with NVRAM comprising a large fraction of the primary memory. We believe serious consideration should be given to a complete re-design of our long serving and trusted demand paging virtual memory systems. To support such a bold claim, we characterized the behavior of real applications with respect to the virtual-to-physical translation buffers. We performed trace driven simulations of translation buffers of similar complexity for demand paging systems (TLBs) and segmented systems (SBs).

Our results indicate that the miss rates for SBs are two to three orders of magnitude smaller than for TLBs. This is not surprising since the number of logical segments is much less than the number of logical pages for any non-trivial application. The management of the smaller segment tables is bound to be less expensive than the huge page tables, yielding further improvements in performance. We show how the SBs can be designed so the segmented virtual memory remains hidden from the programmer and compiler writer.

Hard data notwithstanding, people in our community have grown so accustomed to demand paging that the simpler and more efficient alternative, segmentation, is commonly discarded offhand.

References

- [BCD72] A Bensoussan, C T Clingen, and R C Daley. The Multics virtual memory: Concepts and design. *Comm of the ACM*, 15(5):308–318, May 1972.
- [BGC⁺13] A Basu, J Gandhi, J Chang, M D Hill, and M M Swift. Efficient virtual memory for big memory servers. In *ISCA'13: 40th Intl Symp on Computer Arch*, pages 237–248, 2013.
- [DD68] Robert C Daley and Jack B Dennis. Virtual memory, processes, and sharing in MULTICS. *Comm of the ACM*, 11(5):306–312, May 1968.
- [Den70] Peter J Denning. Virtual memory. *ACM Computing Surveys*, 2(3):153–189, Sep 1970.
- [HCG⁺15] P Hornyack, L Ceze, S Gribble, D Ports, and H M Levy. A study of virtual memory usage and implications for large memory. In *Proc Workshop on Interactions of NVM/FLASH with Operating Systems and Workloads*, 2015.
- [HP90] John L Hennessy and David A Patterson. *Computer Architecture: A Quantitative Approach*. Morgan Kaufmann, 1st edition, 1990. ISBN 1558600698.
- [JNW08] B L Jacob, S W Ng, and D T Wang. *Memory Systems: Cache, DRAM, Disk*. Morgan Kaufmann, 2008. ISBN 0123797513.
- [KCE92] E J Kolding, J S Chase, and S J Eggers. Architecture support for single address space operating systems. In *ASPLOS'92: 5th Intl Conf on Arch Support for Progr Lang and Oper Sys*, pages 175–186, 1992.
- [KELS62] T Kilburn, D B G Edwards, M J Lanigan, and F H Sumner. One-level storage system. In *IRE Trans on Electronic Computers*, EC-11, pages 223–235, 1962.

- [LH16] Lauri P Laux Jr and R A Hexsel. Back to the past: Segmentation with infinite and non-volatile memory. In *WSCAD-SSC'16: XVII Workshop em Sistemas Computacionais de Alto Desempenho*, pages 278–289, 2016.
- [NS07] Nicholas Nethercote and Julian Seward. Valgrind: A framework for heavyweight dynamic binary instrumentation. In *PLDI'07: Proc 28th ACM SIGPLAN Conf on Programming Language Design and Implementation*, pages 89–100, 2007.
- [PBEL14] B Pham, A Bhattacharjee, Y Eckert, and G H Loh. Increasing TLB reach by exploiting clustering in page translations. In *HPCA'14: 20th Int Symp on High-Performance Comp Arch*, pages 558–567, 2014.
- [Ubu17] Ubuntu. SysBench – a modular, cross-platform and multi-threaded benchmark tool, Aug 2017.

Método Rápido de Agrupamento de Vértices para Detecção de Comunidades em Redes Complexas de Larga-escala

Gustavo S. Carnivali¹, Alex B. Vieira², Paulo A. A. Esquef¹, Artur Ziviani¹

¹ Laboratório Nacional de Computação Científica (LNCC)

² Departamento de Ciência da Computação – Universidade Federal de Juiz de Fora (UFJF)

{carnivali, pesquef, ziviani}@lncc.br, alex.borges@ufjf.edu.br

Abstract. *This paper tackles the problem of community detection in large-scale graphs. In the literature devoted to this topic, an iterative algorithm, called Louvain Method (ML), stands out as an effective and fast solution for the problem. However, the first iterations of the ML are the most costly. To overcome this, this paper introduces a fast algorithm for vertex grouping (MRV), which can be a way faster option than the first iterations of the ML, yet similarly effective. The performance of the proposed multistage scheme for community detection, called MRV+ML, is confronted against that of the ML using real-world networks from small to large sizes. Experimental results show that, for large-scale graphs, the MRV+ML detects communities with mean modularity about 3% lower than the ML, but with a mean reduction of about 42% in computation time.*

Resumo. *Este artigo reporta resultados de uma investigação sobre o problema de detecção de comunidades em redes complexas. Na literatura dedicada a esse assunto, um algoritmo iterativo, denominado Método de Louvain (ML), se destaca como opção eficaz e rápida para o problema. Entretanto, as primeiras iterações do ML são sua parte mais custosa. Neste artigo, propõe-se um Método Rápido de Agrupamento de Vértices (MRV) em grafos como uma opção mais rápida do que as primeiras iterações do ML. Através de experimentos envolvendo redes grandes do mundo real, demonstra-se que o sistema proposto MRV+ML identifica comunidades com modularidade similar ao ML (redução média menor de 3%), mas com redução média de aproximadamente 42% no tempo de execução.*

1. Introdução

Um grafo é uma estrutura composta por um conjunto de objetos que podem estar conectados por arestas indicando a existência de uma relação entre um par de objetos. Convenientemente, grafos podem então ser utilizados para representar várias redes complexas do mundo real [Bondy et al., 1976]. Grafos de grande porte podem ser criados na representação de grandes serviços como Facebook e Twitter ou em outros que são comuns em domínios como o WWW [Kwak et al., 2010, Mislove et al., 2007]. De forma similar, grandes grafos encontram aplicação na representação de redes complexas de larga-escala em diversas áreas do conhecimento.

Em diversas redes complexas reais, a distribuição de arestas entre os vértices apresenta-se de forma altamente heterogênea. Isso leva, nesses casos, a uma alta concentração de arestas entre agrupamentos de vértices e uma baixa concentração de arestas

entre esses agrupamentos. Essa característica de redes reais é tipicamente chamada de estrutura de comunidades. Comunidades, também chamadas de agrupamentos, são grupos de vértices que provavelmente compartilham algumas propriedades comuns ou desempenham papéis semelhantes no grafo em estudo. Nesse contexto, considerando a análise de grafos que representam redes complexas reais, um problema comum é a detecção de comunidades [Fortunato, 2010, Harenberg et al., 2014, Khan e Niazi, 2017, Zhao, 2017], que será o alvo principal de investigação deste artigo.

1.1. Detecção de comunidades em redes complexas

O problema de detecção de comunidades consiste em encontrar, em um grafo, agrupamentos de vértices que possuam uma ou mais características em comum. Uma dessas características pode ser a vizinhança em comum entre vértices, que pode ser dada pelo número de arestas que os vértices de um mesmo agrupamento (i.e., uma comunidade) compartilham entre si. Várias métricas podem ser utilizadas para avaliar os agrupamentos e as semelhanças existentes em cada comunidade [Chakraborty et al., 2017]. Nesse contexto, uma métrica muito utilizada para avaliar a qualidade das comunidades é a modularidade Q [Girvan e Newman, 2002, Newman, 2006]. Intuitivamente, $-1 < Q < 1$ é uma medida do quão densamente conectados entre si estão os vértices do mesmo tipo ou classe. O valor de Q se torna mais positivo (resp. negativo) quanto maior (resp. menor) for o número de arestas conectando vértices de uma mesma classe, se comparado com uma distribuição aleatória de arestas entre os mesmos vértices.

O problema de detecção de comunidades, pode ser tratado como um problema de maximização da modularidade das partições encontradas em um grafo. Formalmente, a modularidade é dada por [Girvan e Newman, 2002, Newman, 2006]:

$$Q = \frac{1}{2m} \sum_{ij} \left(A_{ij} - \frac{k_i k_j}{2m} \right) \delta(c_i, c_j), \quad (1)$$

onde m é o número de arestas do grafo, $A_{ij} = 1$ se os vértices i e j forem conectados e $A_{ij} = 0$ caso contrário (i.e., A é a matriz de adjacência do grafo), k_i é o grau do vértice i (ou seja, o número de arestas incidentes sobre i), c_i é a classe ou tipo do vértice i e a função $\delta(c_i, c_j)$ é o δ de Kronecker, ou seja, $\delta(c_i, c_j) = 1$ se os vértices i e j forem da mesma comunidade ou tipo, isso é, $c_i = c_j$; e $\delta(c_i, c_j) = 0$ se os vértices i e j não forem da mesma comunidade ou tipo.

Um levantamento comparativo de diversos métodos de detecção de comunidades pode ser encontrado em [Lancichinetti e Fortunato, 2009, Leskovec et al., 2010]. Dentre eles, o chamado Método de Louvain (ML) [Blondel et al., 2008] destaca-se por identificar comunidades em grafos muito grandes, de modo rápido e eficaz (i.e., com alta modularidade). Um breve descritivo do ML é apresentado na próxima subseção.

1.2. Método de Louvain [Blondel et al., 2008]

O ML para detecção de comunidades em grafos emprega uma heurística iterativa baseada na maximização da modularidade. Inicialmente, o ML atribui cada vértice do grafo a uma comunidade distinta. No primeiro estágio, para todo vértice i do grafo, o ML considera a remoção de i de sua comunidade para a comunidade de cada vizinho j de i . De

forma gulosa, o ML escolhe mover i para a comunidade do vértice vizinho j que leve ao maior ganho em modularidade. Cabe notar que a variação de modularidade produzida pela mudança considerada de i para um vizinho j pode ser computada de forma local e eficiente [Blondel et al., 2008]. Esse processo é repetido para cada vértice até que não haja possível incremento na modularidade, completando a primeira etapa do ML.

O segundo estágio do ML gera um novo grafo (reduzido) onde os novos vértices são as comunidades encontradas no passo anterior. A saída do segundo estágio é submetida à entrada do primeiro estágio e assim por diante, enquanto houver ganho de modularidade nas comunidades encontradas.

Segundo [Blondel et al., 2008], o ML possui complexidade computacional $\mathcal{O}(n \log n)$, onde n é o número total de vértices do grafo. O ML começa com um grafo de n vértices e, ao fim de cada rodada de seus dois estágios, a tendência é obter grafos com um número menor de vértices (comunidades). Assim, as primeiras rodadas do ML concentram a maior parte do custo computacional.

Apesar da baixa complexidade do ML, devido aos grandes grafos que podem ser encontrado atualmente, é possível encontrar na literatura propostas que visam otimizar o ML em relação ao seu tempo de execução. Em [Ozaki et al., 2016], os autores analisaram os processos do ML que implicavam algum desperdício temporal e propuseram soluções simples para otimizá-los. Há trabalhos que aceleram o ML através de implementações que usam computação paralela [Bhowmick e Srinivasan, 2013, Fazlali et al., 2017]. Em [Fortunato e Barthelemy, 2007], os autores investigam os limites de resolução do ML na detecção de comunidades, uma vez que a maximização da modularidade não favorece à detecção de pequenas comunidades. Já em [Aldecoa e Marín, 2013], os autores avaliam o desempenho de 18 métodos de detecção de comunidades e evidenciam a sub-otimalidade dos resultados quando a modularidade é maximizada. Eles propõem então um novo parâmetro global para avaliar a qualidade de partições do grafo, chamada de surpresa [Aldecoa e Marín, 2011], cuja maximização obtida pela combinação de múltiplos algoritmos alcança melhores resultados.

Outra classe de métodos encontrada na literatura se utiliza da estratégia de aplicar algoritmos rápidos ao grafo de entrada, entregando ao ML um grafo mais simples e, por conseguinte, reduzindo o tempo de execução da tarefa de detecção de comunidades. Em [Satuluri et al., 2011], o grafo original é reduzido a partir da eliminação de arestas que possivelmente fariam parte de uma mesma comunidade, antes de ser submetido ao ML. O procedimento reduz o tempo de execução da detecção de comunidades e pouco altera a qualidade das comunidades encontradas. Entretanto, o método se torna ineficiente em grafos com baixo grau médio de conexões (número pequeno de arestas em relação ao número de vértices). Outro método, apresentado em [Peng et al., 2014], gera um corte no grafo a partir de seu k -core, produzindo subgrafos do grafo original com todos os vértices que tenham pelo menos k conexões. Esses subgrafos são aplicados a um algoritmo de detecção de comunidades (como o ML). Após definidas as comunidades dos subgrafos, eles são reconectados apropriadamente.

1.3. Método Proposto

O método proposto neste artigo tem objetivos similares aos da última classe de técnicas analisadas na Seção 1.2: modificar o grafo original e entregar ao ML um grafo de tamanho

reduzido, visando a reduzir o tempo de execução do processo de detecção de comunidades, sem perda significativa na modularidade das comunidades identificadas. Propõe-se um Método Rápido de Agrupamentos de Vértices (MRV), que faz um papel similar à primeira rodada (mais custosa) do ML, i.e., de gerar um grafo com comunidades bem estabelecidas próximas da estrutura do grafo original. O ML é acelerado pois sua primeira fase é substituída por uma que priorize a velocidade e não a qualidade, deixando a valorização da qualidade para as etapas seguintes do ML. A aplicabilidade do MRV, tal como proposto neste artigo, é investigada em cenários restritos a grafos não-ponderados, i.e., que não possuem arestas ou vértices pontuados; grafos não-direcionados e com geração de comunidades disjuntas, i.e., comunidades sem sobreposição de vértices.

O MRV consiste em um algoritmo iterativo onde um vértice é inicialmente atribuído a uma comunidade. De modo iterativo, cada vizinho a esse vértice é pontuado de acordo com uma medida de seu grau de conectividade com os vértices pertencentes à comunidade. É qualificado para integrar a comunidade do vértice de referência o vizinho com maior pontuação acima de um limite mínimo, controlado por uma variável chamada de K , com valor definido experimentalmente (detalhes na Seção 3). Em caso de empate, i.e., mais de um vizinho com a maior pontuação, sorteia-se aleatoriamente um deles para ingressar na comunidade. O vizinho recém-integrado à comunidade passa a ser o vértice de referência e as iterações continuam até não haver mais vizinhos que se qualifiquem para ingressar na comunidade. Nesse ponto, a comunidade é fechada, uma nova comunidade é criada e um vértice ainda sem comunidade é sorteado para ingressá-la. As iterações continuam dessa forma até que todos os vértices do grafo sejam associados a comunidades, que são transformadas em vértices de um grafo reduzido.

Na investigação reportada, inicialmente é avaliado o desempenho do MRV em função de K e, após isso, o desempenho do MRV+ML em relação ao ML. Para a avaliação são utilizados grafos associados a redes reais com número de vértices entre aproximadamente 1 mil e 5 milhões. A investigação de K se torna necessária pois seu valor pode alterar indicadores como o tempo de execução e número de grupos (comunidades). Os testes mostraram que K pode ser ajustado no MRV+ML para reduzir substancialmente o tempo de execução do método de detecção de comunidades, ao custo de uma leve redução na modularidade das comunidades encontradas. Os resultados experimentais mostram que, para grandes redes, a proposta MRV+ML produz uma redução de 45% no tempo médio de execução e perdas médias menores de 3% em qualidade, no sentido da modularidade.

1.4. Organização do Artigo

Além desta introdução, o artigo é organizado da seguinte forma. Uma descrição detalhada do MRV é apresentada na Seção 2, que também inclui a metodologia usada nos experimentos. Os resultados experimentais da avaliação de desempenho do MRV e da composição MRV+ML são reportados na Seção 3. Por fim, as conclusões da investigação feita e possíveis desdobramentos futuros são resumidos na Seção 4.

2. Método Rápido de Agrupamento de Vértices (MRV)

Nesta seção será apresentado em detalhes o método proposto neste artigo. Os objetivos do MRV e um breve resumo de sua heurística foram descritos na Seção 1.3.

Algoritmo 1: Método Rápido de Agrupamento de Vértices (MRAV)

```

1  início
2      Todos os vértices são rotulados como Desagrupado
3      Enquanto (Existir vértice definido como Desagrupado) {
4          Novo grupo G iniciado
5          Pontuacao de todo vértice Desagrupado é zerada
6          Limite = 0
7          Enquanto () {
8              Vertices[] = conjunto de vértices Desagrupado com maior
              Pontuacao
9              Vertice = vertice aleatório de Vertices
10             Se (Pontuacao do Vertice  $\geq$  Limite) {
11                 Adiciona Vertice à G
12                 Vertice é rotulado como Agrupado
13                 Limite é aumentado (Equação 4)
14                 Para Todo (I vizinho de Vertice) {
15                     Pontuacao de I é aumentada (Equação 3)
16                 }
17             }
18             Se Nao {
19                 Encerra grupo G
20             }
21         }
22     }
23 fim

```

O pseudocódigo do MRAV pode ser visto no Algoritmo 1. Como entrada, é recebido um grafo e o algoritmo retorna um conjunto de comunidades (grupos) populadas por vértices. Todo vértice começa rotulado como *Desagrupado* (Linha 2) e com *Pontuacao* 0 (Linha 5). Um novo grupo vazio *G* é criado (Linha 4). A variável *Limite* é iniciada com 0 (Linha 6). As variáveis *Pontuacao* e *Limite* estão relacionadas ao critério (Linha 10), que qualifica um vértice a ingressar no grupo. A pontuação dos vértices candidatos a ingressar no grupo é feita pelo *loop* interno (Linha 7).

A variável *Vertices*[] (Linha 8) vai conter, tipicamente, o vértice (e pontuação associada) desagrupado que obteve a maior das pontuações, dentre aqueles que são vizinhos ao vértice recém alocado ao grupo. Na eventualidade de empate na pontuação máxima, i.e., *Vertices*[] terá mais de um vértice candidato, um dos quais será sorteado (Linha 9) e atribuído à variável *Vertice*. A exceção se dá quando da criação de um grupo, quando a pontuação de todos os vértices desagrupados é zerada (Linha 5). Nesse caso, como *Vertices*[] contém todos os vértices desagrupados, a primeira passagem pelas Linhas 8 e 9 implica fazer uma seleção aleatória dentre os vértices desagrupados.

O vértice selecionado na Linha 9 será incluído no grupo *G* (Linha 11) e rotulado como tal (Linha 12), caso passe o teste da linha 10. Ou seja, o vértice selecionado é incluído no grupo *G* e rotulado como agrupado se sua *Pontuacao* for maior ou igual

ao valor de *Limite*. Caso contrário, o grupo é encerrado e, se ainda houver vértices desagrupados, um novo grupo é criado.

Neste ponto, é oportuno esclarecer o papel conceitual da comparação entre as variáveis *Pontuacao* e *Limite*, assim justificando as linhas 13 e 15 do pseudocódigo. Uma medida razoável do nível de conectividade de um vértice desagrupado B com um vértice agrupado A pode ser dada por:

$$\mathcal{C}(B, A) = \frac{Viz(A, B) + 1}{Grau(B)}, \quad (2)$$

onde $Viz(A, B)$ é o número de vizinhos comuns entre A e B , sendo $Grau(B)$ o número total de vizinhos de B . A soma de uma unidade no numerador da equação é feita para refletir a própria conexão entre A e B , nesse caso, $Viz(A, B) + 1$ pode assumir valores entre 0 e $Grau(B)$ ($0 \leq Viz(A, B) + 1 \leq Grau(B)$) tornando $\mathcal{C}(B, A)$ uma porcentagem dos vizinhos de B . Considere o conjunto de valores de $\mathcal{C}(B, A_j)$, com $j = 1, 2, \dots, N$, computados para cada vértice A_j já agrupado até a iteração N . Se a *média* dos valores de $\mathcal{C}(B, A_j)$, i.e., $\bar{\mathcal{C}} = N^{-1} \sum_{j=1}^N \mathcal{C}(B, A_j)$, for superior ou igual a um limite mínimo K , arbitrariamente prescrito para o nível de conectividade, então B se qualifica a ingressar no grupo.

Graças à natureza iterativa do algoritmo, em que o grupo vai sendo gradativamente populado com um vértice por vez, optou-se por uma implementação prática do critério que não envolve o cômputo de médias. Na iteração N , pode-se de modo equivalente comparar $N\bar{\mathcal{C}} = \sum_{j=1}^N \mathcal{C}(B, A_j)$ com NK . A soma dos valores de $\mathcal{C}(B, A_j)$ é calculada de forma recursiva (Eq. 3) e armazenada na variável *Pontuacao* (Linha 15). De modo análogo, NK é obtido de forma recursiva (Eq. 4) e armazenado na variável *Limite* (Linha 13).

Pelo arrazoado acima, a pontuação de um vértice desagrupado B , candidato a ingressar no grupo é dada pela seguinte dinâmica recursiva:

$$Pontuacao(B) \leftarrow Pontuacao(B) + \frac{Viz(A, B) + 1}{Grau(B)}, \quad (3)$$

com condição inicial $Pontuacao(B) = 0$, forçada após a criação de um grupo (Linha 5). De modo análogo, a dinâmica recursiva da variável *Limite* é dada por:

$$Limite \leftarrow Limite + K, \quad 0 \leq K \leq 1, \quad (4)$$

também com condição inicial $Limite = 0$ (Linha 6).

O vértice recém-integrado ao grupo (Linha 12) passa a ser o vértice de referência para efeito do levantamento dos novos vértices candidatos a ingressar o grupo, i.e., aqueles desagrupados e vizinhos ao vértice de referência. A pontuação desses vértices candidatos é atualizada segundo a Equação 3 (Linha 15). Vale notar que a atualização da pontuação dos vértices candidatos implica na modificação da variável *Vertices[]*, da qual um candidato com a maior pontuação será selecionado (Linha 9) e avaliado quanto à qualificação para ingressar no grupo (Linha 10). Como mencionado anteriormente, se o melhor candidato não atender ao critério de ingresso no grupo, esse é fechado (Linha 19). Outro grupo é criado se ainda houver vértices desagrupados (Linha 3).

A complexidade temporal do algoritmo é dada pela presença de dois *loops*: (i) um *loop* principal que percorre todos os vértices do grafo; e (ii) um *loop* secundário que percorre todos os vizinhos de cada vértice. Como em cada vértice todos os seus vizinhos são percorridos, a complexidade é dependente do número de vizinhos que os vértices possuem. Logo, o algoritmo MRAV possui complexidade computacional de $\mathcal{O}(e)$ onde e é o número total de arestas do grafo.

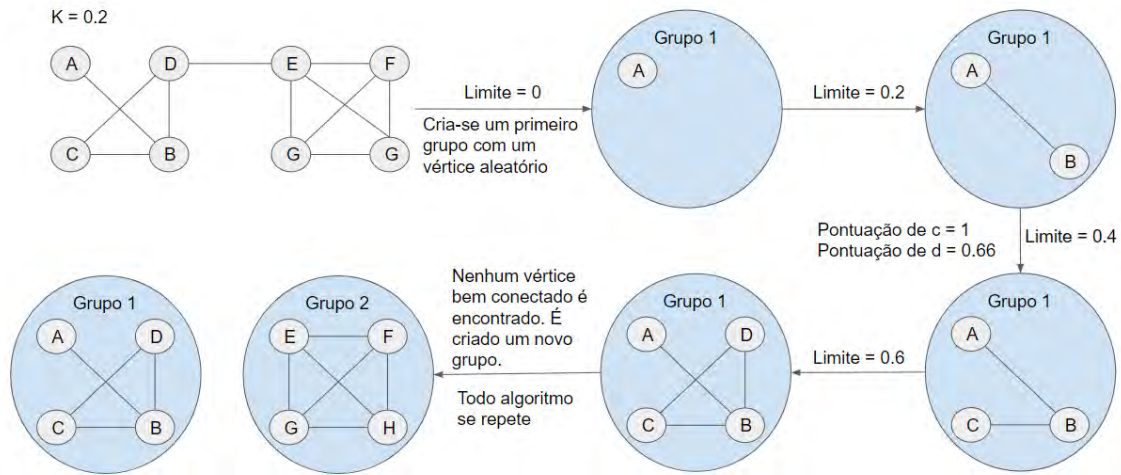


Figura 1. Exemplo ilustrativo das fases do algoritmo MRAV com $K = 0.2$.

Um exemplo visual das iterações do MRAV (com $K = 0.2^1$) sobre um grafo simples pode ser visto na Figura 1. Inicialmente, um novo grupo é iniciado (Grupo 1) e um vértice é aleatoriamente alocado a ele (vértice A). Como B é o único vizinho de A, ele é o único candidato a ser agrupado a A. Ademais, como $Limite = 0$, B é adicionado ao mesmo grupo de A. Após adicionar B, é possível adicionar C ou D ao grupo. Porém, usando a Equação 3, $Pontuacao(C) = 1$ e a $Pontuacao(D) = 0.66$. Escolhendo a maior $Pontuacao$, o vértice C é adicionado ao grupo, já que a $Pontuacao(C) = 1$ é maior que $Limite = 0.2$. Após C ser adicionado, o único vizinho disponível de C é D. Como a $Pontuacao(D) = 1.32$ é maior que $Limite = 0.4$, D é também adicionado ao grupo. O único vizinho disponível de D é E. No entanto, $Pontuacao(E) = 0.25$ é menor que $Limite = 0.6$ e, portanto, o Grupo 1 é encerrado e um novo grupo é criado (Grupo 2). Pela conexão completa dos vértices E, F, G e H, o algoritmo os adicionaria ao mesmo grupo (Grupo 2) de forma análoga à formação do Grupo 1. Dois grupos são, assim, gerados ao fim da execução do algoritmo MRAV nesse exemplo ilustrativo.

3. Experimentos e Resultados

Nesta seção, são descritas a metodologia de testes e as redes complexas reais empregadas na avaliação de desempenho. Em seguida, são reportados os resultados experimentais.

3.1. Metodologia de Testes

Testes experimentais foram projetados e realizados para analisar o desempenho do método proposto (MRAV+ML) em relação ao ML na sua forma original. Pela construção do MRAV, apenas o valor de K (ver Equação 4) pode alterar a eficiência do método em relação ao tempo de execução e à modularidade das comunidades encontradas.

¹Por conveniência, usamos “.” como separador decimal neste artigo.

Tabela 1. Redes reais utilizadas para os testes [Bai et al., 2017, Peng et al., 2014].

Nome	# Vértices	# Arestas
Email [Guimera et al., 2003]	1,133	5,451
Polblogs [Adamic e Glance, 2005]	1,490	16,718
Reactome [Joshi-Tope et al., 2005]	6,327	147,547
PGP [Boguná et al., 2004]	10,680	24,316
DBLP [Yang e Leskovec, 2015]	317,080	1,049,866
Amazon [Yang e Leskovec, 2015]	334,863	925,872
Youtube [Leskovec e Krevl, 2014]	1,134,890	2,987,624
LiveJournal [Mislove et al., 2007]	5,204,176	49,174,620

O valor de K define um limite mínimo para o percentual das conexões de um vértice fora de uma comunidade com vértices dentro dessa comunidade. Se o referido percentual excede K , o vértice se qualifica a ingressar na comunidade. Se configurado com valor muito baixo para K , o MRV relaxa a exigência de nível de conectividade para ingresso em uma comunidade. Logo, vértices são mais facilmente agrupados e o MRV gera poucos grupos mais populosos. Isso tende a reduzir o tempo de execução do conjunto (MRV+ML), mas, possivelmente, também tende a reduzir a modularidade das comunidades encontradas. Ao contrário, K com valor alto faz com que o MRV tenda a produzir muitas comunidades com poucos membros, o que não cumpre o propósito do MRV de entregar ao ML um grafo reduzido. Como consequência, MRV+ML tende a ter desempenho próximo ao do ML, tanto em termos de tempo de execução e modularidade. O melhor valor de K no MRV+ML é aquele que minimiza o tempo de execução e a perda de modularidade das comunidades resultantes, em relação ao ML. Para encontrar tal valor, serão feitos testes sobre 8 grafos comumente utilizados em trabalhos de detecção de comunidades [Bai et al., 2017, Peng et al., 2014], cujas dimensões em número de vértices e arestas são apresentadas na Tabela 1.

Para cada grafo presente na Tabela 1, o desempenho do MRV será avaliado para $K \in \{0.01, 0.02, 0.03, 0.1, 0.3, 0.4, 0.5, 0.6, 0.7, 0.8, 0.9\}$, em termos de indicadores como tempo de execução e número de grupos (comunidades) encontrado. Para aferir o efeito do sorteio aleatório do vértice inicial sobre o desempenho do MRV após a criação de um novo grupo, o MRV será executado 20 vezes para cada valor de K . Assim, valores médios e respectivos intervalos de 99% de confiança são reportados. Testes análogos aos descritos acima são feitos para avaliar o desempenho do MRV+ML em função de K no que tange a indicadores como tempo de execução e modularidade das comunidades encontradas. Com a bateria de testes proposta, espera-se ser possível definir um K para o MRV+ML que minimize tanto o tempo de execução quanto a perda de modularidade das comunidades encontradas, em relação ao ML.

Com o valor de K estudado e estabelecido, o MRV+ML será testado em relação ao ML. Para isso, o ML será também aplicado aos 8 grafos, também 20 vezes, e as médias dos resultados encontrados, para o tempo de execução e a modularidade, serão comparadas com aquelas obtidas pelo MRV+ML, com o valor de K fixado.

Todos os experimentos foram realizados em um servidor com processador Intel Xeon Dual Quad Core (2.27 GHz) com 48 GB de RAM e 1 TB para armazenamento.

3.2. Desempenho do MRV em função de K

Os resultados dos testes especificados na Seção 3.1 serão apresentados e analisados nesta seção. Para isso, o desempenho do MRV em função de K é avaliado para indicadores como o tempo de execução e o número de grupos encontrados.

Em todos os gráficos apresentados (Figuras 2 a 4), o eixo das abscissas representa o valor de K . Para cada rede, a curva sólida expressa a média dos 20 testes realizados, enquanto a região colorida que a envolve representa o intervalo de confiança para cada medida. Em alguns casos, os intervalos de confiança não são visíveis na escala mostrada.

Os resultados do tempo de execução do MRV, em função de K , indicam que o tempo de execução é razoavelmente estável em função de K para a grande maioria das redes listadas na Tabela 1.² Nesse caso, os intervalos de confiança são pequenos, sugerindo que a escolha aleatória do vértice inicial que popula um grupo tem pouco efeito sobre o tempo de execução do MRV.

A Figura 2 mostra o número de grupos gerados pelo MRV em função de K , para as redes listadas na Tabela 1. É possível perceber um aumento no número de grupos com o aumento de K . Esse efeito é o esperado, já que um valor alto de K representa uma exigência de um percentual alto de vizinhos compartilhados entre os vértices dentro do grupo e um vértice fora do grupo, para que esse vértice se qualifique a integrá-lo.

Cada grupo encontrado pelo MRV se torna um vértice de um grafo reduzido a ser entregue ao ML. Como a complexidade do ML é $\mathcal{O}(n \log_2 n)$, com n sendo o número de vértices do grafo de entrada, é desejável empregar o MRV com K pequeno. Cabe ainda notar que, nas escalas empregadas nos gráficos da Figura 2, não são visíveis os intervalos de confiança associados ao número médio de grupos. Isso sugere que a escolha aleatória do vértice inicial que popula um grupo tem efeito desprezível sobre o número médio de grupos produzidos pelo MRV, em função de K .

Para verificar experimentalmente a análise anterior, a Figura 3 apresenta o tempo total gasto pelo MRV+ML em função de K . É possível confirmar a tendência de crescimento do tempo médio de execução do MRV+ML, com o aumento de K , exceto para valores muito pequenos de K . Nota-se ainda uma tendência do intervalo de confiança decrescer com a redução de K .

Por um lado, deseja-se reduzir o tempo médio de execução do MRV+ML em relação ao ML. Por outro lado, deseja-se que sejam similares as modularidades das comunidades encontradas pelo MRV+ML e pelo ML. Para verificar isso, foi calculada a modularidade das comunidades geradas pelo MRV+ML, em função de K . O resultado é expresso na Figura 4. Percebe-se que a modularidade média, para todos os grafos testados, é aproximadamente constante para $K \geq 0.2$ e tende a decrescer para $K < 0.2$.

Dos resultados mostrados nas Figuras 3 e 4, fica claro que, na faixa $0 \leq K \leq 0.1$ (em destaque na Figura 4), reduzir K tende a diminuir o tempo médio de execução do MRV+ML, ao custo de uma queda leve na modularidade média das comunidades detectadas. Em princípio, é possível especular que esteja contido na faixa supracitada o valor de K que otimize em média o tempo de execução e a perda da modularidade. Para

²A figura associada a esses resultados foi omitida devido à limitação de espaço, uma vez que, por haver muito baixa variância, há pouco valor na representação visual desses resultados.

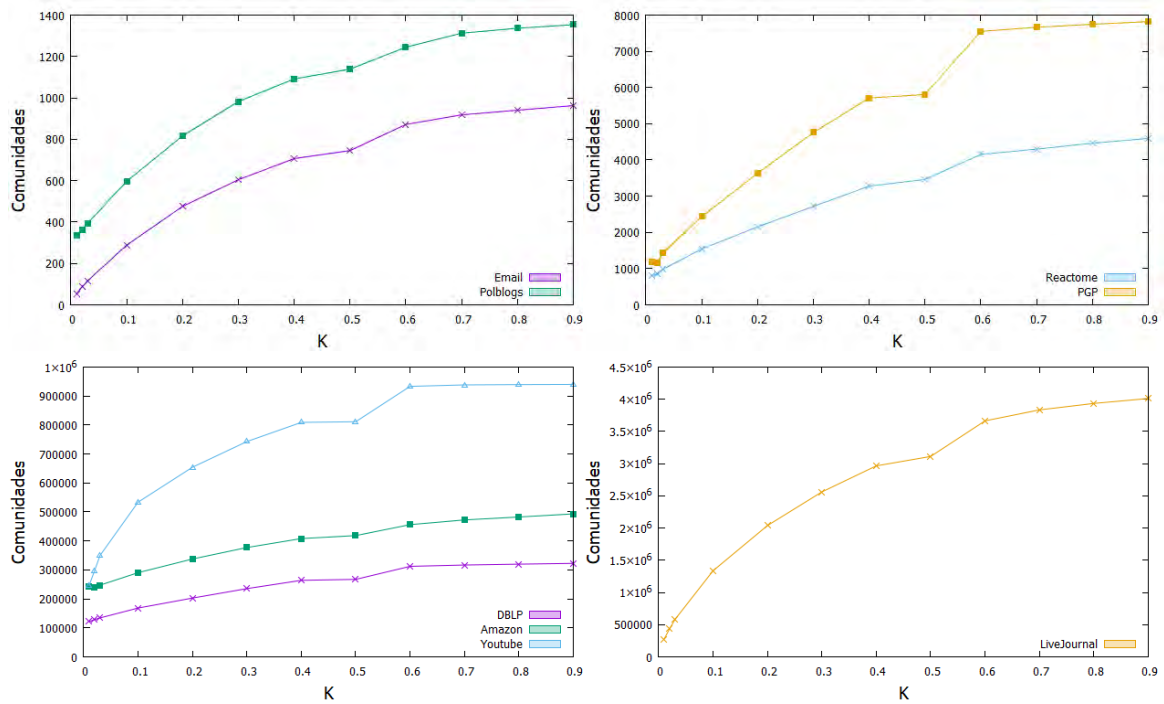


Figura 2. Número de comunidades geradas pelo MRV em função de K .

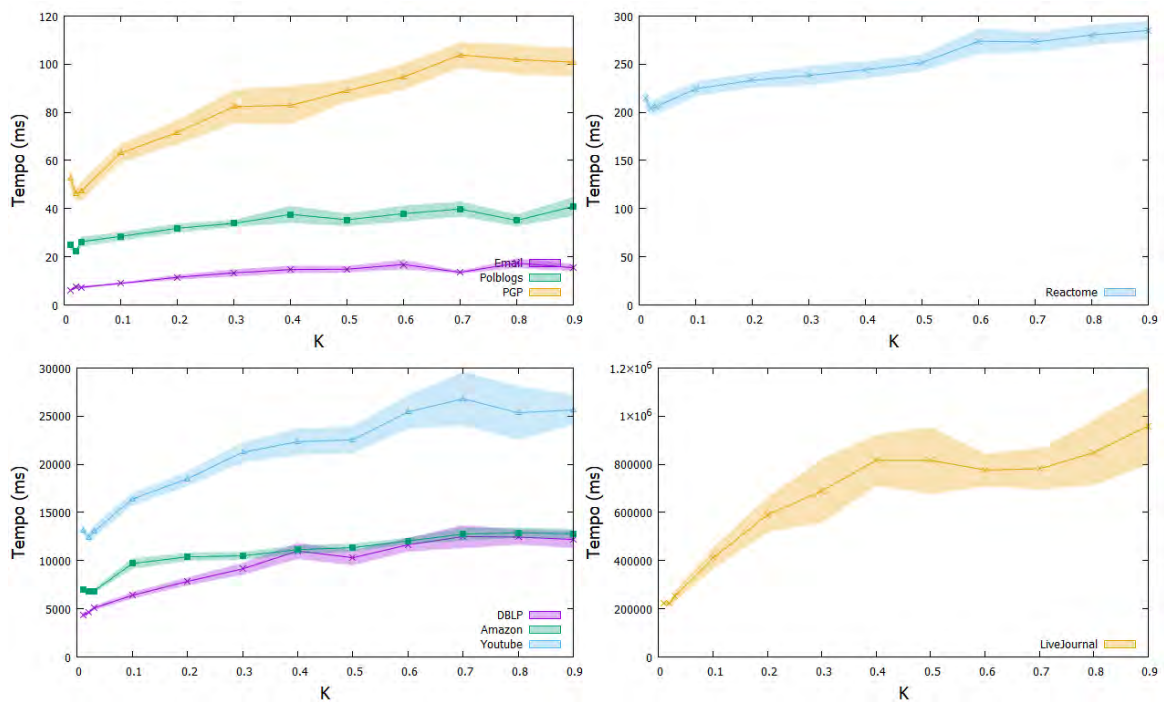


Figura 3. Tempo de execução consumido pelo MRV+ML em função de K .

explorar mais a questão da escolha de K , a próxima seção reporta resultados de testes comparativos entre o ML e o MRV+ML, com $K = 0.01$, $K = 0.03$ e $K = 0.1$, no que tange ao tempo médio de execução e a modularidade das comunidades detectadas.

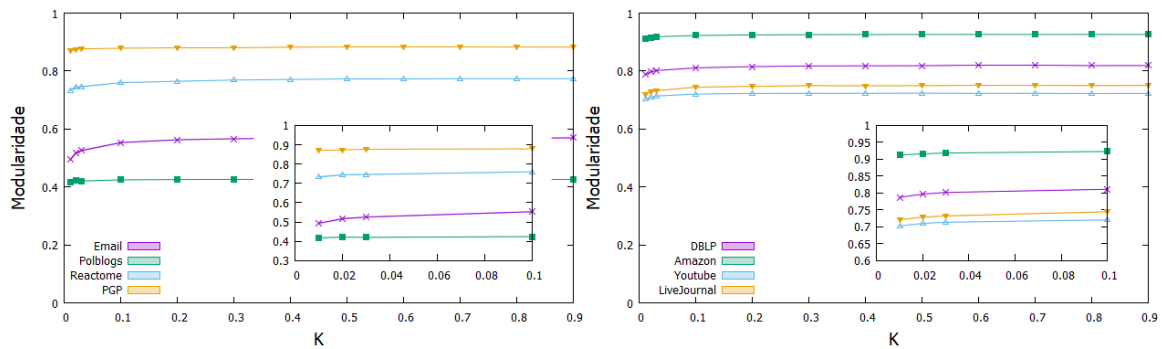


Figura 4. Modularidades das comunidades encontradas pelo MRV+ML, em função de K .

3.3. Comparação de Desempenho: MRV+ML vs ML

Nesta seção, são comparados, para $K \in \{0.01, 0.03, 0.1\}$, os desempenhos do MRV+ML contra os do ML, em termos do tempo médio de execução e da modularidade média das comunidades detectadas. Os resultados encontrados para as 8 redes da Tabela 1 são reportados na Tabela 2. Os intervalos de confiança das 20 medidas experimentais são pequenos (e.g., $< 0.1\%$) e não foram mostrados na tabela para melhor visualização dos resultados. Os valores destacados em negrito na Tabela 2 são os melhores resultados.

Tabela 2. Desempenhos comparativos entre ML e MRV+ML.

	Tempo (ms)				Modularidade			
		K				K		
		0.01	0.03	0.1		0.01	0.03	0.1
REDES	ML	MRV+ML			ML	MRV+ML		
Email	8.8	6.0	7.3	9.0	0.567	0.493	0.525	0.553
Polblogs	13.6	24.9	26.3	28.5	0.426	0.417	0.420	0.424
Reactome	65.5	214.3	206.2	224.5	0.776	0.733	0.745	0.759
PGP	70.2	52.7	47.2	63.2	0.882	0.870	0.876	0.878
DBLP	12200.3	4350.0	5083.3	6400.0	0.820	0.787	0.801	0.811
Amazon	11620.5	7000.0	6845.1	9700.0	0.926	0.912	0.918	0.922
Youtube	18278.8	13200.0	13018.1	16400.0	0.720	0.702	0.713	0.720
LiveJournal	426830	224500	253032	411350	0.748	0.720	0.732	0.744

Tabela 3. Taxas médias de redução de tempo de execução e modularidade (%).

Redes	DBLP	Amazon	Youtube	LiveJournal	Média
Tempo $K = 0.01$	64.3	39.7	27.7	47.4	44.8
Tempo $K = 0.03$	58.3	41.0	28.7	40.7	42.2
Modularidade $K = 0.01$	4.0	1.5	2.5	3.7	2.9
Modularidade $K = 0.03$	2.3	0.8	0.9	2.1	1.5

O MRV+ML produziu modularidades inferiores ao ML em todos os grafos testados. Porém, tal redução se mostrou relativamente pequena. Nota-se ainda no MRV+ML um natural decaimento de modularidade com o decaimento do valor de K .

O ML possui complexidade $\mathcal{O}(n \log n)$, enquanto o MRV possui complexidade $\mathcal{O}(e)$, sendo n o número de vértices e e o número de arestas. Logo, é esperado que o desempenho do MRV+ML seja superior para grafos esparsos. Isso é visto nos testes com redes pouco esparsas (Polblogs e Reactome) em que o MRV+ML obteve piores resultados em relação ao ML. Outros dois resultados inferiores, em relação ao tempo médio de execução, foram obtidos em outras 2 redes menores (Email e PGP). Nesses casos, por o ML já apresentar uma complexidade próxima à linear, é esperado que sua eficiência já seja próxima da ótima, não havendo espaço para melhoria adicional de desempenho.

Entretanto, dentre os 4 maiores grafos (de 0.3 a 5 milhões de vértices), observa-se na Tabela 3 que, em comparação com o ML, o MRV+ML³ é capaz de detectar comunidades com uma queda em modularidade não superior a 4%, mas com uma redução não inferior a 28% no tempo médio de execução, podendo alcançar reduções expressivas como 64%, como no caso da rede DBLP. Em termos de medidas médias de desempenho do MRV+ML, neste conjunto de redes, observa-se que aumentar K favorece a modularidade e penaliza o tempo de execução. Na média, por exemplo, para $K = 0.03$, observa-se uma redução de 42% no tempo de execução e uma perda de 1.5% na modularidade.

4. Conclusão e Trabalhos Futuros

Este artigo reportou uma investigação sobre detecção de comunidades em grafos. Foi proposto um Método Rápido de Agrupamento de Vértices (MRV) que, em conjunto com o Método de Louvain (ML), é capaz de detectar comunidades em grafos com milhões de vértices, com mais rapidez do que o ML, ao custo de uma pequena perda na modularidade das comunidades encontradas.

No MRV, a alocação de um vértice em um grupo é feita se seu percentual de conectividade com o grupo ultrapassar um limite $0 < K < 1$. O MRV usa uma heurística gulosa que prioriza a velocidade da tarefa e entrega um grafo de tamanho reduzido, onde os grupos são os novos vértices. Esse resultado pode ser entendido como similar ao produzido pelas primeiras iterações do ML, só que de modo mais rápido. Portanto, é vantajoso um esquema multi-estágio em que o MRV transforma o grafo original em um grafo reduzido sobre o qual o ML completa a detecção de comunidades.

Experimentos computacionais foram conduzidos envolvendo grafos associados a redes reais com número de vértices entre mil e cinco milhões. Para aferir a robustez do MRV à escolha aleatória do vértice inicialmente alocado a um grupo, foram executadas 20 rodadas, para cada configuração experimental. Foram reportados, portanto, a média e o intervalo de confiança das medidas de desempenho feitas sobre as 20 rodadas.

Para redes com um grande número de vértices (0.3 a 5 milhões), o MRV+ML é uma solução para a detecção de comunidades em que o parâmetro K regula as reduções médias de tempo de execução e de modularidade, em relação ao ML. Entretanto, a redução de tempo é muito mais substancial do que a perda de modularidade. Por exemplo, para $K = 0.03$, a redução média de tempo de execução é de 42%, ao custo de uma perda média de 1.5% na modularidade. Já para $K = 0.01$, a redução média de tempo de execução é de aproximadamente 45%, ao custo de uma perda média de 3% na modularidade.

³O código do MRV+ML implementado em C++ e os grafos utilizados neste trabalho estão disponíveis em <http://lps.lncc.br/index.php/demonstracoes/wperformance18>

Como possíveis desdobramentos da investigação pode-se pensar em comparar o MRV+ML com outros métodos de detecção de comunidades ou outros métodos de otimização do ML. Pode-se analisar mais detalhadamente o comportamento do MRV na faixa $0 < K < 0.1$, com o intuito de encontrar um valor ótimo de K que minimize a perda de modularidade e o tempo de execução. Outra questão a ser explorada é a possibilidade de estender o escopo de aplicação do método para detectar comunidades em grafos ponderados ou grafos direcionados.

Agradecimentos

À CAPES, FAPERJ, FAPEMIG, FAPESP e ao CNPq, pelo apoio financeiro.

Referências

- Adamic, L. A. e Glance, N. (2005). The political blogosphere and the 2004 US election: divided they blog. In *Proc. 3rd Int. Workshop on Link Discovery*, pages 36–43. ACM.
- Aldecoa, R. e Marín, I. (2011). Deciphering network community structure by surprise. *PLoS one*, 6(9):e24195.
- Aldecoa, R. e Marín, I. (2013). Surprise maximization reveals the community structure of complex networks. *Scientific Reports*, 3:1060.
- Bai, L., Cheng, X., Liang, J., e Guo, Y. (2017). Fast graph clustering with a new description model for community detection. *Information Sciences*, 388:37–47.
- Bhowmick, S. e Srinivasan, S. (2013). A template for parallelizing the louvain method for modularity maximization. In *Dynamics on and of Complex Networks, Volume 2*, pages 111–124. Springer.
- Blondel, V. D., Guillaume, J.-L., Lambiotte, R., e Lefebvre, E. (2008). Fast unfolding of communities in large networks. *Journal of Statistical Mechanics: Theory and Experiment*, 2008(10):P10008.
- Boguná, M., Pastor-Satorras, R., Díaz-Guilera, A., e Arenas, A. (2004). Models of social networks based on social distance attachment. *Physical Review E*, 70(5):056122.
- Bondy, J. A., Murty, U. S. R., et al. (1976). *Graph theory with applications*. Elsevier.
- Chakraborty, T., Dalmia, A., Mukherjee, A., e Ganguly, N. (2017). Metrics for community analysis: A survey. *ACM Computing Surveys (CSUR)*, 50(4):54.
- Fazlali, M., Moradi, E., e Malazi, H. T. (2017). Adaptive parallel Louvain community detection on a multicore platform. *Microprocessors and Microsystems*, 54:26–34.
- Fortunato, S. (2010). Community detection in graphs. *Physics reports*, 486(3-5):75–174.
- Fortunato, S. e Barthelemy, M. (2007). Resolution limit in community detection. *Proceedings of the National Academy of Sciences*, 104(1):36–41.
- Girvan, M. e Newman, M. E. (2002). Community structure in social and biological networks. *Proceedings of the National Academy of Sciences*, 99(12):7821–7826.
- Guimera, R., Danon, L., Diaz-Guilera, A., Giralt, F., e Arenas, A. (2003). Self-similar community structure in a network of human interactions. *Physical Review E*, 68(6):065103.

- Harenberg, S., Bello, G., Gjeltrema, L., Ranshous, S., Harlalka, J., Seay, R., Padmanabhan, K., e Samatova, N. (2014). Community detection in large-scale networks: a survey and empirical evaluation. *Wiley Interdisciplinary Reviews: Computational Statistics*, 6(6):426–439.
- Joshi-Tope, G., Gillespie, M., Vastrik, I., D'Eustachio, P., Schmidt, E., de Bono, B., Jasal, B., Gopinath, G., Wu, G., Matthews, L., et al. (2005). Reactome: a knowledgebase of biological pathways. *Nucleic Acids Research*, 33(suppl_1):D428–D432.
- Khan, B. S. e Niazi, M. A. (2017). Network community detection: A review and visual survey. *CoRR*, abs/1708.00977.
- Kwak, H., Lee, C., Park, H., e Moon, S. (2010). What is twitter, a social network or a news media? In *Proc. 19th Int. Conf. on World Wide Web*, pages 591–600. ACM.
- Lancichinetti, A. e Fortunato, S. (2009). Community detection algorithms: a comparative analysis. *Physical Review E*, 80(5):056117.
- Leskovec, J. e Krevl, A. (2014). SNAP Datasets: Stanford large network dataset collection. <http://snap.stanford.edu/data>.
- Leskovec, J., Lang, K. J., e Mahoney, M. (2010). Empirical comparison of algorithms for network community detection. In *Proc. 19th Int. Conf. on World Wide Web*, pages 631–640. ACM.
- Mislove, A., Marcon, M., Gummadi, K. P., Druschel, P., e Bhattacharjee, B. (2007). Measurement and analysis of online social networks. In *Proc. 7th ACM SIGCOMM Conf. on Internet Measurement*, pages 29–42. ACM.
- Newman, M. E. (2006). Modularity and community structure in networks. *Proc. National Academy of Sciences*, 103(23):8577–8582.
- Ozaki, N., Tezuka, H., e Inaba, M. (2016). A simple acceleration method for the Louvain algorithm. *Int. Journal of Computer and Electrical Engineering*, 8(3):207.
- Peng, C., Kolda, T. G., e Pinar, A. (2014). Accelerating community detection by using k-core subgraphs. *arXiv preprint arXiv:1403.2226*.
- Satuluri, V., Parthasarathy, S., e Ruan, Y. (2011). Local graph sparsification for scalable clustering. In *Proc. ACM SIGMOD Int. Conf. on Management of Data*, pages 721–732.
- Yang, J. e Leskovec, J. (2015). Defining and evaluating network communities based on ground-truth. *Knowledge and Information Systems*, 42(1):181–213.
- Zhao, Y. (2017). A survey on theoretical advances of community detection in networks. *Wiley Interdisciplinary Reviews: Computational Statistics*, 9(e1403):1–13.

Design, Implementation and Evaluation of IPv4/IPv6 Longest Prefix Match support in P4 Dataplanes

Fabricio E Rodriguez Cesen¹, P Gyanesh Kumar Patra¹,
Christian Esteve Rothenberg¹, Gergely Pongracz²

¹University of Campinas (UNICAMP), Brazil

²Ericsson Research, Hungary.

{frodri, gyanesh, chesteve}@dca.fee.unicamp.br

{Gergely.Pongracz}@ericsson.com

Abstract. *New trends in dataplane programmability inside Software Defined Networking (SDN) paradigm are in an effort to bring multi-platform support with a high-level definition of the dataplane pipeline functions. The Multi-Architecture Compiler System for Abstract Dataplanes (MACSAD) can integrate the Protocol-Independent Packet Processors (P4) domain-specific language and the OpenDataPlane Project (ODP) APIs, to define a programmable dataplane across multiple targets in a unified compiler system. In this paper, we present and evaluate the IPv4/IPv6 Longest Prefix Match (LPM) support in MACSAD. We develop a new ODP Helper library implementing the IPv6 lookup mechanism based on the current IPv4 solution and evaluate its performance and scalability for diverse workloads and target platform configurations.*

1. Introduction

Ubiquity and deep proliferation of Internet services are driving the bandwidth requirements upward exponentially, and also the necessity to carryout the actual capability evaluation of the networks. A flexible, (re-)configurable network is seen as a solution to deal with the continuously evolving network features and ever-growing bandwidth requirements by re-defining the dataplane on demand.

Software Defined Networking (SDN) [Kreutz et al. 2015] is an emerging network architecture to advocate separation of control plane from data plane. The first communications standard interface defined between the control and data plane was OpenFlow [McKeown et al. 2008]. OpenFlow adoption was marred because of its protocol dependent nature and difficulty to define new custom protocols. Programming Protocol-Independent Packet Processors (P4) [Bosshart et al. 2014] is an open source domain specific language (DSL) for expressing how packets are processed by the pipeline of a network element. P4 is evolving as a de facto language to define dataplane in SDN infrastructures. Similarly OpenDataPlane (ODP) [OpenDataPlane 2013], an open-source project, has emerged as another candidate for describing programmable dataplanes providing a common set of Application Programming Interfaces (APIs) for multiple targets.

Multi-Architecture Compiler System for Abstract Dataplanes (MACSAD) [Patra et al. 2016, Patra et al. 2017] is an approach to converge P4 and ODP through a common compilation process delivering portability of dataplane applications without

compromising target performance. During its development we observed the absence of IPv6 lookup mechanism in ODP (limited to IPv4 and no short-term plans related to this implementation by ODP team) to support more levels of addressing hierarchy, and a greater number of addressable nodes, and proposed an ODP Helper library with a lookup mechanism for IPv6 as a contribution to ODP open-source community and also extended the effort to add the support to MACSAD too. We lead a complete *performance* and *scalability* evaluation of existing IPv4 and new entrant IPv6 for diverse test workloads (packet traces, table sizes) and target platform configurations (e.g., I/O, CPU #cores).

The rest of this text contains the following topics. Background details and related works are covered in Section 2. Section 3 describes our architecture proposal for the design and implementation of IPv4/IPv6 Longest Prefix Match (LPM) support for MACSAD. Section 4 shows the performance evaluation results. Finally, conclusion and future works are discussed in Section 5.

2. Background and Related Work

This section defines three main concepts of our research work: P4 expressing how packets are processed ; ODP as the API for the networking dataplane; and finally MACSAD.

P4 Protocol Independent Switch Architecture (PISA) [Gurevich 2015, McKeown 2016] allows custom definition of network protocols in switch design approach using match+action abstractions. Top-down analysis reveals the need of a Domain Specific Language (DSL) to describe packet processing in these devices. P4 as a high level DSL provides a high enough abstractions to define dataplane, invariably supporting (re-)configuration of PISA devices. P4 is designed to be protocol independent to be able to support custom protocols and target independent to write dataplane applications agnostic to target.

ODP project is a networking dataplane API specification. It defines a set of high-level APIs spanning a common set of standard features across multi targets including ARM, Power PC, and x86 making dataplane applications portable. It enables programmers to write dataplane applications without knowing the underlying hardware. It allows programmers to leverage any specific hardware acceleration capabilities too. ODP is at a higher abstraction level than Data Plane Development Kit (DPDK)¹ and Netmap², and can use their user-space fast packet processing I/O to improve performance. Netmap works with linux kernel and allow usage of all linux based tools. DPDK fastpath is separated from linux and it restricts access of the linux based tools to configure interfaces.

ODP provides a helper library³ as an extended library supporting table management such as hash table, and IP lookup (IPv4-only). On the contrary DPDK offers various fully optimized table management libraries similar to ODP including IPv6 support. An application developed using ODP APIs needs to be linked to the *ODP implementation* block (see Fig. 1) in ODP software stack of the target platform.

MACSAD architecture overview is shown in Figure 2. It has three main modules: (i) Auxiliary Frontend, (ii) Auxiliary Backend and (iii) Core Compiler.

¹<http://dpdk.org/>

²<http://info.iet.unipi.it/~luigi/netmap/>

³<https://github.com/Linaro/odp/tree/master/helper>

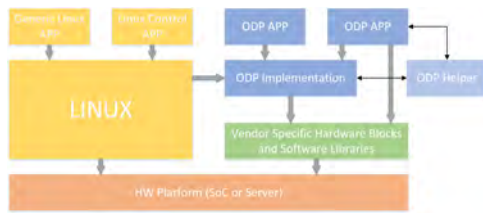


Figure 1. ODP Architecture.
Source: [OpenDataPlane 2013]

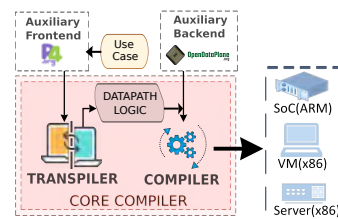


Figure 2. MACSAD Architecture.
Source: [Patra et al. 2016]

- **Auxiliary Frontend** The Auxiliary Frontend creates the Intermediate Representation (IR) for the Core Compiler, based on the P4 code as an input. The p4-hlir project is used to translates the P4 programs into a High Level IR (HLIR).
- **Auxiliary Backend** Is used to give a common Software Development Kit (SDK) for the compiler incorporating the ODP API.
- **Core Compiler** Encompasses the Transpiler and Compiler internal modules. The *Transpiler* determines the lookup mechanism, the size, and type of tables that are going to be created. The *Compiler* takes in Transpiler output and compile along with ODP APIs provided by the Auxiliary Backend to create the MACSAD Switch (MACS) (MACSAD compiled binary code is referred to as MACS) for the desired target.

While there is potentially large amount of literatures on IPv6 implementations, we briefly emphasize on related programmable dataplane activities similar to our proposed IPv4/IPv6 LPM solutions. OpenvSwitch (OVS) [OpenvSwitch 2016] is the traditional widely adopted OpenFlow based software switch with support for DPDK packet I/O for higher performance. PISCES [Shahbaz et al. 2016] is developed by extending OVS bringing in high-level DSL such as P4 support to achieve programmability. But it is limited by the restricted OVS pipeline abstractions and only supports IPv4. Similarly T4P4S [Laki et al. 2016], the closest alternative to MACSAD, is also a software switch which maps P4 abstractions to DPDK with the help of a Hardware Abstraction Layer. However, it is not multi-platform, and DPDK's LPM library for IPv6 is not supported.

3. Layer-3 forwarding (IPv4/IPv6) Implementation

This section presents LPM prototype support in MACSAD in terms of use cases, namely, Layer-3 forwarding with IPv4 (L3-IPv4) and IPv6 (L3-IPv6).

These use cases are implemented with support of ODP Helper library for LPM lookup mechanism where 32-bit and 128-bit keys are used for IPv4 and IPv6 address lookup. The P4 pipeline consists of two tables; IP lookup is performed on the first table along with corresponding actions of standard L3 packet processing (i.e., MAC re-writing, TTL/Hop Limit decrement). Then, the final packet update happens at egress section by the second table, which changes the source MAC address before sending out the packet.

- **L3-IPv4** IPv4 lookup algorithm in MACSAD uses a binary tree to perform the prefix lookup. We chose this root prefix to be 16-bit netmask. The binary tree has three levels (16-8-8) with a worst case scenario of 3 memory accesses for each IPv4 lookup.
- **L3-IPv6** Under this use case, we developed a new ODP Helper library module based on the available IPv4 solution. It is similar to DPDK⁴ solution with 15 levels of tables. The 1st level is of 16-bit followed by 14 additional levels of 8-bit each).

⁴http://dpdk.org/doc/guides-16.04/prog_guide/lpm6_lib.html

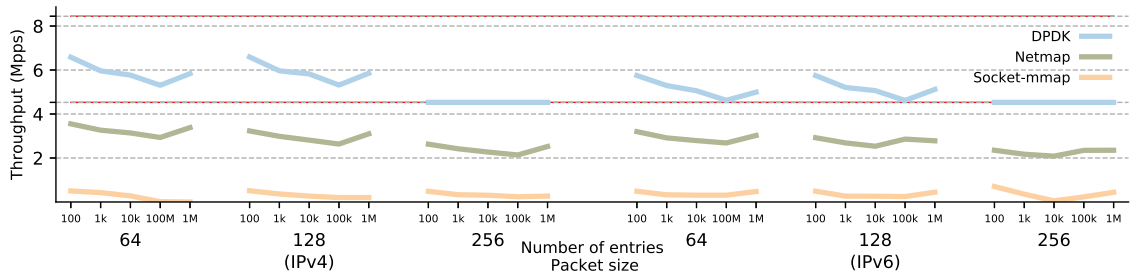


Figure 3. IPv4/IPv6 forwarding performance for different I/O drivers (1 CPU core).

4. Performance Evaluation

We evaluate performance for the two LPM use cases using three different packet I/O engines (DPDK, Netmap, Socket_mmap). For each combination, we explore the scalability for different workloads (packet traces, table entries) and configuration options (e.g., CPU cores) using Network Function Performance Analyzer (NFPA) [Csikor et al. 2015] as a benchmarking tool. To generate the traces we developed a packet crafter tool⁵ that will provide the necessary PCAP files to be used with NFPA. The pipeline implementation and other informations for reproducibility purposes including the P4 programs⁶ used by MACSAD and the traffic generator tool (BB-Gen) are available in our public repositories.

4.1. Testbed and Methodology

Our testbed contains two Lenovo ThinkServer RD640 servers with Intel Xeon E5-2620v2, 6 Cores, Hyper-Threading disabled, running at 2.1GHz, 8*8GB DDR3, a dual-port Intel X540-AT2 NIC (10G), and run with Ubuntu Linux 16.04 LTS (kernel 4.4). The Tester server runs NFPA with DPDK v17.08 and PktGen v3.4.5, and connected to the Device Under Test (DUT [Bradner and McQuaid 1999]). The DUT supports multiple packet I/Os to illustrate the ability to accommodate various different platform features, such as DPDK v17.08, ODP v1.16.0.0, Netmap v11.2, and the basic Linux Socket_mmap provided by the Linux kernel. The MACS is configured to forward packets received from one port to the other and eventually back towards NFPA, which in turn analyzes the packet throughput concerning packets per second (pps) and bits per second (bps).

For both L3-IPv4 and L3-IPv6, different number of cores (1, 2, 4, and 6) were allocated to the DUT, distinctive workloads were configured by setting different number of IP prefixes (100, 1K, 10K, 100K, 1M) in the lookup table and a matching number of L3 flows in the synthetic traces were used.

L3-IPv4. Figure 3 (Left Side) shows the performance of L3-IPv4 for different Forwarding Information Base (FIB) sizes and packet I/O drivers. The red y axes labels refer the line rate for different packet sizes. (i.e., 8.44 Mpps for 128 bytes and 4.52 Mpps for 256 bytes). The results for L3-IPv4 is grouped into three sectors indicating different packet sizes (i.e., 64, 128, 256). Each sector is further divided into five different points marking the complexity of the pipeline, i.e., the size of the FIB (100, 1K, etc.). It can be observed that MACS with DPDK reaches the line rate with packets sizes of 256 bytes regardless of the FIB table size. The performance of Netmap is comparatively lower but it reaches

⁵<https://github.com/intrig-unicamp/BB-Gen>

⁶<https://github.com/intrig-unicamp/macsad-usecases>

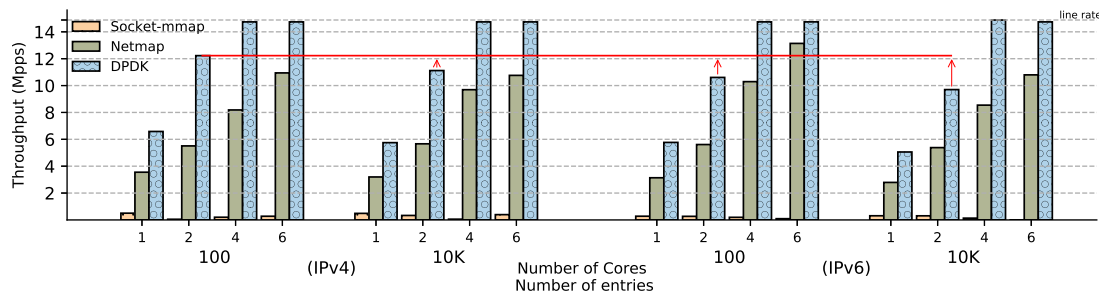


Figure 4. IPv4/IPv6 different cores performance (64 bytes packets).

line rate with 512B packets. Also it is interesting to note that, the measured results for 1M FIB entries are better than for 100K FIB entries. From the results, it is clear that the Linux Socket_mmap driver never saturates the 10G interfaces even with the largest packets (1518 bytes) due to the highly increased number of system calls, fundamental kernel scheduling, costly context switching, etc. imposed by the Linux kernel itself.

L3-IPv6. Results for the L3-IPv6 use case with 1 CPU core are shown on the right side of Fig. 3. The performance results lead to a conclusion similar to L3-IPv4 where DPDK reaches line rate with 256 byte packets for all FIB sizes. There are some performance differences in case of Netmap driver, a slight drop as the number of FIB entries grows, and what is more, when the FIB size reaches 1M the line rate is not achieved even with the biggest packet size. However, when comparing our results to L3-IPv4, we must point out that the peculiarity with 100K and 1M number of entries observed before also applies for L3-IPv6. From the IPv4 and IPv6 results, it is clear that with small packets (i.e., 64 and 128) when the FIB number of entries increases, the performance slightly reduce.

Figure 4 shows a throughput comparison as the number of cores increases from 1 to 6. When the number of CPU cores increases, MACSAD can process more packets resulting in higher throughput. It is notable (red line Fig. 4) that as the number of FIB entries increases, the throughput reduces slightly. Moreover, when table key size increase from 32 bytes (IPv4) to 128 bytes (IPv6) the performance also decreases. This is a significant finding in the understanding of how the complexity of the number of FIB entries and key sizes affect the throughput. This assumption might be addressed in future studies, analyzing the performance with different key sizes and including a variation of FIB size.

5. Conclusions and Future Work

We contributed with the addition of use cases to MACSAD, confirming ability of MACSAD to offer (re-)configurable SDN dataplanes supporting different pipelines while confirming to portability and performance as well. We demonstrated the performance of our LPM implementations by running MACS over different packet I/O drivers (DPDK, Netmap, Socket_mmap). Confirming that Socket-mmap is slower as it is the linux default driver without any fastpath advantages from Netmap or DPDK. Netmap performance is lower than DPDK on ODP system because in case of Netmap packet copy operation becomes costlier as in case of DPDK support ODP implement zero-copy feature. With this work, we accomplished some open source contributions. The developed IPv6 lookup library will be suggested for adoption by ODP project. We developed a new packet crafter tool that natively creates packets for different standard and custom protocols, and able to generate PCAP files up to more than 1M entries with different headers distributions.

Furthermore, we added various trace files to the NFPA repository too.

We will continue to improve the IPv6 library by implementing support for different 1st level sizes and varying number of subtree levels. We are also planning to analyze how the performance is being affected by the variation of the prefix length and investigate additional performance properties, e.g., packet loss, latency, CPU cycles, etc.

Acknowledgments

This work was supported by the Innovation Center, Ericsson Telecomunicações S.A., Brazil under grant agreement UNI.61.

References

- Bosshart, P., Daly, D., Gibb, G., Izzard, M., McKeown, N., Rexford, J., Schlesinger, C., Talayco, D., Vahdat, A., Varghese, G., and Walker, D. (2014). P4: Programming protocol-independent packet processors. *ACM SIGCOMM Computer Communication Review*.
- Bradner, S. and McQuaid, J. (1999). Benchmarking methodology for network interconnect devices. RFC 2544, RFC Editor. <https://www.ietf.org/rfc/rfc2544.txt>.
- Csikor, L., Szalay, M., Sonkoly, B., and Toka, L. (2015). Nfpa: Network function performance analyzer. *IEEE Conference on NFV and SDN Demo Track*.
- Gurevich, V. (2015). P4 Tutorial. <https://p4.org/assets/Nov-2015-P4-Bootcamp-Labs-Guide.pdf>.
- Kreutz, D., Ramos, F., Esteves Verissimo, P., Esteve Rothenberg, C., Azodolmolky, S., and Uhlig, S. (2015). Software-defined networking: A comprehensive survey. *Proceedings of the IEEE*.
- Laki, S., Horpácsi, D., Vörös, P., Kitlei, R., Leskó, D., and Tejfel, M. (2016). High speed packet forwarding compiled from protocol independent data plane specifications. In *ACM SIGCOMM'16 Posters and Demos*.
- McKeown, N. (2016). Programming the Forwarding Plane. <https://forum.stanford.edu/events/2016/slides/plenary/Nick.pdf>.
- McKeown, N., Anderson, T., Balakrishnan, H., Parulkar, G., Peterson, L., Rexford, J., Shenker, S., and Turner, J. (2008). Openflow: Enabling innovation in campus networks. *SIGCOMM Comput. Commun. Rev.*
- OpenDataPlane (2013). OpenDataPlane.org. <https://www.opendataplane.org>.
- OpenvSwitch (2016). OpenvSwitch. <http://openvswitch.org/>.
- Patra, P. G., Rothenberg, C. E., and Pongracz, G. (2016). Macsad: Multi-architecture compiler system for abstract dataplanes (aka partnering p4 with odp). *ACM SIGCOMM Demo and Poster Session*.
- Patra, P. G., Rothenberg, C. E., and Pongracz, G. (2017). Macsad: High performance dataplane applications on the move. *IEEE HPSR High Performance Switching and Routing*.
- Shahbaz, M., Choi, S., Pfaff, B., Kim, C., Feamster, N., McKeown, N., and Rexford, J. (2016). Pisces: A programmable, protocol-independent software switch. *ACM SIGCOMM Computer Communication Review*.

Network Address Translation using a Programmable Dataplane Processor

Juan Sebastian Mejia Vallejo¹, Daniel Lazkani Feferman¹,
Christian Esteve Rothenberg¹

¹Departamento de Engenharia de Computação e Automação Industrial (DCA)
Faculdade de Engenharia Elétrica e de Computação (FEEC)
Universidade Estadual de Campinas (Unicamp)
Caixa Postal 6101 - 13.083-970 - Campinas, SP, Brasil

{jmejia, fefer, chesteve}@dca.fee.unicamp.br

Abstract. *A short-term solution for the depletion of Internet Protocol (IP) addresses and scaling problems in network routing is the reuse of IP address by placing Network Address Translators (NAT) at the borders of stub domains. In this article, we propose an implementation of NAT using Programming Protocol-Independent Packet Processors (P4) language, taking advantage of its features such as target-agnostic dataplane programmability. Through the MACSAD framework, we generate a software switch that achieves high performance with the support of different hardware (H/W) and Software (S/W) platforms. The main contributions of this paper relate to the performance evaluation results of the NAT implementation using P4 language with MACSAD compiler.*

1. Introduction

Considering the continuous growth of Internet services such as Voice Over IP, Multimedia Over IP, on-line games and the progressive depletion of public IPv4 addresses, the Network Address Translation (NAT) [Egevang and Francis 1994] became the primary method to allow multiple Private IPs to get access to the Internet through a limited number of Public IP. Typically, an extensive range of private IP (e.g., 10.0.0.0 to 10.255.255.255), enables the communication of servers, printers, etc. However, the number of public IPs are limited and costly. In recent years, an initiative of turning rigid hardware-based networks into software-based emerged and it was named as Software-Defined Networking (SDN), splitting the control and dataplane functions to turn some network functionalities into virtualized software devices running on servers (e.g., off-the-rack x86 servers), switches or even cloud computing infrastructure [Han et al. 2015].

OpenFlow is currently maintained by the ONF (Open Networking Foundation) and is one of the first projects following the SDN methodology by letting the administrators to define dataplane functionalities. However, after more than 10 years since its foundation, the interface continues to have H/W target compatibility issues and slow support for new packet headers. In the recent development, the P4 [Bosshart et al. 2014] language introduces dataplane programmability by allowing multiple targets to use the same Domain Specific Language (DSL) to resolve the OpenFlow issues. MACSAD framework delivers cross-platform portable dataplane applications, it brings performance, flexibility and portability in dataplane. Thus, the P4 NAT S/W switch uses MACSAD and concepts of SDN and Network Function Virtualization (NFV) dataplanes [Niu et al. 2016]

to achieve a high performance programmable NAT switch. We present the NAT performance evaluation varying either target platform parameters (e.g., Packet I/O, number of cores) and packet size. The rest of this paper is structured as follows: Section II provides the background details. Section III describes the Methodology. Section IV briefly expose the Related work. Finally, Section V discusses conclusion and future works.

2. Background

2.1. Programming Protocol-Independent Packet Processors (P4)

Considering the development of OpenFlow (OF) protocol over the years, few limitations were found (e.g., most switches have multiple policies and stages of match+action tables, limited TCAM space, etc.). Furthermore, to include a new header on OF it was necessary to update its version with retro-compatibility, making the release of new versions too slow [Bosshart et al. 2014]. Initially, the first version of OF started with 12 fields. Today, the last version contains more than 40 fields and there are important headers that are not supported yet. The P4 language defines three main goals to resolve those limitations:

- **Reconfigurable in the field:** Redefine packet parsing and processing in the field should be even after it is implemented.
- **Protocol independence:** The dataplane should be configurable and not tied to a specific header format.
- **Target independence:** Packet-processing function should be independent of the target where it will be deployed. Hence the compiler could map to different forwarding devices.

P4 is a high-level language that defines how the pipeline of a network forwarding device should process the packets. The language uses an abstract forwarding model that can be defined in three main phases: first, the P4 program describes the header structure and parser to extracts the header fields. Second, the pipeline is shaped with a set of match-action tables, here the P4 program can match the header fields and execute some rules, and finally, the de-parser emits the new packet headers to send to the output port.

2.2. Multi-Architecture Compiler System for Abstract Dataplanes (MACSAD)

The MACSAD is a P4 compiler [Patra et al. 2016] that focuses on high performance with portability and flexibility. As shown in Figure 1a and 1b, the MACSAD is composed of modules compilation process :

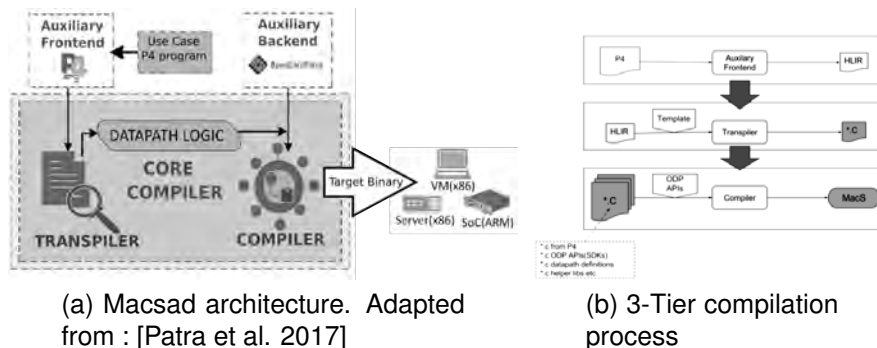


Figure 1. MACSAD architecture.

- **Auxiliary frontend:** responsible to aggregate several Domain Specific Language (DSL). It creates an Intermediate Representation (IR) from P4 code (both $P4_{14}$ and $P4_{16}$ versions); the core compiler uses this representation. In this module, the P4-hlr project is

used to translate P4 programs to a High-Level Intermediate Representation (HLIR).

- **Auxiliary backend:** aims to give a common SDK, using OpenDataPlane (ODP) APIs [OpenDataPlane 2013]. Furthermore, it contains developed libraries to bring support for P4 primitives.
- **Core compiler:** includes the transpiler and compiler modules. It merges the result of the frontend (the HLIR) and backend (the ODP APIs) to provide the binary which will be used by the device either by a Virtual Machine (VM), Raspberry (ARM), server (x86) or a SoC (ARM).

Transpiler receives the result from the Auxiliary frontend and automatically generates the Data-path Logic codes. This tool is responsible for the definition of the size, queues, lookup mechanism, and types of tables that will be created using the target's resources. The group of ".c" files generated by the transpiler contains ODP APIs, helper libraries and parts of the P4 program.

Compiler creates an executable S/W switch from the generated "C" code above mentioned, in our case a NAT and the controller interface to add the table entries. Currently, MACSAD uses Low-Level Virtual Machine and GNU Compiler Collection (GCC) compiler to guarantee the support of multiple targets.

3. Methodology

3.1. Testbed

As depicted in Figure 3b Our testbed includes two servers with Intel Xeon E5-2620v2 processors (6 cores), 64GB of memory, running Ubuntu Linux 16.04 LTS (kernel 4.4) and two dual-port Intel X540-AT2 NIC (10G). For the experiments, one node (Tester) runs a packet generator (NFPA) [Csikor et al. 2015] sending packets of different sizes to the other node with DUT via 10GB NIC. Packets are sent back through another NIC of the Tester to the first node. NFPA uses Pktgen-dpdk v3.4.5 [Turull et al. 2016] and DPDK v17.08 for packet generation. The DUT supports multiple packet I/Os with DPDK v17.08, ODP v1.16.0.0, and Netmap v11.2 versions.

3.2. NAT Dataplane P4/MACSAD Implementation

This use case has been implemented with P4₁₆, which is the latest P4 version, bringing new instructions to define tables, actions and controls. Figure 2 illustrates the implemented pipeline of our NAT P4 program¹, which is divided in Upload (UL) and Download (DL) data paths. The packet processing is supported in the MACSAD architecture (See in Fig.1a) and the dataplane is shaped with a set of tables described as follow:

- **Set interface** - it set up the network interface as external or internal to separate the UL and DL traffic;
- **L2** - The NAT acts as an L2 learning switch and processes ARP packets coming from the host; an entry is created (or updated) in the MAC address table.
- **NAT UL/DL** - Since users within a private IP network send packets to a public network, NAT is required to translate IPv4 address and TCP ports in both ways, this table store TCP port and IPv4 address entries to perform packet processing and forwarding. Otherwise, incoming packets without equivalent entries are dropped.

¹<https://github.com/intrig-unicamp/macsad-usecases/blob/master/p4-16/nat.p4>

· **IPv4 routing** - The routing table stores the next hop based on the IP address. It is based on the Longest Prefix Match (LPM) implementation.

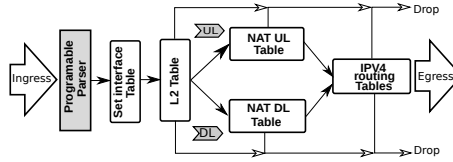


Figure 2. NAT pipeline

3.3. Performance Evaluation

Figure 3a depicts the NAT use case handling traffic between a Private and an External (Public) network, with the primary dataplane functions divided into a UL path coming from the *Host* (IP address 10.1.1.10) to an *Internet Server* (IP: 213.1.1.1), and DL path from the *Server* back to the *Host*. Figure 3b shows the testbed scenario.

UL: The UL traffic starts with *Host* that sends a packet destined to *Server*, when the packet arrives at our NAT software switch, the source IP is rewritten with the public IP and the following TCP port. Finally, the NAT performs IPv4 packet forwarding and select the output port to send the packet through the external network.

DL: As a response, the *Server* (IP: 213.1.1.1) sends TCP traffic back to the host (10.1.1.10) via NAT software switch through the external interface using NAT Public IP, which is converted to the Private IP by the NAT. Finally, it completes the IP packet forwarding by selecting the next hop and output port towards *Host*.

We have generated random traffic traces for both datapaths varying the source and destination IPv4, L2 source address, TCP source and destination ports inside the Forwarding Information Bases (FIBs). All these fields were filled through an SDN controller. We run MACSAD with three CPUs combinations (2, 4, 6 CPUs), FIB table of 100 entries, and three Packet I/O drivers (Socket-mmap, Netmap, DPDK). In the next subsection, we present the testbed setup and performance results.

3.4. Results

The bars in Figures 4a and 4b measure the performance on NAT with a packet flow of 100 entries. From the results, we observe that the best performance for all the packets sizes was with DPDK NIC driver, in fact, this PktI/O achieve line rate (i.e., 4,52 Mpps for a packet size of 256B) both UL and DL datapaths with medium and bigger packet sizes (256B-1518B). On the other hand, the worst case is clearly with 64B and 128B using Socket-mmap, as it handles more packets per second. However, this was already

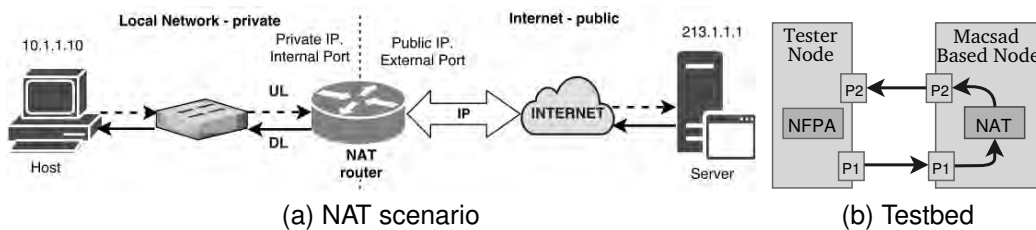


Figure 3. NAT Scenario.

expected as for low sizes it needs to match more tables, impacting the results. In Figure 4b we observe the performance increase by adding CPU cores, this is mainly using notable on Netmap and DPDK NIC drivers, considering that more cores usually means more instructions can be done, we were already expecting these results. Also, due to queues type; unlike of Socket-mmap which use hardware queues, both DPDK and Netmap, can assign virtual queues to increase performance.

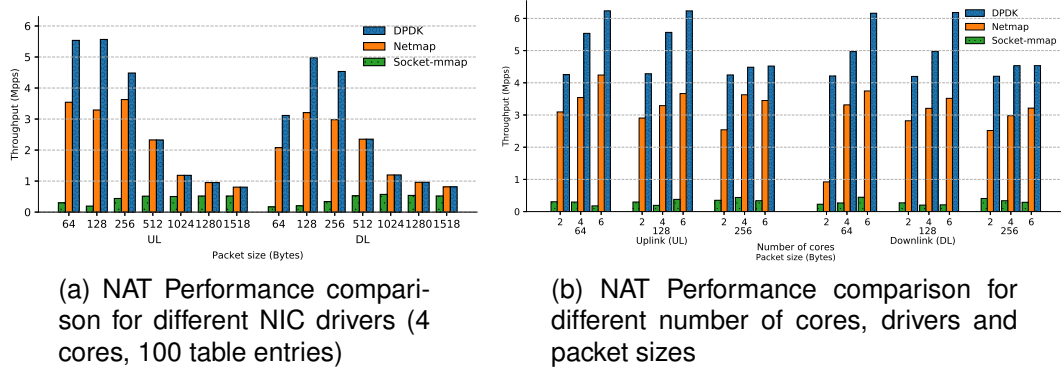


Figure 4. NAT performance results.

4. Related Work

The work in [Hwang et al. 2015] describes a S/W router, with Layer 3 forwarding functionalities that reside in distinct VMs. They use NetVM platform obtaining throughputs of up to 10 Gbps. Rather of running into H/W limitations such as NIC, their implementation is limited by the available processing capacity. In [Roberto et al. 2013] we found an approach to virtualize a Broadband Remote Access Server (BRAS) based on Click OS, a tiny Xen virtual machine designed specifically for network processing it can achieve line rate of 10Gbps with Netmap and VALE PktI/O drivers. The work in [Bondan et al. 2014] presents the management requirements in the context of a specific NFV enabler platform called ClickOS; they propose a network scenario with a NAT function. However, they studied the requirements to deploy the virtualized devices to ease the adoption of NFV by network operators. In [Palkar et al. 2015] presents E2, a framework for NFV packet processing. It is a solution to manage network functions for placement, scaling and other functionalities. In the evaluations, its data plane built on SoftNIC, a programmable software switch uses a virtualized NAT which achieves line rate of 10Gbps with 60 Bytes of packets size. Its approach differs ours as we are using P4 to specify dataplane functions with multi-target support.

5. Conclusions and Future Work

In this article, we have presented a brief description of the NAT S/W dataplane using P4 language; a new language that aims to revolutionize networks by giving programmability to the dataplane.

We have exposed a NAT S/W switch running on MACSAD over x86 platform and different PktI/O drivers (DPDK, Netmap, Socket-mmap), achieving 10G line rate with medium and large packets, confirming our approach of support complex SDN dataplanes with portability and high performance. As a future work, we intend to implement and

show the performance results in other platforms such as VMs and ARM systems, also adding new features considering the IPv6 deployment such as IPv6 to IPv4 translation mechanism. Our results encourage us to identify and resolve the bottlenecks mainly on smaller packets size and consider more performance metrics as processing delay, CPU cycles, etc.

Acknowledgments

This work was supported by the Innovation Center, Ericsson Telecomunicações S.A., Brazil under grant agreement UNI.61.

References

- Bondan, L., d. Santos, C. R. P., and Granville, L. Z. (2014). Management requirements for clickos-based network function virtualization. pages 447–450.
- Bosshart, P., Daly, D., Gibb, G., Izzard, M., McKeown, N., Rexford, J., Schlesinger, C., Talayco, D., Vahdat, A., Varghese, G., and Walker, D. (2014). P4: Programming protocol-independent packet processors. *ACM SIGCOMM Computer Communication*.
- Csikor, L., Szalay, M., Sonkoly, B., and Toka, L. (2015). Nfpa: Network function performance analyzer. *IEEE Conference on Network Function Virtualization and Software Defined Networks Demo Track*.
- Egevang, K. B. and Francis, P. (1994). The ip network address translator (nat). RFC 1631, RFC Editor. <http://www.rfc-editor.org/rfc/rfc1631.txt>.
- Han, B., Gopalakrishnan, V., Ji, L., and Lee, S. (2015). Network function virtualization: Challenges and opportunities for innovations. *IEEE Communications Magazine*, 53(2):90–97.
- Hwang, J., Ramakrishnan, K. K., and Wood, T. (2015). Netvm: High performance and flexible networking using virtualization on commodity platforms. *IEEE Transactions on Network and Service Management*, 12(1):34–47.
- Niu, Z., Xu, H., Tian, Y., Liu, L., Wang, P., and Li, Z. (2016). Benchmarking nfv software dataplanes. *CoRR*, abs/1605.05843.
- OpenDataPlane (2013). OpenDataPlane.org. <https://www.opendataplane.org>. Accessed: 2018-01-30.
- Palkar, S., Lan, C., Han, S., Jang, K., Panda, A., Ratnasamy, S., Rizzo, L., and Shenker, S. (2015). *E2: A Framework for NFV Applications*. SOSP '15. ACM, New York, USA.
- Patra, P., Rothenberg, C., and Pongracz, G. (2017). MACSAD: High performance dataplane applications on the move. *IEEE International Conference on High Performance Switching and Routing, HPSR*, 2017-June.
- Patra, P. G., Rothenberg, C. E., and Pongrácz, G. (2016). MACSAD: Multi-Architecture Compiler System for Abstract Dataplanes (Aka Partnering P4 with ODP). In *ACM SIGCOMM'16 Demo and Poster Session*.
- Roberto, B., Thomas, D., H, F., A, M., M, J., N, S., and K, H.-J. (2013). Rethinking Access Networks with High Performance Virtual Software BRASes. *EWSDN*.
- Turull, D., Sjödin, P., and Olsson, R. (2016). Pktgen: Measuring performance on high speed networks. *Computer communications*, 82:39–48.

Análise do Impacto da Elasticidade em Nuvens Computacionais Híbridas

Emanuel F. Coutinho¹, Danielo G. Gomes², Maurício M. Neto³
Leonardo O. Moreira¹, José Neuman de Souza³

¹Instituto Universidade Virtual (UFC VIRTUAL)

²Departamento de Engenharia em Teleinformática (DETI)

³Mestrado e Doutorado em Ciência da Computação (MDCC)
Universidade Federal do Ceará – Fortaleza – CE – Brasil

emanuel@virtual.ufc.br, danielo@ufc.br, maumneto@alu.ufc.br

leoomoreira@virtual.ufc.br, neuman@ufc.br

Abstract. *Hybrid computational clouds can be an attractive option in the use of computational resources with elasticity. However, adequate elasticity management is necessary because computational resources can be inefficiently provisioned, resulting in waste or idleness. This paper proposes an analysis of the elasticity effects on a hybrid computational cloud. As a main result, it was identified that elasticity maintains the quality level defined for the attendance of applications executed in a hybrid cloud in face of varied workloads.*

Resumo. *Nuvens computacionais híbridas podem ser uma atrativa opção na utilização de recursos computacionais com elasticidade. Porém é necessária uma gestão adequada da elasticidade, pois recursos computacionais podem ser provisionados de maneira ineficiente, resultando em desperdício ou ociosidade. Este artigo propõe uma análise dos efeitos da elasticidade em uma nuvem computacional híbrida. Como principal resultado, identificou-se que a elasticidade mantém o nível de qualidade definido para o atendimento das aplicações executadas em uma nuvem híbrida diante de cargas de trabalho variadas.*

1. Introdução

O aumento do acesso aos ambientes de nuvens computacionais, devido à facilidade de utilização e pagamento por uso, provoca um acréscimo na quantidade de usuários e suas respectivas cargas de trabalho. Assim, provedores de serviços devem gerenciar melhor seus recursos para garantir o nível de qualidade acordado (*Service Level Agreement* - SLA). Caso contrário, ocorrendo violações no SLA, multas podem ser aplicadas. Devido a esse crescimento na utilização dos recursos computacionais, uma das características principais da computação em nuvem tem se tornado bastante atrativa: a elasticidade.

[Herbst et al. 2013] definiram elasticidade como o quanto um sistema é capaz de se adaptar a variações na carga de trabalho pelo provisionamento e desprovisionamento de recursos de maneira autônoma, de modo que em cada instante no tempo os recursos computacionais disponíveis combinem com a demanda da carga de trabalho o mais próximo possível. Atualmente diversos provedores (e.g. Amazon EC2, Microsoft Azure e HP Cloud Services) disponibilizam serviços de elasticidade aos usuários.

Em geral, para avaliar o desempenho de sistemas computacionais são utilizadas métricas de alguma característica do ambiente, como CPU, memória e rede, pois muitas vezes a elasticidade está associada a algum recurso do provedor. Além disso, existe uma grande quantidade de tecnologias e estratégias para o provimento da elasticidade, aplicadas aos diferentes tipos de nuvens computacionais.

Nuvens híbridas surgem como uma atraente opção em termos de custo/benefício. Entretanto, sem uma gestão adequada da elasticidade, recursos computacionais podem ser provisionados de maneira ineficiente, resultando em desperdício de recursos ou ociosidade. A nuvem híbrida possibilita aos usuários que, possuam acesso a sua própria nuvem privada, possam redimensionar recursos de computação para as nuvens públicas ocasionalmente [Imai et al. 2013]. Nesse contexto, podemos analisar a seguinte hipótese: a elasticidade mantém o nível de qualidade definido para o atendimento das aplicações executadas em uma nuvem computacional híbrida.

O SLA definido para o atendimento das aplicações executadas em uma nuvem híbrida é um fator importante, pois diferentes ambientes computacionais são mais complexos de se integrarem entre si, e de manter a qualidade no atendimento aos serviços dos usuários. Além disso, diferentes provedores possuem diferentes tecnologias e políticas as quais os usuários se submetem. Espera-se que haja uma associação com a elasticidade, pois esta capacidade se propõe a ajustar os recursos do ambiente de maneira que estes se adequem às necessidades das cargas de trabalho impostas. Também se espera uma manutenção do SLA devido à adequação dos recursos provida pela elasticidade. Nesse contexto, o objetivo deste trabalho é avaliar os efeitos da elasticidade sobre uma nuvem computacional híbrida diante de cargas de trabalho aplicadas ao ambiente.

2. Material e Métodos

Os experimentos têm como objetivo avaliar a elasticidade de uma nuvem híbrida, onde inicialmente são utilizados recursos de uma nuvem privada, e conforme a necessidade por mais recursos, estes são adicionados a partir de uma nuvem pública. Para apoiar a análise de desempenho, utilizou-se um *framework* conceitual específico para elasticidade em nuvem, apresentado em [Coutinho 2014]. Ele é composto por três macroatividades, relacionadas ao planejamento, inicialização de serviços e análise.

Os experimentos utilizaram dois ambientes diferentes de nuvens computacionais: uma nuvem privada e uma nuvem pública. Para a nuvem privada, o OpenNebula 3.8, com cada máquina virtual foi criada com 1 VCPU, 1 GB de memória RAM e sistema operacional Linux Ubuntu Server 12.04 64 bits. Para a nuvem pública, foi utilizada a plataforma da Microsoft Azure, com instâncias criadas do tipo A1 padrão (1 núcleo e 1.75 GB de memória RAM) e sistema operacional Linux Ubuntu Server 14.04 64 bits. Utilizou-se como servidor *web* o Apache Tomcat, balanceador de carga o NGINX, e gerador de cargas de trabalho HTTPERF. Para a instanciação das atividades do *framework* utilizou-se Java e *shell script*. A Figura 1 exibe o *testbed* utilizado, baseado na arquitetura proposta por [Coutinho et al. 2016b].

Arquivos texto registram o *log* das operações de coleta das máquinas virtuais e das demais informações consolidadas. O *log* gerado para cada máquina virtual em arquivo texto contém a data da coleta, valores de utilização de CPU, memória, disco, rede e tempo de resposta das requisições, assim como a média de utilização de CPU, alocação

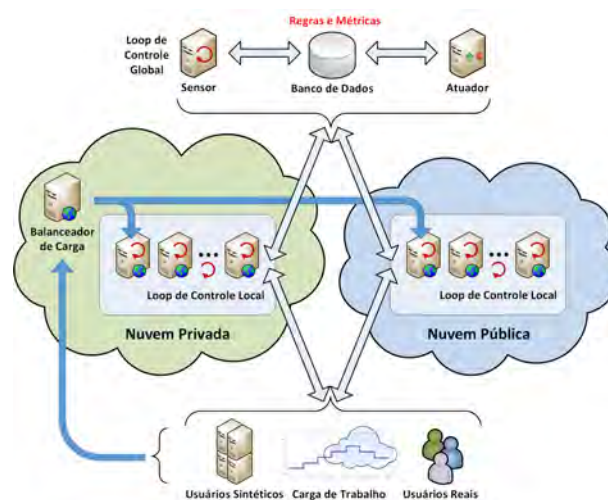


Figura 1. Arquitetura e ambiente experimental [Coutinho et al. 2016b]

de recursos e dados sobre a elasticidade. Os dados coletados e consolidados são apresentados sob a forma de gráficos de linha. O intervalo de coletas definido foi de 1 segundo adicionado do custo da coleta e análise dos resultados.

A geração de cargas de trabalho para os experimentos ocorreu de duas maneiras distintas: (i) requisições encaminhadas diretamente na máquina virtual do balanceador de carga, geradas pelo HTTPERF e por navegadores *web*, distribuídas entre as demais máquinas virtuais alocadas; e (ii) requisições executadas diretamente nas máquinas virtuais da infraestrutura. Dessa maneira é possível emular a concorrência pelos recursos em um ambiente de computação em nuvem.

Para a elasticidade, um mecanismo baseado na arquitetura proposta em [Coutinho et al. 2016b] foi implementado. Uma estratégia de elasticidade horizontal foi utilizada, onde sempre que recursos são necessários, novas máquinas virtuais são adicionadas ou removidas do balanceador de carga conforme a necessidade. Para disparar ações de elasticidade, foi utilizada a média do percentual de utilização de CPU das máquinas virtuais.

Algumas métricas específicas para a elasticidade foram utilizadas neste trabalho. As métricas propostas por [Coutinho et al. 2016a] são baseadas nos conceitos de elasticidade da Física e da Microeconomia, e avaliam a alocação e ajuste de recursos de uma nuvem computacional, indicando a necessidade por mais recursos ou não. Tais métricas possibilitam uma avaliação da elasticidade por meio de uma análise numérica e gráficos. Adicionalmente, as métricas de Elasticidade de *Scaling Up* e Elasticidade de *Scaling Down* [Herbst et al. 2013] foram utilizadas para comparação.

Os limiares utilizados para as ações de elasticidade foram: acima de 70% (aloca uma nova máquina virtual), abaixo de 60% (desaloca uma máquina virtual), e entre 60% e 70% (mantém alocação). Esse valor foi calculado como a média das 10 últimas coletas de utilização de CPU nas máquinas virtuais. Como mecanismo de predição foi utilizado regressão multilinear sobre valores de utilização de CPU, memória, disco e rede. Para o provisionamento dos recursos, a estratégia de balanceamento de carga foi utilizada, onde máquinas virtuais são adicionadas conforme a necessidade.

3. Resultados

Para o experimento, apenas uma máquina virtual na nuvem privada e na nuvem pública foram utilizadas. A Figura 2 exibe a média de utilização de CPU consolidada em todas as máquinas virtuais, a alocação das máquinas virtuais, as métricas de elasticidade baseadas em conceitos da Física e da Microeconomia [Coutinho et al. 2016a] e o tempo de resposta das requisições. Este experimento teve duração de 36min10s. Dessa maneira foi possível verificar se a infraestrutura proveria recursos a partir das duas nuvens, constituindo uma nuvem híbrida, conforme a necessidade gerada pela carga de trabalho aplicada.

A média do percentual de utilização de CPU foi bastante diversificada, chegando em alguns pontos próximo a 100% de utilização. Isto ocorreu devido a muitas requisições e poucas máquinas virtuais para o atendimento. Entretanto, sempre que o limiar de 70% estabelecido como limite superior era ultrapassado, a máquina virtual da nuvem pública era adicionada, reduzindo o valor do percentual médio de utilização de CPU. Violações ocorreram por causa da utilização de apenas duas máquinas virtuais no experimento (poucos recursos), e assim que a utilização de CPU superava o limite e ambas máquinas virtuais estavam alocadas, não existia mais a possibilidade de se alocar mais recursos. Momentos de alocação das duas máquinas virtuais em geral coincidem com momentos de alta utilização de CPU.

As métricas para elasticidade possuíram valores baixos até cerca da metade do

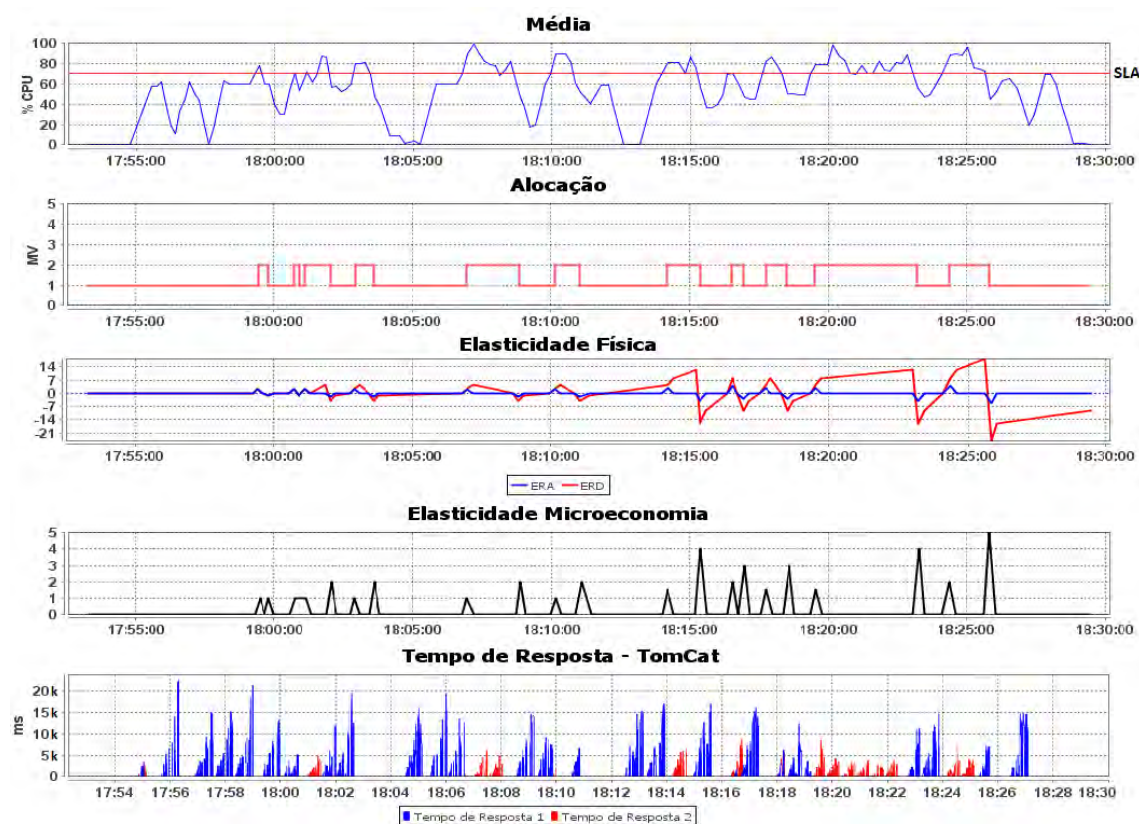


Figura 2. Média de utilização de CPU, alocação das máquinas virtuais, métricas de elasticidade baseadas em conceitos da Física e da Microeconomia e tempo de resposta das requisições.

experimento. A partir da metade do tempo do experimento os valores começaram a aumentar, e a utilização média de CPU em geral permaneceu elevada, indicando uma necessidade maior de recursos. Na maioria dos casos, momentos de alocação e desalocação coincidiram com os picos nas métricas de elasticidade. O que provocou essa variação foi o SLA definido e a velocidade na qual recursos são alocados e desalocados.

O tempo de resposta das requisições possuiu os maiores valores no início do experimento, e à medida em que o experimento avançava, estes tempos foram reduzindo. Como só haviam duas máquinas virtuais disponíveis para o experimento, assim que as requisições eram distribuídas pelo balanceador de carga, o tempo de resposta diminuía. Porém isso não implicava no pleno atendimento do SLA em todo o período do experimento, visivelmente identificado no gráfico de utilização de CPU.

3.1. Discussão dos Resultados

A Tabela 1 exibe métricas de elasticidade dos experimentos e da literatura. A Elasticidade de *Scaling Up* e Elasticidade de *Scaling Up* indicaram a velocidade de alocação e desalocação de recursos. Isso se deve ao fato de apenas existir duas máquinas virtuais envolvidas no experimento, e rapidamente se alocava e desalocava recursos.

A duração das requisições em vários pontos dos experimentos resultou em altos valores. Em geral, uma requisição duraria 2ms, e algumas duraram cerca de 20.000ms, sendo muito tempo para uma requisição. O tempo de resposta não foi considerado nas regras para ações de elasticidade, os três experimentos foram prejudicados no atendimento das requisições. O ideal é que métricas de aplicação, como o tempo de resposta e vazão, sejam incluídas nas regras para a execução de ações de elasticidade.

A utilização de máquinas virtuais em nuvens públicas não teve um alto impacto nos resultados dos experimentos do ponto de vista de infraestrutura. Entretanto, utilizar o percentual médio de utilização de CPU como métrica para acionar as ações de elasticidade poderia ser mais efetiva, pois facilmente se atingiria o limite definido, gerando violações no SLA e altos tempos de resposta das requisições. A latência da rede pode ter sido um fator que influenciou no elevado tempo de resposta, podendo também ter sido utilizada como uma regra para elasticidade.

A Figura 2 exibiu ED_i picos com uma diferença visual em relação à largura dos picos, sendo relativamente estreitos, indicando uma não estabilidade crescente nas alocações dos experimentos. Para ER_{Ai} e ER_{Di} , a partir da metade do tempo dos experimentos, ocorreu uma maior variação nos valores, indicando uma alocação mais intensa. Entretanto, a alta variação entre extremos indicou uma necessidade de recursos maior do que o ambiente possui.

Foi observado pelos experimentos realizados que na maior parte do tempo a elasticidade manteve o SLA definido para o atendimento das requisições na nuvem híbrida.

Tabela 1. Métricas de elasticidade

Métricas	Experimento
Elasticidade Média da Física (ER_{Ai})	0.00
Elasticidade Média da Física (ER_{Di})	-0.08
Elasticidade Média da Microeconomia (ED_i)	0.70
Elasticidade de <i>Scaling Up</i>	0.04
Elasticidade de <i>Scaling Down</i>	0.09

Entretanto, estabilidade não necessariamente implica em atendimento ao SLA. O experimento possuiu maior estabilidade nos estados de alocação, porém necessitou de mais recursos e mais tempo passou alocando e desalocando recursos, obtendo a maior quantidade de violações. Em relação à quantidade de recursos alocados, o experimento, mesmo tendo apenas 2 máquinas virtuais envolvidas, realizou várias alocações e desalocações.

Para a melhoria do ambiente, sugere-se a utilização de novas regras para a execução de ações de elasticidade, sendo a mais indicada o tempo de resposta das requisições e a latência da rede. É possível utilizar estratégias de alocação e predição mais efetivas, para evitar momentos longos de subprovisionamento e sobreprovisionamento de recursos, evitando violações no SLA, ociosidade e desperdício.

4. Conclusão

O mecanismo de elasticidade aplicado manteve o nível de qualidade definido para o atendimento das requisições na nuvem híbrida, confirmando a hipótese proposta. As principais conclusões deste trabalho são: (i) as métricas de elasticidade aplicadas possuem alta relação com o projeto de carga de trabalho; (ii) a elasticidade se comportou bem em relação à utilização de CPU, mas não foi eficiente em relação ao tempo de resposta das requisições; (iii) métricas de aplicação, como tempo de resposta, devem ser utilizadas em ações de elasticidade e em regras para melhorar o desempenho do ambiente.

Apenas métricas de infraestrutura foram utilizadas como critério de qualidade e ações de elasticidade, o que prejudicou a avaliação do ambiente com métricas de aplicação como o tempo de resposta. O ideal era ter mais experimentos para comparar tais valores, e ter uma análise mais completa, com subsídio das métricas calculadas. Como trabalhos futuros pretende-se investigar os efeitos da elasticidade em nuvens híbridas com diferentes mecanismos de elasticidade, como elasticidade vertical, replicação e estratégias combinadas, utilizando métricas de aplicação.

Referências

- Coutinho, E. F. (2014). *FOLE: Um Framework Conceitual para Avaliação de Desempenho da Elasticidade em Ambientes de Computação em Nuvem*. Doutorado, Mestrado e Doutorado em Ciência da Computação (MDCC), Universidade Federal do Ceará (UFC), Fortaleza.
- Coutinho, E. F., Rego, P. A., Gomes, D. G., and de Souza, J. N. (2016a). Physics and microeconomics-based metrics for evaluating cloud computing elasticity. *J. Netw. Comput. Appl.*, 63(C):159–172.
- Coutinho, E. F., Rego, P. A. L., Gomes, D. G., and de Souza, J. N. (2016b). An architecture for providing elasticity based on autonomic computing concepts. In *Proceedings of the 31st Annual ACM Symposium on Applied Computing, SAC '16*.
- Herbst, N. R., Kounev, S., and Reussner, R. (2013). Elasticity in cloud computing: What it is, and what it is not. In *Proceedings of the 10th International Conference on Autonomic Computing (ICAC 2013), San Jose, CA*, pages 23–27. USENIX.
- Imai, S., Chestna, T., and Varela, C. (2013). Accurate resource prediction for hybrid iaas clouds using workload-tailored elastic compute units. In *Utility and Cloud Computing (UCC), 2013 IEEE/ACM 6th International Conference on*, pages 171–178.

Simulação escalável de epidemias em redes baseadas em passeios aleatórios

João Vitor B. Tavares¹, Giulio Iacobelli¹, Daniel R. Figueiredo¹

¹Programa de Engenharia de Sistemas e Computação (PESC) / COPPE
Universidade Federal do Rio de Janeiro (UFRJ) - Rio de Janeiro, RJ

{jtavares,giulio,daniel}@land.ufrj.br

Abstract. *Understanding how networks and dynamic processes relate is a important topic of research, and particularly in the context of epidemics spreading through networks. In this work, we consider the scenario where individuals move through a network. Contagion occurs when two (or more) individuals are in the same location (vertex). The aim of this work is to build a scalable simulator to this type of epidemic that can be used to analyze the behavior of epidemics in varios scenarios. Analytical and empirical results indicate that the simulator is scalable in the network size, simulation time and number of individuals.*

Resumo. *Entender como redes e processos dinâmicos se relacionam é um tema central de pesquisa, e em particular no contexto de epidemias que se desdobram sobre redes. Neste trabalho, consideramos o cenário onde indivíduos se movimentam por uma rede, que representa a estrutura do espaço de movimentação. O contágio pode ocorrer quando dois (ou mais) indivíduos se encontram em um mesmo local (vértice). O objetivo deste trabalho é projetar e implementar um simulador eficiente que possa ser utilizado para caracterizar o comportamento deste tipo epidemia em diferente cenários. Uma avaliação teórica e empírica indica que o simulador é escalável no tamanho da rede, tempo de simulação e número de indivíduos.*

1. Introdução

Ao longo de toda sua existência, a humanidade convive com epidemias de diversos tipos, e mais recentemente, com cascatas de informação em mídias sociais que podem ser representadas por epidemias.

Modelos epidêmicos em geral atribuem estados epidêmicos aos indivíduos, sendo os mais comuns: *Suscetível* (indivíduos que não estão contagiados mas que ainda podem ser contagiados), *Contraído* (indivíduos que foram contagiados mas ainda não são capazes de transmitir a epidemia), e *Infectado* (indivíduos contagiados que são capazes de propagar a epidemia).

Modelos mais recentes de epidemia em redes mais consideram a rede como sendo uma estrutura de locais sobre a qual indivíduos se movimentam [Draief and Ganesh 2011, Zhou and Liu 2009]. Neste cenário, cada vértice da rede representa um local e cada aresta indica a possibilidade de locomoção direta entre dois locais. Indivíduos possuem um estado epidêmico e o contágio pode ocorrer quando dois (ou mais) indivíduos se encontram em um mesmo local. Este trabalho estuda este tipo de modelo de epidemia em redes.

Neste trabalho projetamos e implementamos um simulador eficiente de eventos discretos para simular epidemias SCIS em redes com indivíduos se movimentando. Em particular, uma análise teórica e experimental mostra que, para um tempo fixo de simulação T , o tempo de execução não depende do tamanho da rede, que o tempo de execução cresce linearmente com tempo de simulação T , e que o tempo de execução cresce como $\mathcal{O}(K \log K)$ com o número K de indivíduos na rede.

2. Trabalhos Relacionados

Existe um grande número de modelos epidêmicos na literatura, incluindo muitos baseados em redes [Zhou and Liu 2009, Draief and Ganesh 2011, Ganesh et al. 2005, Pastor-Satorras and Vespignani 2001], além de alguns simuladores de epidemia em redes.

Barret et al. propõem um simulador escalável e paralelo de epidemias em redes, *EpiSimdemics* [Barrett et al. 2008]. Uma rede de contatos representada por um grafo bipartido que varia no tempo é avaliado, onde uma classe de vértices representa as pessoas e a outra representa as localidades e uma aresta é adicionada entre os vértices caso uma pessoa decida visitar uma localidade. Uma diferença com o presente artigo é que a movimentação dos indivíduos está restrita a rede, podendo estes se movimentarem apenas para vértices vizinhos ao vértice em que se encontram.

Alguns outros trabalhos analisam a propagação de epidemia utilizando indivíduos que se movimentam em uma rede [Zhou and Liu 2009, Draief and Ganesh 2011]. Entretanto estes reportam resultados mais teóricos e assintóticos (quando o número de vértices da rede vai ao infinito), enquanto estamos interessados em resultados para redes de tamanho finito, com um número finito de indivíduos.

3. Modelo Epidêmico

Iremos considerar um modelo de epidemia em redes diferente da abordagem tradicional, onde indivíduos possuem mobilidade. Para isso, definimos $G(V, E)$, onde V é o conjunto de vértices que representam as localidades por onde os indivíduos se movimentam e E é o conjunto de arestas representando a possibilidade de mobilidade direta entre as localidades. Definimos também K como o número de indivíduos que se movimentam pela rede de acordo com passeios aleatórios independentes. O tempo que um indivíduo permanece em um vértice antes de se locomover para um vizinho é dado por uma variável aleatória exponencial com parâmetro λ . Desta forma, ao entrar em um vértice, o indivíduo permanece um tempo aleatório até se movimentar para um vértice vizinho.

A todo instante, cada indivíduo se encontra em um único estado epidêmico, podendo ser Suscetível, Contraído ou Infectado. Além disso, iremos adotar o modelo SCIS que define as transições entre os estados epidêmicos. A medida que os indivíduos se movimentam pela rede, ocorrem encontros entre indivíduos infectados e suscetíveis. Em cada encontro deste tipo o contágio ocorre de acordo com uma variável aleatória de Bernoulli de parâmetro p_u , onde u é um vértice da rede. Ou seja, com probabilidade p_u o indivíduo suscetível é contagiado pelo indivíduo infectado no instante do encontro. A associação da probabilidade de transmissão p_u ao vértice permite modelar localidades da rede com diferentes taxas de infecção.

Os indivíduos permanecem no estado suscetível até que sejam infectados e quando isso ocorre, eles passam imediatamente para o estado contraído. O tempo de permanência

no estado contraído possui distribuição exponencial com parâmetro τ . Ao final deste tempo o indivíduo muda para o estado Infectado. Uma vez infectado, o indivíduo permanece neste estado por um tempo aleatório com distribuição exponencial com parâmetro γ .

4. Simulador

Para estudar o modelo de epidemia em redes proposto, projetamos e implementamos um simulador baseado em eventos discretos. Os eventos do simulador são *Caminhar* (C), *Infectar* (I) e *Recuperar* (R), sendo cada evento aplicado a um indivíduo em um determinado instante de tempo. Ou seja, cada evento é representado por um tipo de evento $\{C, I, R\}$, o indivíduo a quem pertence o evento $i \in \{1, \dots, K\}$ e o seu tempo de ocorrência t no relógio de simulação. Os eventos da lista estão ordenados crescentemente de acordo com t .

Algorithm 1 Laço principal do simulador

```

while  $S_t \neq K$  e tempo de simulação  $< T$  do
   $e \leftarrow$  remover da LE
  if  $e.tipo == \text{Caminhar}$  then
    Gerar Caminhar para  $e.passeio$ 
    Inserir Caminhar na LE
     $v \leftarrow \text{mover}(e.passeio)$ 
    if  $e.passeio.estado == infectado$  then
      infectar( $v$ )
    else if  $e.passeio.estado == suscetivel$  then
      ser-infectado( $v$ )
    end if
  else if  $e.tipo == \text{Infectar}$  then
    Gerar Recuperar
    Inserir Recuperar na LE
    infectar( $e.passeio.vertice$ )
  else if  $e.tipo == \text{Recuperar}$  then
    ser-infectado( $e.passeio.vertice$ )
  end if
end while

```

Ao processar o evento caminhar $\{C, i, t\}$, caso o indivíduo i esteja no estado suscetível e seja contagiado ao entrar no vértice de destino, é gerado um evento *Infectar* para o indivíduo (repare que a transição para o estado contraído é imediata, ocorrendo no mesmo instante do encontro). Por outro lado, se o indivíduo que se movimenta estiver no estado infectado, é gerado um evento *Infectar* para todo indivíduo que ele contagiar no vértice destino. Ao final, é gerado um novo evento *Caminhar* cujo tempo de ocorrência é $t_1 = t + \Delta_1$, sendo Δ_1 uma variável aleatória exponencial com parâmetro λ , definida na seção 3.

O processamento do evento $\{I, i, t\}$ muda o estado do indivíduo de contraído para infectado. Ao tornar-se infectado, o indivíduo pode infectar indivíduos suscetíveis (caso haja) que estão no mesmo vértice que ele, gerando um novo evento *Infectar* para cada

indivíduo contagiado. O processamento do evento *Infectar* gera um evento *Recuperar* para o mesmo indivíduo, ou seja, é gerado um evento $\{R, i, t_1\}$, cujo tempo de ocorrência é $t_1 = t + \Delta_2$, sendo Δ_2 uma variável aleatória exponencial com parâmetro γ .

Ao processar o evento $\{R, i, t\}$, o indivíduo i muda do estado infectado para o estado suscetível. Quando isso ocorre, o indivíduo pode ser novamente contagiado por indivíduos infectados que estejam no mesmo vértice que ele. Deste modo, no mesmo instante de tempo t , o indivíduo passa do estado infectado para o suscetível e do suscetível para o contraído, e um novo evento *Infectar* é gerado. Todo evento *Infectar* é gerado no instante em que o indivíduo muda para o estado epidêmico contraído. Em particular, o evento $\{I, i, t_1\}$ é gerado no instante t no qual o indivíduo i é contagiado, onde $t_1 = t + \Delta_3$, sendo Δ_3 uma variável aleatória exponencial com parâmetro τ .

O algoritmo 1 mostra o laço principal do simulador que processa o próximo evento da lista de eventos, gerando os eventos necessários. Repare que a simulação termina quando não há mais indivíduos infectados ou o tempo de simulação atinge o limite de T . As funções *ser-infectado()* e *infectar()* são fundamentais para a complexidade do simulador e serão discutidas em seguida.

O tamanho da lista de eventos do simulador é proporcional ao número de indivíduos K , havendo sempre um evento *Caminhar* para cada indivíduo. Os eventos *Infectar* e *Recuperar* são mutuamente exclusivos, pois caso o indivíduo esteja no estado contraído, existe um evento *Infectar* na lista de eventos e caso ele esteja no estado infectado, existe um evento *Recuperar* na lista de eventos. Além disso, no estado suscetível, o indivíduo não possui nenhum dos dois eventos. Portanto, cada indivíduo pode ter no mínimo um e no máximo dois eventos na lista de eventos. Assim, o tamanho da lista de eventos respeita a seguinte desigualdade: $K \leq |LE| \leq 2K$.

5. Desempenho computacional do simulador

A complexidade computacional do simulador depende tanto do algoritmo 1 quanto dos algoritmos que processam cada tipo de evento. Um aspecto central são as estruturas de dados utilizadas para armazenar a lista de eventos, a rede e os indivíduos. Os eventos são armazenados em uma fila de prioridade mínima, onde a complexidade das operações de remover e inserir tem custo $\mathcal{O}(\log K)$. O evento caminhar precisa escolher um vértice vizinho na rede, mas isto tem complexidade constante $\mathcal{O}(1)$, pois o grafo é armazenado é um vetor de adjacências. Desta forma a complexidade do laço principal do simulador não depende da estrutura e do tamanho da rede.

Na função *ser-infectado*, é utilizado a equação $1 - (1 - p_u)^{k_u}$ (k_u é o número de indivíduos no vértice u), ao custo $\mathcal{O}(1)$, para determinar se o indivíduo será contagiado por outros indivíduos infectados no vértice destino. Já na função *infectar*, é preciso iterar sobre todos os indivíduos suscetíveis no vértice destino para determinar individualmente se cada um será (ou não) contagiado, ao custo $\mathcal{O}(k_u)$. Sempre que houver um contágio, um novo evento *Infectar* é gerado para o indivíduo recém infectado e inserido na lista de eventos ao custo $\mathcal{O}(\log K)$. Como os indivíduos se movimentam aleatoriamente, $k_u = K$, no pior caso, e consequentemente a operação *infectar*, e portanto o laço principal do simulador, tem custo $\mathcal{O}(K \log K)$.

Repare que no laço principal, a simulação termina quando o tempo de simulação alcança o tempo limite de simulação T , de forma que a complexidade do simulador

seja linear no tempo de simulação (dado todos os outros parâmetros do modelo). Entretanto, o número de eventos processados por unidade de tempo de simulação depende dos parâmetros do modelo, tais como taxa de caminhar, taxa de recuperar, número de indivíduos e tamanho da rede. Ou seja, o tempo de execução necessário para simular um tempo T de simulação depende dos parâmetros do modelo, mas dado os parâmetros, este tempo de execução é linear em T .

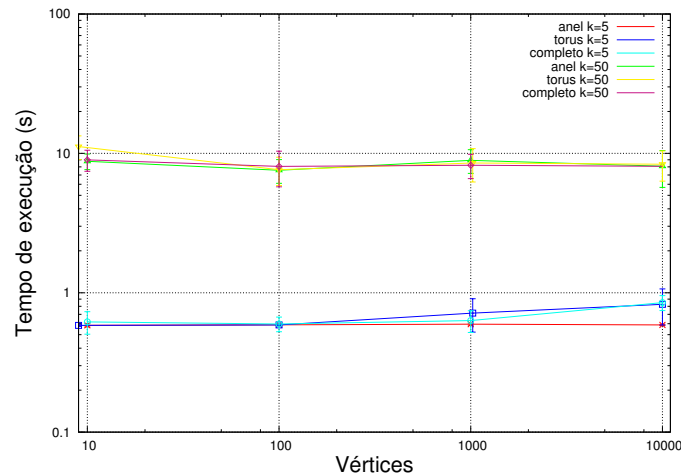


Figura 1. Tempo execução em função do tamanho da rede, n , para diferentes redes e número de indivíduos ($K = 5$, $K = 50$) em escala log-log. Parâmetros: $T = 10000$, $\lambda = 1$, $\gamma = 1$, $\tau = 1$.

5.1. Tempo de execução

Nesta seção iremos apresentar o tempo de execução do simulador (tempo de relógio de parede) para diferentes cenários. Os tempos foram coletados em uma CPU Intel Xeon ilustram a escalabilidade do simulador, sendo os valores de menor importância. Para cada cenário, 30 rodadas independentes foram realizadas, obtendo-se a média do tempo de execução, que é apresentada nos gráficos a seguir.

A figura 1 apresenta o tempo de execução em função do número de vértices da rede, para diferentes redes e diferentes números de indivíduos em cada cenário. Repare que o tempo de execução é constante no tamanho da rede, e independe da estrutura da rede. Para redes com maior número de indivíduos ($K = 50$), o tempo aumenta com relação ao caso com menos indivíduos ($K = 5$), mas permanece constante com o aumento da rede.

A figura 2 apresenta o tempo de execução do simulador em função do tempo limite de simulação (T) para três tipos de rede, de epidemias com $K = 5$ e $K = 50$ indivíduos. Note que o desempenho do simulador é linear no tempo de simulação, corroborando a análise teórica, pois em escala log – log, a reta de inclinação 1 corresponde a uma relação linear na escala convencional. Os resultados também indicam que este tempo não depende da estrutura da rede (retas praticamente idênticas para as três redes). Entretanto, repare que o cenário com 50 indivíduos demanda praticamente 10 vezes mais tempo de execução

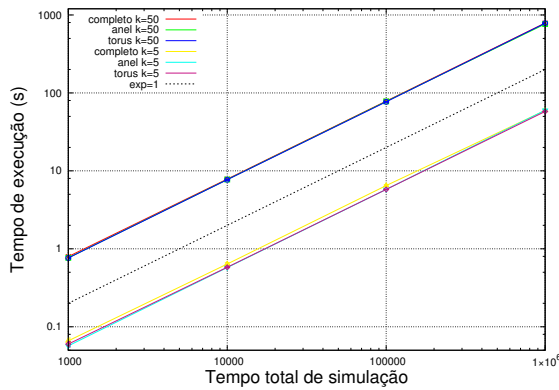


Figura 2. Tempo de execução em função do tempo de simulação (T), em escala log-log. Parâmetros:
 $K = \{5, 50\}$, $n = 25$, $\lambda = 1$, $\gamma = 1$, $\tau = 1$

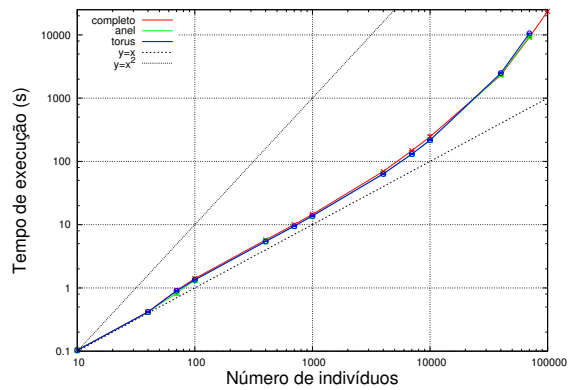


Figura 3. Tempo de execução em função do número de indivíduos na rede (K), em escala log-log (retas com inclinação 1 e 2 apenas para referência). Parâmetros:
 $T = 1000$, $n = 50$, $\lambda = 1$, $\gamma = 1$, $\tau = 1$

do que com 5 indivíduos. Este comportamento é esperado, pois teremos aproximadamente 10 vezes mais eventos a serem processados por unidade de tempo de simulação.

A Figura 3 apresenta o tempo de execução do simulador em função do número de indivíduos (K) para três redes. Nota-se que, para K pequeno (menor que 1000), o simulador apresenta uma escalabilidade praticamente linear (reta com inclinação 1 na escala log – log). Entretanto, conforme K aumenta (para valores maiores que 1000), a inclinação da curva representando o tempo de execução também aumenta. Este comportamento está de acordo com a análise teórica de pior caso, que tem complexidade $\mathcal{O}(K \log K)$. Repare que para valores grandes de K , o fator $\log K$ faz diferença. Novamente, observamos que o desempenho do simulador não depende da estrutura da rede, já que as curvas são praticamente idênticas para as três redes.

Referências

- Barrett, C. L., Bisset, K. R., Eubank, S. G., Feng, X., and Marathe, M. V. (2008). Epi-simdemics: An efficient algorithm for simulating the spread of infectious disease over large social networks. In *ACM/IEEE Conf on High Performance Computing (SC)*.
- Draief, M. and Ganesh, A. (2011). A random walk model for infection on graphs: spread of epidemics and rumours with mobile agents. *Discrete Event Dynamic Systems*, 21:41–61.
- Ganesh, A., Massoulié, L., and Towsley, D. (2005). The effect of network topology on the spread of epidemics. In *IEEE Infocom*, pages 1455–1466.
- Pastor-Satorras, R. and Vespignani, A. (2001). Epidemic spreading in scale-free networks. *Phys. Rev. Lett.*, 86:3200–3203.
- Zhou, J. and Liu, Z. (2009). Epidemic spreading in communities with mobile agents. *Physica A: Statistical Mechanics and its Applications*, 388:1228–1236.

Uma Análise de Desempenho sobre uma MicroFog

Douglas Siqueira*, Paulo H. Rocha*, Tomaz Bisneto*, Paulo A. L. Rego[‡] e
Francisco A. Silva*

¹* Núcleo de Pesquisas Avançadas em Sistemas Distribuídos (NUPASD)
Universidade Federal do Piauí (UFPI)

[‡]Grupo de Redes de Computadores, Engenharia de Software e Sistemas (GREat)
Universidade Federal do Ceará (UFC)

faps@ufpi.edu.br, pauloalr@ufc.br

Resumo. *A Computação em Nuvem é um paradigma de distribuição de recursos de processamento e armazenamento pela Internet. Uma área relacionada é a Fog Computing, que é uma Cloud com a proximidade ao usuário final. Pelo fato de que servidores tradicionais adotados na Computação em Nuvem possuem elevado preço e consumo energético. Este trabalho apresenta uma avaliação de desempenho de uma MicroFog (uma Fog de pequeno porte), formada por microcontroladores Raspberry Pi, comparada com servidores tradicionais. Foi utilizada um algoritmo de processamento de imagens e realizada uma análise de sensibilidade dos fatores: (A) servidores (tradicionais e MicroFog), (B) carga de trabalho e (C) tamanho da imagem.*

1. Introdução

A Computação em Nuvem é um paradigma voltado a resolver problemas de armazenamento e processamento de dados computacionais [Mansur et al. 2010]. As aplicações baseadas no conceito de Nuvem, descentralizam o processamento de um único computador, assim necessitando de servidores, na maioria das vezes bastante potentes. O que gera uma grande parte dos investimentos dentro da arquitetura da nuvem.

Os altos preços de tais equipamentos dificultam a sua aquisição por empresas de pequeno porte [Wilcox et al. 2015]. A busca por redução de custos é um dos itens dominantes no projeto das infraestruturas dos serviços em nuvem [Verdi et al. 2010]. Outro problema em decorrência do uso dos servidores é o seu alto gasto de energia. De acordo com [Shehabi et al. 2016] em 2014, os *data centers* nos EUA consumiram cerca de 70 bilhões de *Killowatts*, representando cerca de 1,8% do consumo total de energia dos EUA.

Fog Computing é um paradigma que estende a *Cloud* e seus serviço, levando-os para a borda da rede [Patil 2015]. O que evita o alto gasto energético. Por esta razão, o presente trabalho desenvolve uma *MicroFog*, com microcontroladores Raspberry e o compara com outros servidores. Além de utilizar o método *Design of Experiments* e as métricas consumo de energia, CPU e tempo de resposta para analisar os resultados.

O restante deste trabalho está estruturado da seguinte forma: A Seção 2 mostra como foram conduzidos os experimentos e seus respectivos resultados. Na Seção 3, são detalhados os trabalhos relacionados. E por fim, a Seção 4 apresenta as conclusões e trabalhos futuros.

2. Análise de Desempenho

Esta seção apresenta o ambiente de desenvolvimento e da análise comparativa, como também dos resultados obtidos com o presente trabalho.

2.1. Ambiente e Metodologia de Avaliação

Para este trabalho foi adotado um *Broker* de Mensagens (o RabbitMQ) utilizado para o envio das imagens. As imagens foram segmentadas usando a biblioteca *OpenCV*. O envio e a segmentação das imagens foram realizados 48 vezes para cada grupo de imagens com mesmo tamanho. Na Tabela 1 detalha os fatores e níveis escolhidos para a avaliação de desempenho do trabalho.

Tabela 1. Fatores e Níveis da Avaliação de Desempenho

Fatores	Níveis
Servidores	Servidor 01 (Core i3 4 Núcleos 2,5 Ghz e 7,7 GB de memória RAM)
	Servidor 02 (Core i3 4 Núcleos 1,7 Ghz e 4,0 GB de memória RAM)
<i>MicroFog</i> de Raspberries	
Tamanho da Imagem	Imagem 110 KB
	Imagem 320 KB
Carga	15 Imagens
	30 Imagens

Durante os experimentos três métricas foram observadas: consumo de CPU, tempo em segundos, e o gasto de energia em *Kilowatt* e o preço coletado em diferentes sites de *e-commerce* no Brasil. A *MicroFog* proposta neste trabalho possui três Raspberries. Com estes é coletado o tempo de processamento e gasto de CPU por instrumentação de código e o gasto energético é mensurado por um equipamento chamado Wanf WF-D02A¹.

2.2. Resultados

Esta seção apresenta resultados e discussões sobre os experimentos.

2.2.1. Análise Focando na Carga de Trabalho

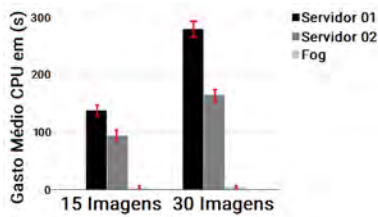
A Figura 1 apresenta um conjunto de gráficos variando as métricas de interesse (CPU, energia e tempo de processamento). Nos resultados obtidos através do consumo de energia (Figuras 1(a) e 1(b)), a *MicroFog* possui menores valores já o Servidor 02 e Servidor 01 obteve um pequeno aumento.

As Figuras 1(c) e 1(d) nestes gráficos ocorreu o melhor desempenho da *MicroFog* com relação aos demais equipamentos. Nas Figuras 1(e) e 1(f) exibem os resultados do tempo de processamento. Há um destaque maior no Servidor 01, que obteve um menor tempo por ter o melhor poder computacional.

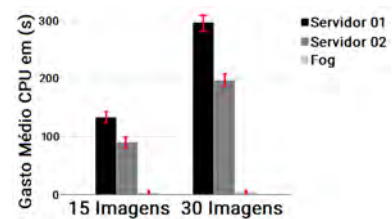
As Figuras 1(e) e 1(f) exibem os resultados do tempo de processamento. A *MicroFog*, devido ao seu *hardware* limitado, leva um tempo maior para processar as imagens. Esta mesma explicação é aplicada ao Servidor 02. Já o Servidor 01, por possuir um poder computacional mais elevado, acaba tendo grande vantagem nesse critério.

A Tabela 2 que mostra o preço médio dos diferentes equipamentos coletados em diferentes sites de *e-commerce* populares no Brasil. A partir da média obtida desta análise,

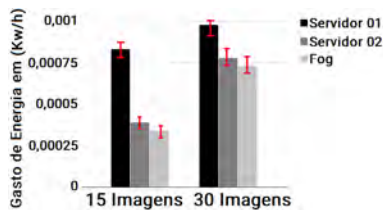
¹<http://solarlab.se/solpanel/datablad/WF-D02A-energy-meter.pdf>



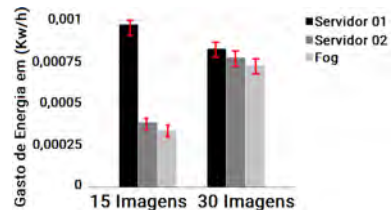
(a) Consumo da CPU com Imagem de 110 kB



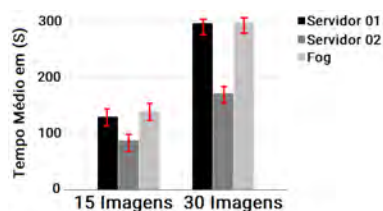
(b) Consumo da CPU com Imagem de 320 kB



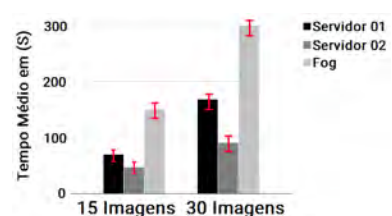
(c) Consumo de Energia com Imagem de 110 kB



(d) Consumo de Energia com Imagem de 320 kB



(e) Tempo de Processamento em Segundos com Imagem 110kB



(f) Tempo de Processamento em Segundos com Imagem 320kB

Figura 1. Comparação das Arquiteturas variando a Carga de Trabalho

pode ser determinado que a *MicroFog* de Raspberries é o equipamento mais viável com relação ao baixo custo.

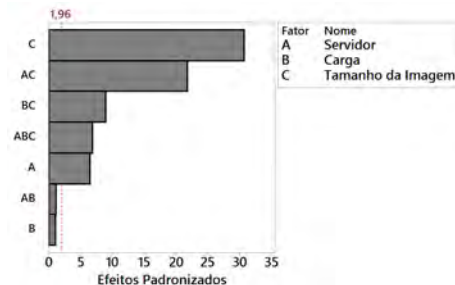
Tabela 2. Preço dos Equipamentos.

Dispositivos	Preço R\$
Servidor 01	1849.50
Servidor 02	1924.50
<i>MicroFog</i>	656.25

2.2.2. Análise Focando nos Fatores

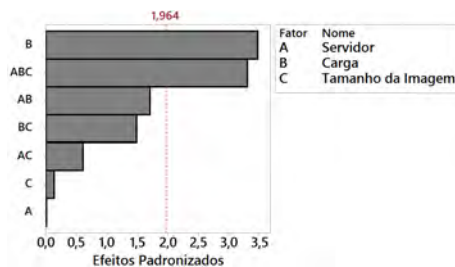
Nas Figura 2, 3 e 4 apresentam um *Design of Experiments* (DoE) trata-se de uma técnica de planejamento e análise de experimentos. Nas Tabelas 4, 5 e 6 mostra os valores de efeito obtidos com o DoE. A Figura 2 mostra os efeitos dos fatores sobre o tempo de processamento das imagens pela CPU. Assim como a Tabela 4. O principal efeito desta foi o Tamanho da Imagem. A Figura 3 e a Tabela 5 ilustra os efeitos dos fatores sobre o tempo de execução das imagens. Apesar de apenas dobrar o número de imagens isso não causou um grande efeito. Tendo como o principal a carga neste caso Enquanto a Figura 4 e a Tabela 6 apresentam os efeitos dos fatores sobre a Energia em *Kilowatts*. Neste caso,

o fator de maior impacto foi o Tamanho da Imagem. Portanto, deve-se atentar-se se a imagem é grande ou pequena.



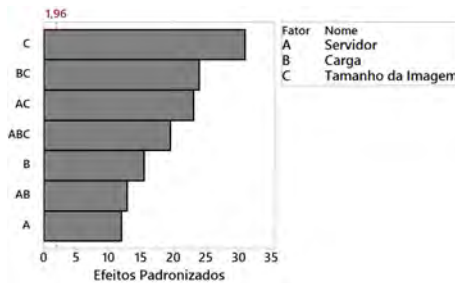
Fatores	Efeitos
A	-46,86
B	-0,693
C	2,0140
A+B	0,352
A+C	-0,6605
C+B	0,02464
A+B+C	-0,008749

Figura 2. e Tabela 4. Efeitos em Relação a CPU



Fatores	Efeitos
A	0,4
B	19,71
C	-0,074
A+B	-4,48
A+C	0,157
C+B	-0,0353
A+B+C	0,03621

Figura 3. e Tabela 5. Efeitos em Relação ao Tempo de Execução



Fatores	Efeitos
A	17,37
B	21,06
C	37,59
A+B	27,18
A+C	17,45
C+B	27,54
A+B+C	22,68

Figura 4. e Tabela 6. Efeitos em Relação a Energia em *Kilowatts*

2.2.3. Análise Focando nos Níveis

A Figura 5 apresenta gráficos de interação. Nas quais são identificadas por linhas não paralelas. Quanto maior for a inclinação entre as linhas, maior o grau de interação. Linhas paralelas indicam ausência de interação entre os fatores.

O gráfico da Figura 5(a) apresenta a maior interação em termos de Gasto de CPU entre os fatores Carga e Servidor. Assim quando a MicroFog é adotada o impacto da Carga é mais notado. Na Figura 5(b) as três combinações possuem uma certa interação. Visualmente, a inclinação maior é combinando Tamanho da Imagem com Servidor. A Figura 5(c) mostra a interação dos fatores para Energia em *Kilowatts*. Neste caso, os valores estão condizentes com as tabelas de efeitos apresentadas anteriormente.

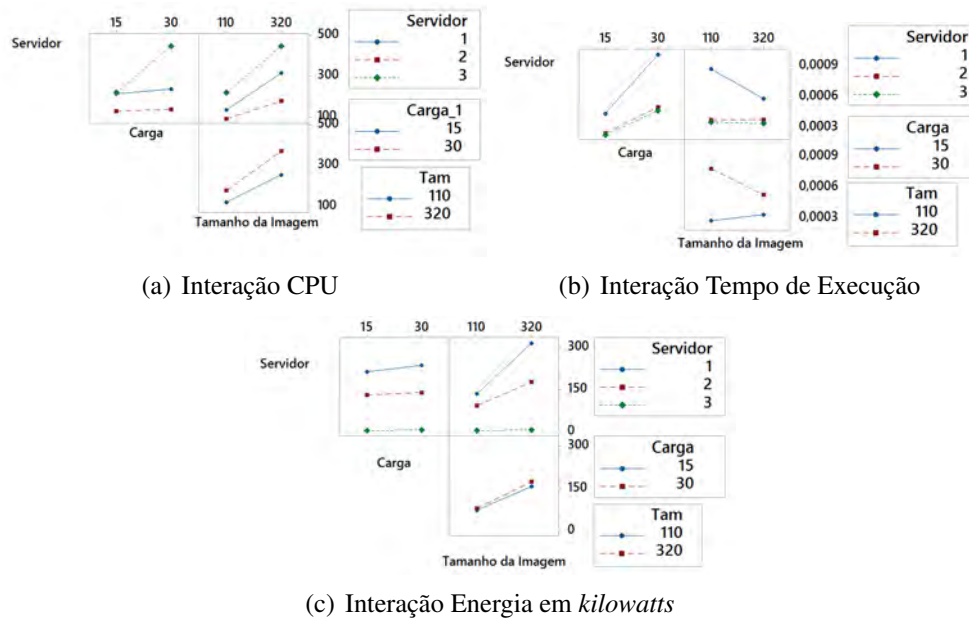


Figura 5. Interação dos Níveis dos Fatores

3. Trabalhos Relacionados

Esta seção apresenta um levantamento sobre os principais trabalhos que criaram *Clouds*, *Fogs* ou *Clusters* com microcontroladores Raspberry Pi ou outros tipos de microcontroladores. No trabalho [Kiepert 2013] foi desenvolvido um *cluster* com 32 Raspberries. Dentre eles, estavam simuladores de Raspberries para aumentar o poder computacional do *cluster*. No trabalho [Abrahamsson et al. 2013] foi realizada uma *cloud* com Raspberries. Como também foi desenvolvido um sistema para controle de cada nó Raspberry.

[Tso et al. 2013] apresenta a Glasgow Raspberry Pi Cloud (PiCloud), um modelo em escala de um *Data Center* composto de Raspberries. Em [Cloutier et al. 2014] desenvolveu uma *cloud* com 8 Raspberries e os comparou com outros equipamentos. Dentre eles, placas de *desktops*, utilizando como métrica principal a CPU dos equipamentos. No trabalho de [Franks and Yerby 2014] há um detalhamento dos desafios da criação de um supercomputador e possibilidades de baixo custo para que a criação seja usada para a indústria.

No artigo de [Princy and Nigel 2015] ocorreu a implementação de uma *Cloud* de armazenamento de arquivos em tempo real. Isto foi possível utilizando uma aplicação chamada OwnCloud². Em [Jalali et al. 2016] percebeu-se que aplicar uma *cloud* com Raspberries é eficiente em se tratando de aplicações dentro da Internet das Coisas.

4. Conclusão

Com o crescente uso das tecnologias em Nuvem, a necessidade de *hardware* e tecnologias relacionadas está sempre aumentando. A realidade financeira é um grande empecilho que torna a aquisição de equipamentos algo custoso. Assim, este trabalho teve como objetivo desenvolver uma *MicroFog* com microcontroladores Raspberry Pi e analisar o seu desempenho comparando-a com servidores de pequeno porte. Como resultado a *MicroFog* gasta

²<https://owncloud.org>

menos energia. Na análise de desempenho permitiu analisar 3 fatores: (A) servidores (tradicionais e MicroFog), (B) carga de trabalho e (C) tamanho da imagem. Os resultados apontam que a escolha do servidor teve maior impacto tanto no tempo de execução quanto no gasto de energia. Como trabalho futuro, pretende-se realizar uma análise com outros fatores, como níveis de usuários e requisições simultâneas.

Referências

- Abrahamsson, P., Helmer, S., Phaphoom, N., Nicolodi, L., Preda, N., Miori, L., Angri-man, M., Rikkila, J., Wang, X., Hamily, K., et al. (2013). Affordable and energy-efficient cloud computing clusters: the bolzano raspberry pi cloud cluster experiment. In *Cloud Computing Technology and Science (CloudCom), 2013 IEEE 5th International Conference on*, volume 2, pages 170–175. IEEE.
- Cloutier, M. F., Paradis, C., and Weaver, V. M. (2014). Design and analysis of a 32-bit embedded high-performance cluster optimized for energy and performance. In *Proceedings of the 1st International Workshop on Hardware-Software Co-Design for High Performance Computing*, pages 1–8. IEEE Press.
- Franks, S. and Yerby, J. (2014). Creating a low-cost supercomputer with raspberry pi. In *Proceedings of the Southern Association for Information Systems Conference*.
- Jalali, F., Hinton, K., Ayre, R., Alpcan, T., and Tucker, R. S. (2016). Fog computing may help to save energy in cloud computing. *IEEE Journal on Selected Areas in Communications*, 34(5):1728–1739.
- Kiepert, J. (2013). Creating a raspberry pi-based beowulf cluster. *Boise State University*, pages 1–17.
- Mansur, A. F. U., Gomes, S. S., Lopes, A. d. A., and Biazus, M. (2010). Novos rumos para a informática na educação pelo uso da computação em nuvem (cloud education): Um estudo de caso do google apps. In *Foz do Iguaçu: Anais do XVI Congresso Internacional ABED de Educação a Distância*.
- Patil, P. V. (2015). Fog computing. *College of Engg & Tech*.
- Princy, S. E. and Nigel, K. G. J. (2015). Implementation of cloud server for real time data storage using raspberry pi. In *Green Engineering and Technologies (IC-GET), 2015 Online International Conference on*, pages 1–4. IEEE.
- Shehabi, A., Smith, S., Sartor, D., Brown, R., Herrlin, M., Koomey, J., Masanet, E., Horner, N., Azevedo, I., and Lintner, W. (2016). United states data center energy usage report.
- Tso, F. P., White, D. R., Jouet, S., Singer, J., and Pezaros, D. P. (2013). The glasgow raspberry pi cloud: A scale model for cloud computing infrastructures. In *Distributed Computing Systems Workshops (ICDCSW), 2013 IEEE 33rd International Conference on*, pages 108–112. IEEE.
- Verdi, F. L., Rothenberg, C. E., Pasquini, R., and Magalhaes, M. (2010). Novas arquiteturas de data center para cloud computing. *Minicursos do XXVIII SBRC*, pages 103–152.
- Wilcox, E., Jhunjhunwala, P., Gopavaram, K., and Herrera, J. (2015). Pi-crust: a raspberry pi cluster implementation. Technical report, Technical report, Texas A&M University.

Uso de Simulador de Geração de Tráfego para a Melhoria de Desempenho de Sistemas Legados Críticos

Eduardo Teixeira¹, Mateus Manuel¹, Renan Costa¹

¹ Universidade de Brasília (UnB) – Brasília, DF – Brazil

{eduhenr, mateusmanuel, renanrf}@unb.br

Abstract. *Some legacy systems tend to have a barely scalable architecture, making little progress on performance issues. On the other hand, access to the Internet has been increasing, making more people connected and therefore, forcing systems to maintain efficiency with the new demand. This work presents an architectural change carried out in the legacy registration system of the University of Brasília aiming at performance improvement. It is also shown a traffic generator (TRAFFICGEN), built to subject the system to a load similar that occurs during enrollment periods and thus various improvements were made. It has been found that the modified system has achieved a performance improvement of more than 5× compared to the production version with half the servers used and still supports a load increase of at least 215 % without performance degradation.*

Resumo. *Alguns sistemas legados tendem a possuir uma arquitetura pouco escalável, fazendo com que pouco se possa evoluir em questões de desempenho. Por outro lado, o acesso a internet vem crescendo, fazendo com que mais pessoas tenham acesso e obrigando, conseqüentemente, que os sistemas evoluam para manter a eficiência com a nova demanda. Nesse trabalho é apresentada uma mudança arquitetural realizada no sistema legado de matrículas da Universidade de Brasília visando a melhoria de desempenho. Também é demonstrado um gerador de tráfego (TRAFFICGEN) construído para submeter o sistema a uma carga semelhante à que ocorre durante os períodos de matrícula e dessa forma fossem realizadas diversas melhorias. Foi verificado que o sistema modificado alcançou uma melhoria de desempenho de mais de 5× com relação à versão de produção com a metade dos servidores utilizados e que ainda suporta um aumento de carga de pelo menos 215% sem degradação do desempenho.*

1. Introdução

Grandes investimentos são realizados em sistemas de software e para que as organizações obtenham retorno desse investimento, o software deve ser utilizado por vários anos. O tempo de duração de sistemas de software é variável e pode permanecer por mais de 10 anos [Lehman 1980]. Muitos desses sistemas ainda são fundamentais para as organizações e qualquer falha desses serviços acarretará um sério efeito em seu dia a dia [Bennett and Rajlich 2000].

Ao longo dos últimos, em particular após o REUNI [Brasil 2007], a Universidade de Brasília (UnB) aumentou consideravelmente o quantitativo de cursos de graduação e,

consequentemente, o número de alunos. Por outro lado, os sistemas apoio, como o MW, não acompanharam este ritmo de evolução e hoje apresentam deficiências no que tange ao desempenho durante a execução de atividades críticas como o processo de matrícula.

Neste processo, em uma das etapas finais, os alunos concorrem pelas vagas existentes obedecendo o critério *First Come First Served* [Tanenbaum and Wetherall 2010] (FCFS), tendo em vista haver um número maior de alunos do que vagas disponíveis em alguns cursos. Este critério, por sua vez faz com que haja um contingente de demanda ao servidor que supera 500 conexões concorrentes, o que por sua vez acarreta em problemas de indisponibilidade e lentidão no atendimento destas demandas. Este trabalho apresenta em detalhes as ações e técnicas utilizadas com o intuito de melhorar o desempenho deste sistema legado e permitir não apenas a sobrevida do sistema mas principalmente atender com maior celeridade as demandas em momentos de pico como o que ocorre durante o período de matrícula.

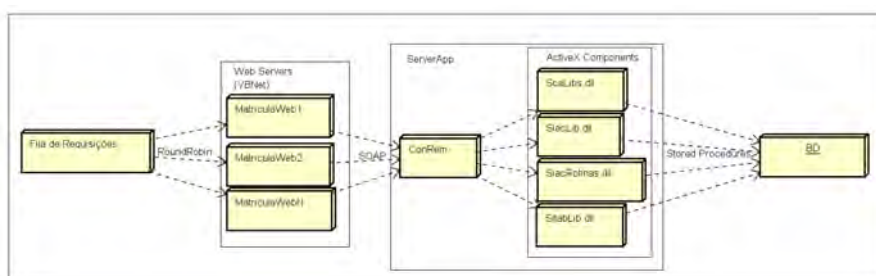


Figura 1. Arquitetura de Produção do MW.

Foi realizado o estudo da arquitetura de produção do MW, e foram identificados alguns pontos críticos. Um deles está relacionado ao tratamento da fila de requisições, que após o *round robin* entre os servidores Web (*WebServers*), é realizado por meio de um único servidor, o *ServerApp*, conforme mostrado na Figura 1. O *ServerApp* é um servidor de aplicação que contém o *WebService ConnectionRemote* (*ConRem*) que se comunica com os *WebServers*. O *ConRem* possui diversos componentes *ActiveX* que realizam o acesso ao banco de dados por meio de *stored procedures* que contém as regras de negócio da aplicação. Dessa forma, para que as operações sejam realizadas no servidor de banco de dados é necessário enviar as requisições ao *ConRem* por meio do protocolo *SOAP*, que por sua vez acessa os componentes *ActiveX* que finalmente realizam chamadas para as *stored procedures*. Ou seja, existem limitações na escalabilidade da aplicação devido ao processamento de todas as requisições serem realizadas por um único servidor e também devido ao alto grau de *overhead* para que a aplicação persista os dados em banco. Além disso, caso ocorra uma queda ou um alto volume de requisições no *ServerApp* ocorre o comprometimento da qualidade de serviço e até mesmo do funcionamento para os usuários que estejam utilizando o sistema.

Um outro ponto crítico encontrado nessa arquitetura se deve ao fato de que todas as regras de negócio do sistema se encontram em *stored procedures* escritas na linguagem *Transact-SQL* (T-SQL). Essas *stored procedures* são chamadas em cadeia e utilizam tabelas temporárias para a comunicação entre elas. O acesso concorrente de muitas chamadas a *stored procedures* gera um alto consumo de CPU no *ServerApp* e consequentemente lentidão no processamento da fila de requisições. Além disso, o uso de *stored proce-*

dures para regras de negócio dificulta o entendimento e a depuração do código, pois as ferramentas de *debugging* para a linguagem T-SQL são muito limitadas, podendo produzir resultados diferentes dos ambientes próprios de linguagens de programação. Devido a esse modo de funcionamento, é frequente o estouro de vagas em ofertas de disciplinas sendo necessária a intervenção manual dos coordenadores dos departamentos para realizar os ajustes necessários para cada disciplina ofertada.

2. Métodos

Foi proposta uma arquitetura alternativa ao modelo atual mostrado na Figura 1. Nessa arquitetura foi retirada da aplicação a comunicação com o Webservice ConRem, por ser identificado arquiteturalmente como um ponto único de gargalo e de falha. Por meio dessa retirada existe uma maior abertura para o paralelismo nas execuções das consultas ao banco de dados que podem ser realizadas diretamente pela aplicação. Além disso, caso ocorra a falha de um WebServer não ocorre o comprometimento de todo o sistema, afetando apenas os usuários conectados no IIS em que ocorreu a queda [Coulouris et al. 2011].

Além da retirada do ConRem foi realizada a conversão do projeto e de suas dependências para o compilador Visual Studio 2015 com o objetivo de melhorar o desempenho do binário executável gerado e da sua utilização com a versão mais recente do IIS (Internet Information Server 8.5). O IIS 8.5 realiza o gerenciamento de múltiplas conexões simultâneas de forma mais otimizada e pode ser utilizado com um sistema operacional mais recente, como o Windows 2012 R2, que também possui um melhor desempenho para gerenciamento de recursos de I/O e CPU.

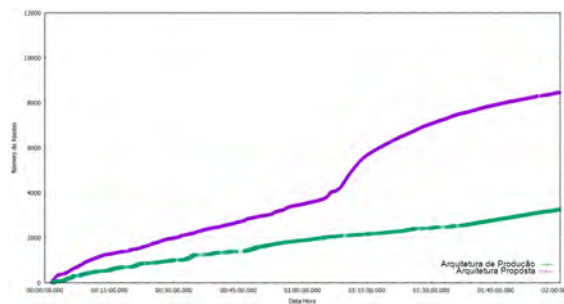
Também foi realizada a conversão do código de T-SQL para VBNet por ser a mesma linguagem utilizada no *site* e com o objetivo de melhorar o entendimento das regras de negócio e da depuração do código, estudar potenciais pontos de melhoria e efetuar manutenções evolutivas e corretivas mais rapidamente. Entretanto, o uso de tabelas temporárias foi mantido pois a sua conversão implicaria em um alto risco devido a grande quantidade de regras de negócio envolvidas.

Foi realizada a modificação de todas as consultas em banco (*select*) utilizando a cláusula *with(nolock)* para diminuição das serializações em função da grande quantidade de consultas realizadas pelo processo de matrícula. Também foi utilizada a estratégia de *locks* de linha *with(updlock)* transacional para atualização das tabelas de oferta e reserva de vagas em disciplinas para evitar o estouro de vagas.

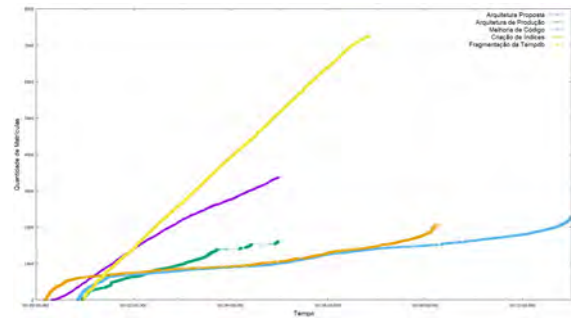
3. Resultados

A arquitetura proposta foi implantada em produção no segundo período de 2017 (2017/2). Foram utilizados 12 WebServers com Windows Server 2012 R2, 16 *gigabytes* de memória e 16 núcleos cada. Devido a conversão de código existente de T-SQL para VBNet ocorreram 2 *bugs* no período de matrícula, por conta do comportamento específico da linguagem T-SQL ao tratar variáveis não inicializadas. Entretanto, esses problemas foram corrigidos rapidamente e as atualizações em ambiente de produção foram efetuadas no mesmo dia da requisição, o que comprovou o alcance da melhoria almejada com relação ao melhor entendimento e correções rápidas. Além disso, também verificou-se que não ocorreram mais os estouros de vagas, pois a crítica da matrícula para a aquisição de vagas é realizada

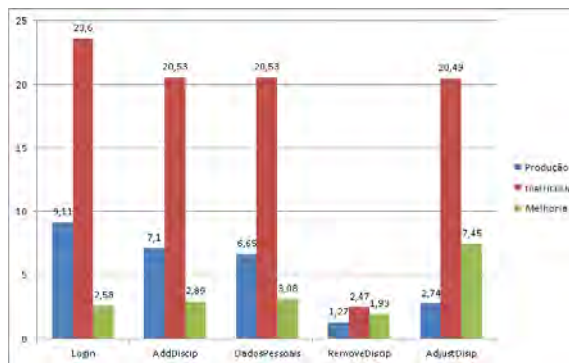
atomicamente. Para a Arquitetura Proposta foram obtidos os resultados da Figura 2(a). Apesar da melhoria de desempenho do sistema com a "Arquitetura Proposta" ser $2.59\times$ superior nas primeiras 2 horas, ocorreram problemas de *timeout* de banco, o que gerou lentidão e erros de acesso na primeira hora. Isso foi constatado pelo comportamento linear da "Arquitetura Proposta" com relação à quantidade de matrículas efetuadas no tempo e ocorreu devido ao grande volume de requisições concorrentes dos estudantes pelas vagas remanescentes.



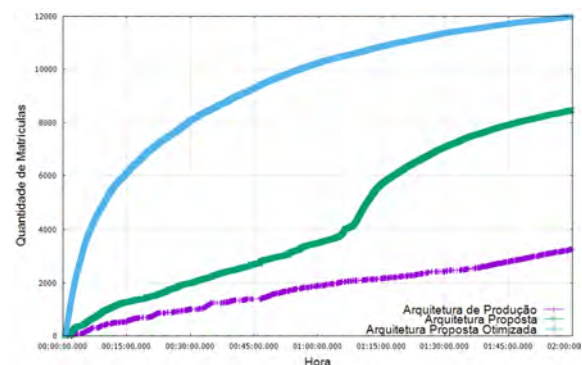
(a) Desempenho das Arquiteturas de Produção e Proposta.



(b) Resultados Obtidos com os Testes de Stress.



(c) Resultados Obtidos com os Testes de Stress



(d) Desempenho das Arquiteturas de Produção e Proposta e Otimizada.

Figura 2. Comportamento da Quantidade de Matrículas no Tempo.

Devido a esse problema de desempenho que ocorreu na primeira hora de funcionamento da Arquitetura Proposta foi desenvolvido um simulador escrito na linguagem C++ para gerar tráfego com um volume de requisições parecido com o que ocorre durante o sistema em produção. Esse gerador de tráfego, denominado TRAFFICGEN, realiza a carga para a memória de todas as matrículas do período 2017/2 e realiza os POSTs HTTP utilizando várias conexões TCP concorrentes originadas de várias *threads* e processos que se comunicam por meio de *Message Passing Interface* (MPI) [Cameron Hughes 2003]. Os *buffers* dos POSTs foram capturados com o *sniffer Wireshark* durante a operação das funcionalidades consideradas mais importantes: *login*; obtenção de dados pessoais; remoção de disciplinas; adição de disciplinas, sendo essa a mais crítica. Dessa forma, esses *buffers* puderam ser enviados mudando apenas as informações relativas ao número de matrícula, disciplina e turma de cada aluno. Com o uso do TRAFFICGEN foram realizados os testes de eficiência e de *stress*.

Com os testes de *stress* submetidos por meio do TRAFFICGEN foi possível en-

contrar diversos problemas de desempenho a cada bateria de testes e realizar vários ajustes para resolvê-los, como é possível observar na Figura 2(b). O envio de requisições do TRAFFICGEN para os testes de *stress* foi realizado em rajada durante 5 minutos, sendo encerrada a geração de tráfego após esse intervalo, porém esperando o resultado das requisições enfileiradas até o término de todas as *threads* e processos iniciados, demorando cerca de 11 minutos para a execução no pior caso, como pode ser visto na Figura 2(b). Dessa forma, na Figura 2(b) o eixo *x* corresponde ao tempo em minutos da geração de tráfego e espera pelo término das *threads* e processos iniciados pelo TRAFFICGEN, enquanto que o eixo *y* corresponde ao número de matrículas realizadas. Foi utilizado o intervalo de 5 minutos pois este tempo foi suficiente para verificar o comportamento do tempo médio e executar várias baterias de testes por ser um tempo curto. Entre as melhorias efetuadas as que tiveram os melhores resultados foram a melhoria do código-fonte (em azul), a criação de novos índices em tabelas (em laranja) e a fragmentação do banco de dados responsável pelo tratamento de tabelas temporárias (em amarelo). Na Figura 2(b) também é possível observar e comparar os resultados das otimizações realizadas com o comportamento das Arquiteturas de Produção (em verde) e Proposta (em roxo) durante o funcionamento durante os primeiros 5 minutos que ocorreram na Figura 2(a). Entre as melhorias de código-fonte foram realizadas o *refactoring* da rotina de *login*, a retirada do uso de *views* e a reescrita das consultas especificando as colunas necessárias ao invés de todas com o uso de "*", sendo esta última um uso recomendado pelas melhores práticas no que tange ao SQL. Além disso, foi modificado o acesso aos cursores dos *resultsets* para o modo sem serialização e sem cópia. Cada uma dessas melhorias foi realizada cumulativamente na ordem: melhoria do código-fonte, criação de novos índices e a fragmentação das tabelas temporárias.

Como pode ser visto, o maior ganho obtido ocorreu com o uso de fragmentação das tabelas temporárias, pois apesar da simplificação arquitetural efetuada, o sistema modificado ainda utiliza intensamente tabelas temporárias para a realização das regras de negócio. Durante o tempo em que o teste de *stress* do TRAFFICGEN gerou carga foram realizadas 7.264 matrículas no melhor caso para a modificação "Fragmentação da TempDb"(em amarelo na Figura 2(b)), enquanto que na "Arquitetura Proposta" foram realizadas 3.368 matrículas (em roxo na Figura 2(b)), sendo este último obtido dos primeiros 5 minutos de funcionamento real em ambiente de produção obtido da Figura 2(a). Dessa forma, o sistema ainda suporta um aumento de carga de pelo menos 215%, sem degradação do desempenho. O aumento de carga suportado pode até ser maior que os 215%, pois além de o ambiente de produção possuir um poder computacional maior do que o do TRAFFICGEN, a carga real a qual o sistema foi submetido foi inferior à do ambiente do gerador de tráfego, sendo este submetido a 1440 conexões simultâneas, enquanto que o sistema real alcançou o pico de 520 conexões simultâneas.

No ambiente do TRAFFICGEN esses ajustes alcançaram uma melhoria $7.45\times$ superior para a funcionalidade mais crítica (adição de disciplina) e em média $2.6\times$ superior para as demais funcionalidades, como pode ser visto na Figura 2(c), onde as barras vermelhas e azuis correspondem ao número de operações por segundo (eixo *y*) para cada funcionalidade (eixo *x*) e a barra verde corresponde ao ganho obtido. Dessa forma, para a realização das matrículas no primeiro semestre letivo de 2018 (2018/1) foi realizada a diminuição dos WebServers de produção pela metade, por ser essa a menor média de ganho encontrada, como mostrado na Figura 2(c).

As otimizações realizadas com o apoio do gerador de tráfego TRAFFICGEN foram aplicadas na "Arquitetura Proposta" e foi obtido o resultado da "Arquitetura Proposta Otimizada", conforme mostrado na Figura 2(d), onde o eixo x corresponde à hora do dia a partir da liberação de funcionamento do sistema que ocorreu às 00:00hs e o eixo y corresponde ao número de matrículas efetuadas.

É possível observar que o sistema após as alterações efetuadas teve um comportamento logarítmico ao invés de linear com relação ao número de matrículas no tempo, levando a uma melhoria de desempenho $2.92\times$ superior na primeira hora se comparado com a "Arquitetura Proposta", e $5.41\times$ em relação à "Arquitetura de Produção".

4. Conclusão

A melhoria de desempenho em sistemas distribuídos nem sempre está associada ao aumento da capacidade de processamento do *hardware*, sendo que os maiores ganhos geralmente ocorrem em mudanças no modo de funcionamento da arquitetura desses sistemas [Tanenbaum and Steen 2008].

Este trabalho mostrou ser possível, com o auxílio de um simulador de geração de tráfego, simplificar a arquitetura e diminuir pela metade o número de servidores do sistema legado MW e mesmo assim obter um ganho de desempenho $5.41\times$ maior do que a antiga versão. Além disso, o sistema ainda suporta um aumento de pelo menos 215% na carga sem comprometer a qualidade no desempenho.

Dessa forma, foi possível dar uma sobrevida a um sistema legado crítico sem necessariamente reescrever toda a aplicação, o que teria um custo muito maior do que a escrita do TRAFFICGEN. Esses resultados alcançados geraram uma grande satisfação da comunidade acadêmica com o Sistema de Matrículas da Universidade de Brasília no primeiro período letivo de 2018.

Referências

- Bennett, K. H. and Rajlich, V. T. (2000). Software maintenance and evolution: a roadmap. In *Proceedings of the Conference on the Future of Software Engineering*, pages 73–87. ACM.
- Brasil (2007). Decreto nº 6.096 de 24 de abril de 2007a. http://www.planalto.gov.br/ccivil_03/_ato2007-2010/2007/decreto/d6096.htm. Accessed: 2017-03-28.
- Cameron Hughes, T. H. (2003). *Parallel and Distributed Programming Using C++*. Editora Addison Wesley, first edition.
- Coulouris, G., Dollimore, J., Kindberg, T., and Blair, G. (2011). *Distributed Systems: Concepts and Design*. Addison-Wesley Publishing Company, USA, 5th edition.
- Lehman, M. M. (1980). Programs, life cycles, and laws of software evolution. *Proceedings of the IEEE*, 68(9):1060–1076.
- Tanenbaum, A. S. and Steen, M. V. (2008). *Distributed systems: principles and paradigms*. Editora Pearson Prentice Hall, second edition.
- Tanenbaum, A. S. and Wetherall, D. J. (2010). *Computer Networks*. Editora Pearson Prentice Hall, fifth edition.

Modelos de Resposta para Experimentos Randomizados em Redes Sociais de Larga Escala*

Francisco Galuppo Azevedo, Bruno Demattos Nogueira,
Fabricio Murai, Ana Paula Couto da Silva

¹Departamento de Ciência da Computação
Universidade Federal de Minas Gerais (UFMG) – Belo Horizonte, MG – Brazil

{franciscogaluppo, bruno.demattos, murai, ana.coutosilva}@dcc.ufmg.br

Abstract. *A/B tests are randomized experiments frequently used by companies that offer services on the Web for assessing the impact of new features. During an experiment, each user is randomly redirected to one of two versions of the website, called treatments. Several response models were proposed to describe the behavior of a user in a social network website, where the treatment assigned to her neighbors must be taken into account. However, there is no consensus as to which model should be applied to a given dataset. In this work, we propose a new response model, derive theoretical limits for the estimation error of several models, and obtain empirical results for cases where the response model was misspecified.*

Resumo. *Testes A/B são experimentos randomizados muito utilizados por empresas que oferecem serviços na Web para avaliar o impacto de novas funcionalidades. Durante um experimento, cada usuário é redirecionado aleatoriamente para uma de duas versões do site, chamadas tratamentos. Diversos modelos de resposta foram propostos para descrever o comportamento de um usuário em sites de redes sociais, onde o tratamento atribuído aos seus vizinhos deve ser considerado. Porém, não há consenso sobre qual modelo deve ser aplicado a um conjunto de dados. Neste trabalho, propomos um novo modelo de resposta, derivamos limites teóricos para o erro de estimação de diversos modelos, e obtemos resultados empíricos para o caso de especificação incorreta do modelo.*

1. Introdução

Há alguns anos grandes empresas que oferecem serviços na Web (como Amazon, e-Bay, Facebook, Google e LinkedIn) vêm percebendo a importância de se conduzir pesquisas sobre a experiência dos usuários para a tomada de decisões, a nível de desenvolvimento (p. ex., mudanças no funil de compra ou no layout do site) e também a nível de negócio (p. ex., novas funcionalidades e diferenciais em relação aos concorrentes) [Kohavi et al. 2013]. Este impacto é quantificado através de medidas de interesse, tais como a fração de usuários que retorna ao site, o número médio de cliques por usuário, o lucro obtido através de anúncios no site etc. Para cada medida, deve-se estimar o efeito médio do tratamento, conhecido como *average treatment effect* (ATE).

*The authors' work has been partially funded by the EUBra-BIGSEA project by the European Commission under the Cooperation Programme (MCTI/RNP 3rd Coordinated Call), Horizon 2020 grant agreement 690116, CAPES, CNPq and FAPEMIG.

Dentre as técnicas existentes, testes A/B se destacam como uma das mais proeminentes devido a sua capacidade de quantificar mudanças comportamentais objetivamente, calcular resultados e determinar a significância estatística de forma automática (sem que seja necessário feedback explícito do usuário) [Kohavi et al. 2009]. Testes A/B consistem em experimentos randomizados com duas variantes de um tratamento. Usuários que visitam um website são aleatoriamente atribuídos ao grupo de controle (versão atual) ou ao grupo de tratamento (versão sendo testada).

Os testes A/B têm como premissa a *Stable Unit Treatment Value Assumption* (SUTVA). A SUTVA é a suposição de que não há interferência entre os indivíduos do experimento, i.e., que o comportamento de um indivíduo depende apenas do tratamento que lhe foi atribuído. Enquanto esta suposição é válida para testes clínicos envolvendo placebos e medicamentos, experimentos em redes sociais normalmente violam esta condição (efeito conhecido como *spillover*) [Xu et al. 2015].

Idealmente, gostaríamos de comparar os resultados de dois experimentos simultâneos: no primeiro, toda a população está no grupo de controle e, no segundo, todos estão no grupo de tratamento. Assim, mesmo que a SUTVA fosse violada, conseguiríamos estimar o ATE. Como isso não é possível, para se estimar o ATE é preciso assumir um modelo que descreve como a resposta de um indivíduo varia conforme o tratamento que é atribuído a ele e ao restante da população. No caso de redes sociais, assume-se que a resposta de um indivíduo é influenciada apenas pelos seus relacionamentos diretos.

Modelos de resposta são importantes não só para estimar o ATE, como também para comparar vários métodos de estimação quanto a sua acurácia. Como a função de resposta em um experimento real é desconhecida, a comparação entre métodos é feita a partir de dados sintéticos, gerados utilizando-se modelos de resposta. Tradicionalmente, os modelos de resposta utilizados são diferentes daqueles assumidos pelos métodos propostos. Contudo, para que os resultados em relação ao erro de estimação sejam colocados em perspectiva, é fundamental conhecer os limites teóricos do erro de estimação, caso o modelo correto fosse empregado.

Este trabalho possui três contribuições principais. A primeira consiste em propor um novo modelo de resposta, chamado τ -exposure, baseado em limiar de exposição. Neste modelo um indivíduo é considerado “exposto” se ele recebe o controle (tratamento) e uma fração de seus vizinhos maior que τ também recebe controle (tratamento). A segunda contribuição é a derivação analítica dos limites inferiores para o erro de estimação quando o modelo correto de interferência é utilizado. Finalmente, iremos utilizar diversos modelos de resposta para realizar um estudo empírico do erro resultante ao se especificar o modelo incorreto durante a estimação. A versão estendida deste trabalho está disponível em [Azevedo et al. 2018].

2. Trabalhos relacionados

Com o auxílio de um grafo é possível definir modelos de resposta computacionalmente tratáveis. Estes modelos fazem parte da classe *neighborhood treatment response*. Em alguns destes modelos, assume-se que um indivíduo é considerado “exposto” ao tratamento se ele e pelo menos k de seus vizinhos (ou uma fração maior que q) recebem o mesmo tratamento [Backstrom and Kleinberg 2011, Ugander et al. 2013]. Apesar de apresentarem resultados empíricos, nenhum destes trabalhos estuda os limites teóricos de estimação.

Em outro trabalho propõe-se um modelo linear para a resposta de um nodo em função do tratamento dele e dos seus vizinhos [Gui et al. 2015]. Para reduzir o impacto de supor incorretamente tal relação linear, os autores usam randomização a nível de cluster. Por outro lado, o modelo, que produz respostas reais, é aplicado a respostas binárias sem que haja um estudo sobre o impacto desta inconsistência. Embora diversos modelos de resposta tenham sido propostos para capturar a interferência, não existe uma regra para escolher aquele que melhor representa o conjunto de dados, tampouco um entendimento do erro que a especificação incorreta pode ocasionar. Atualmente, existe apenas um método que permite testar se a SUTVA é válida [Saveski et al. 2017], assim como calcular a probabilidade de erro do Tipo I (rejeitar a SUTVA quando ela é válida).

3. Modelos de resposta

Descrevemos a seguir a notação usada nos modelos de resposta. Seja cada unidade (ou indivíduo) do experimento indexada por $i \in 1, \dots, N$. Defina $\mathbf{Z} \in \mathcal{Z}^N$ e $\mathbf{Y} \in \mathcal{Y}^N$ como sendo os vetores de tratamento e de resposta, onde $Z_i \in \mathcal{Z}$ e $Y_i \in \mathcal{Y}$ são, respectivamente, o tratamento atribuído ao e a resposta do usuário i . Defina $g_i(\mathbf{Z})$ como sendo uma função específica à unidade i , calculada sobre a atribuição de tratamento $\mathbf{Z}$. Quando as unidades estão conectadas segundo uma topologia de rede, exemplo típicos de tais funções são a fração ou o número de vizinhos a que foram atribuídos um dado tratamento. Defina o vetor coluna $\mathbf{x}_i = [1, Z_i, g_i(\mathbf{Z})]^\top$. Em geral, existe um componente estocástico ϵ_i associado à unidade i , amostrado independentemente para $i = 1, \dots, N$ a partir da distribuição Gaussiana com média zero e variância desconhecida σ^2 . A seguir, iremos considerar apenas experimentos envolvendo dois tratamentos (testes A/B), logo $Z_i \in \mathcal{Z} \equiv \{0, 1\}$.

Modelo linear de resposta. Quando as respostas são números reais (i.e., $Y_i \in \mathcal{Y} \equiv \mathbb{R}$), o modelo linear de resposta costuma ser utilizado:

$$Y_i = \mathbf{x}_i^\top \boldsymbol{\beta} + \epsilon_i, \quad (1)$$

onde $\boldsymbol{\beta} = (\beta_0, \beta_1, \beta_2)$ são parâmetros do modelo.

Modelo probit. Uma forma de se produzir respostas binárias (i.e., $Y_i \in \mathcal{Y} \equiv \{0, 1\}$) a partir de uma função real (p. ex., $\mathbf{x}_i^\top \boldsymbol{\beta} + \epsilon_i$) consiste em usar o sinal da função para determinar a resposta. Este é caso do modelo probit:

$$\Pr(Y_i = 1 | \mathbf{x}_i) = \Phi(\mathbf{x}_i^\top \boldsymbol{\beta}) \quad (2)$$

onde $\Phi(\cdot)$ é a função de distribuição cumulativa da Normal de média 0 e variância σ^2 .

Modelo logístico. Outra forma de se produzir respostas binárias é mapear o valor de uma função real $f(\cdot)$ em uma probabilidade, usada para amostrar o valor da resposta. Nesse caso, $f(\cdot)$ precisa ser determinística para que seja possível estimar os parâmetros do modelo. Um exemplo desse tipo de modelo é o modelo logístico:

$$\Pr(Y_i = 1 | \mathbf{x}_i) = \frac{1}{1 + \exp(-\mathbf{x}_i^\top \boldsymbol{\beta})}. \quad (3)$$

Modelo τ -Exposure: um novo modelo baseado em limiar de exposição. No modelo *fractional q-neighborhood response* de [Ugander et al. 2013], um indivíduo é considerado “exposto” a um tratamento (A ou B) se ele e uma fração maior que q de seus vizinhos for atribuída am mesmo tratamento. A suposição implícita feita por este modelo é

de que existe um limiar de exposição: uma vez que um nodo v é considerado “exposto”, aumentar a fração de vizinhos atribuídos ao mesmo tratamento não afeta a distribuição da resposta Y_i . Este modelo não pode ser usado para geração de respostas sintéticas, pois não especifica a distribuição das respostas (seja o nodo exposto ou não).

Assim, propomos o modelo de resposta τ -Exposure baseado na suposição da existência de um limiar de exposição $\tau \geq 0.5$. Seja $g_i(\mathbf{Z})$ a fração de vizinhos do nodo i no grupo de tratamento. A resposta de um indivíduo é dada por

$$Y_i = \epsilon_i + \begin{cases} \beta_0 & \text{se } Z_i = 0 \text{ e } g_i(\mathbf{Z}) \leq 1 - \tau, \\ \beta_0 + \beta_2(g_i(\mathbf{Z}) - (1 - \tau)) & \text{se } Z_i = 0 \text{ e } g_i(\mathbf{Z}) > 1 - \tau, \\ \beta_0 + \beta_1 & \text{se } Z_i = 1 \text{ e } g_i(\mathbf{Z}) \geq \tau, \\ \beta_0 + \beta_1 + \beta_2(g_i(\mathbf{Z}) - \tau) & \text{se } Z_i = 1 \text{ e } g_i(\mathbf{Z}) < \tau. \end{cases} \quad (4)$$

Para que o modelo seja realista, é necessário que $\beta_1\beta_2 > 0$, ou seja, que impacto do tratamento do nodo e daquele dos seus vizinhos na resposta esperada tenha o mesmo sinal. Além disso, é necessário que $|\beta_2\tau| \leq |\beta_1|$ para que as curvas de $E[Y_i]$ quando $Z_i = 0$ e $Z_i = 1$ não se cruzem.

Modelo τ -exposure com respostas binárias. As respostas ($Y_i \in \mathbb{R}$) deste modelo podem ser transformadas em respostas binárias de maneira similar ao modelo probit ou ao modelo logístico. Neste artigo, para a geração de respostas binárias, iremos considerar o sinal de Y_i , assim como é feito no modelo probit.

Efeito Médio do Tratamento. Novamente, assumimos que Y_i é uma variável aleatória condicionada em $\mathbf{Z}$. Neste caso, o efeito médio do tratamento (*average treatment effect* ou ATE) é o valor esperado da diferença em médias (*difference in-means*) entre a resposta de uma unidade quando a população está em tratamento (i.e., $\mathbf{Z} = 1$) e a resposta da unidade quando a população está em controle (i.e., $\mathbf{Z} = 0$):

$$\text{ATE} = E \left[\frac{1}{N} \sum_{i=1}^N Y_i(\mathbf{Z} = 1) - \frac{1}{N} \sum_{i=1}^N Y_i(\mathbf{Z} = 0) \right]. \quad (5)$$

Esta equação pode ser especializada para cada um dos modelos apresentados nesta seção (maiores detalhes em [Azevedo et al. 2018, Seção 5]).

Estimadores. Os estimadores para cada um dos modelos apresentados nesta seção são detalhados em [Azevedo et al. 2018, Seção 6].

4. Limites inferiores para o erro de estimadores não-enviesados

Nesta seção derivamos limites inferiores (alguns assintóticos) para o erro de qualquer estimador não-enviesado do ATE, para cada um dos modelos de resposta. Para isto, usamos o *Cramér-Rao Lower Bound* (CRLB), que relaciona o erro médio quadrático (MSE) com a quantidade de informação contida nos dados à respeito dos parâmetros a serem estimados, medida pela matriz de informação de Fisher (FIM). Denotamos por $T_1(\mathbf{X})$, $T_2(\mathbf{X})$, $T_3(\mathbf{X})$ e $T_4(\mathbf{X})$ estimadores não-enviesados do ATE para cada um dos respectivos modelos. As demonstrações podem ser encontradas no nosso relatório técnico [Azevedo et al. 2018].

Teorema 1. *O limite inferior do erro de estimação para o modelo linear é dado por $\text{MSE}(T_1(\mathbf{X})) \geq [0 \ 1 \ 1]\sigma^2(\mathbf{X}^\top \mathbf{X})^{-1}[0 \ 1 \ 1]^\top$. Este limite é válido mesmo que o termo da variância σ^2 seja desconhecido.*

No caso do modelo probit, foi provado em [Demidenko 2001] que a informação contida em uma amostra sobre β , medida pela matriz de informação de Fisher (FIM) é $\mathcal{I}(\mathbf{X}) = \sum_{i=1}^N \frac{\phi^2(s_i)}{\Phi(s_i)(1-\Phi(s_i))} \mathbf{x}_i \mathbf{x}_i^\top$, onde $s_i = \mathbf{x}_i^\top \beta$. Com isso, provaremos o Teorema 2.

Teorema 2. *O limite inferior assintótico do erro de estimação para o modelo probit (2) é dado por $MSE(T_2(\mathbf{X})) \geq (\nabla_{\beta} h)^\top \mathcal{I}^{-1}(\mathbf{X}) \nabla_{\beta} h$, onde*

$$\nabla_{\beta} h = (\phi(\beta^\top \mathbf{1}) - \phi(\beta_0), \phi(\beta^\top \mathbf{1}), \phi(\beta^\top \mathbf{1})) .$$

Segundo [Erhardt 2017], a FIM do modelo logístico é dada por $\mathcal{I}(\mathbf{X}) = \mathbf{X}^\top \mathbf{W} \mathbf{X}$, onde $\mathbf{W} = [W_{ij}]_{N \times N}$ é a matriz diagonal tal que $W_{ii} = \exp(s_i)/(1 + \exp(s_i))^2$. Em [Azevedo et al. 2018], usamos este fato para provar o seguinte teorema.

Teorema 3. *O limite inferior assintótico do erro de estimação para o modelo logit (3) é dado por $MSE(T_3(\mathbf{X})) \geq (\nabla_{\beta} g)^\top \mathcal{I}^{-1}(\mathbf{X}) \nabla_{\beta} g$, onde*

$$\nabla_{\beta} g = \left(\frac{1}{e^{\beta^\top \mathbf{1}} + 1} - \frac{1}{(e^{\beta^\top \mathbf{1}} + 1)^2} - \frac{e^{\beta_0}}{(e^{\beta_0} + 1)^2}, \frac{e^{\beta^\top \mathbf{1}}}{(e^{\beta^\top \mathbf{1}} + 1)^2}, \frac{e^{\beta^\top \mathbf{1}}}{(e^{\beta^\top \mathbf{1}} + 1)^2} \right).$$

Teorema 4. *O limite inferior do erro de estimação para o modelo τ -exposure é dado por $MSE(T_4(\mathbf{X})) \geq [0 \ 1 \ 1] \sigma^2 (\mathbf{X}^\top \mathbf{X})^{-1} [0 \ 1 \ 1]^\top$, onde a matrix $\mathbf{X}$ é definida em [Azevedo et al. 2018].*

5. Metodologia de Avaliação

Nesta seção descrevemos a metodologia que adotamos no estudo empírico do erro resultante ao se assumir, durante a estimação, um modelo de interferência incorreto. Para tanto, geramos respostas com os modelos apresentados anteriormente e estudaremos o valor do Erro Quadrático Médio (MSE) ao estimar com os diferentes estimadores, incluindo o MLE do modelo correto. Os resultados obtidos foram omitidos por razões de espaço, mas podem ser encontrados em [Azevedo et al. 2018].

Datasets. Utilizamos três redes sociais Bitcoin OTC, Enron Email e Wiki-Vote disponíveis na coleção de datasets SNAP, de Stanford¹.

Parâmetros. Cada um dos modelos é parametrizado por um vetor $\beta = (\beta_0, \beta_1, \beta_2)$ cujas coordenadas tem uma interpretação semelhante. O parâmetro β_0 quantifica a propensão intrínseca da população e será fixado em zero. O parâmetro β_1 quantifica a influência do tratamento atribuído ao indivíduo na sua resposta. Finalmente, o parâmetro β_2 quantifica a influência da fração dos vizinhos de um nodo em tratamento na sua resposta. Consideramos os seguintes casos: (i) SUTVA é válida: $\beta = (0, 1, 0)$; (ii) tratamento do nodo é irrelevante: $\beta = (0, 0, 1)$; (iii) tratamento dos vizinhos é menos relevante que do nodo: $\beta = (0, 1, 0.5)$; (iv) tratamento dos vizinhos é tão relevante quanto do nodo: $\beta = (0, 1, 1)$; e (v) tratamento dos vizinhos é mais relevante que do nodo: $\beta = (0, 1, 2)$. Neste trabalho, o parâmetro adicional do modelo τ -exposure será fixado em $\tau = 0.85$.

Vetores de tratamento. Para cada combinação de grafo, vetor β e modelo de resposta, geramos um vetor de tratamento $\mathbf{Z}$ aleatório (binário), em que $P(Z_i = 1) = 0.5, i \in \mathcal{V}$.

¹<http://snap.stanford.edu/>

Vetores de resposta. Para cada vetor de tratamento, geramos 1000 vetores de resposta $\mathbf{Y}$ para um dado modelo.

Estimadores. Para cada par $(\mathbf{Z}, \mathbf{Y})$, calculamos diversas estimativas para o ATE. Usamos o estimador de mínimos quadrados do modelo linear para estimar o ATE nos casos em que a resposta é real (i.e., $Y_i \in \mathbb{R}$). Usamos o MLE dos modelos probit e logístico para estimar o ATE nos casos em que a resposta é binária (i.e., $Y_i \in \{0, 1\}$). Além disso, os estimadores $\widehat{\text{ATE}}_{\text{SUTVA}}$ e $\widehat{\text{ATE}}_{\tau}$ foram aplicados a todos os pares $(\mathbf{Z}, \mathbf{Y})$.

6. Conclusões

Embora já exista um método para aceitar ou rejeitar a SUTVA [Saveski et al. 2017], não existe um método para determinar qual modelo de resposta melhor descreve os dados. Quando a SUTVA é rejeitada, é preciso assumir um modelo para estimar o ATE. Mesmo que o modelo assumido descrevesse perfeitamente a função de resposta dos usuários, existe um erro inerente a flutuações estatísticas, que derivamos analiticamente neste trabalho. Usando como referência este erro inerente, avaliamos os erros obtidos ao se especificar incorretamente o estimador. Observamos que alguns erros de especificação não elevaram muito o MSE (p. ex., assumir Probit, quando o modelo de resposta é Logístico, ou ainda, assumir o τ -exposure com τ elevado, quando o modelo de resposta é Linear). Contudo, o erro depende, em geral, da rede e dos parâmetros do modelo de resposta.

Referências

- Azevedo, F. G., Nogueira, B. D., Murai, F., and da Silva, A. P. C. (2018). Modelos de resposta para experimentos randomizados em redes sociais de larga escala. arXiv:1803.03497.
- Backstrom, L. and Kleinberg, J. M. (2011). Network bucket testing. *WWW*, pages 615–624.
- Demidenko, E. (2001). Computational aspects of probit model. *Mathematical Communications*, 6(2):233–247.
- Erhardt, E. B. (2009 (acessado 22 de janeiro de 2017)). *Logistic Regression and Newton-Raphson*.
- Gui, H., Xu, Y., Bhasin, A., and Han, J. (2015). Network A/B Testing: From Sampling to Estimation. *WWW*, pages 399–409.
- Kohavi, R., Deng, A., Frasca, B., Walker, T., Xu, Y., and Pohlmann, N. (2013). Online controlled experiments at large scale. In *KDD*, pages 1168–1176.
- Kohavi, R., Longbotham, R., Sommerfield, D., and Henne, R. M. (2009). Controlled experiments on the web: survey and practical guide. *Data mining and knowledge discovery*, 18(1):140–181.
- Saveski, M., Pouget-Abadie, J., Saint-Jacques, G., Duan, W., Ghosh, S., Xu, Y., and Airolidi, E. M. (2017). Detecting Network Effects. In *KDD*, pages 1027–1035.
- Ugander, J., Karrer, B., Backstrom, L., and Kleinberg, J. (2013). Graph cluster randomization: Network exposure to multiple universes. In *KDD*, pages 329–337.
- Xu, Y., Chen, N., Fernandez, A., Sinno, O., and Bhasin, A. (2015). From infrastructure to culture: A/B testing challenges in large scale social networks. In *KDD*, pages 2227–2236.

SmartLB: Proposta de um balanceador de carga para redução de tempo de execução de aplicações em ambientes paralelos

Vinicius R. S. dos Santos¹, Edson L. Padoin^{1,2},
Philippe O. A. Navaux², Jean-François Méhaut³

¹Universidade Reg. do Noroeste do Estado do Rio G. do Sul (UNIJUI) - Ijuí - RS - Brasil

{vinicius.ribas,padoin}@unijui.edu.br

²Universidade Federal do Rio Grande do Sul (UFRGS) - Porto Alegre - RS - Brasil

navaux@inf.ufrgs.br

³Universite Grenoble Alpes - Grenoble - França

jean-francois.mehaut@imag.fr

Abstract. *This article presents a new load balancer to reduce the execution time and energy consumption of parallel applications. The balancer algorithm collects system information and uses it to take balancing splits. For implementation, the parallel programming model CHARM++ was used. The preliminary results showed a reduction of the execution time of up to 40.9% and up to 47.7% in energy consumption.*

Resumo. *Este artigo apresenta um novo balanceador de carga para redução do tempo de execução e consumo de energia de aplicações paralelas. O algoritmo do balanceador coleta informações do sistema e as utiliza para tomar decisões de balanceamento. Para implementação foi utilizado o modelo de programação paralela CHARM++. Os resultados preliminares apresentaram redução do tempo de execução de até 40,9% e até 47,7% no consumo de energia.*

1. Introdução

Cada vez mais aplicações científicas são simuladas em ambientes paralelos. Tais simulações computacionais representam atualmente um percentual significativo no total das demandas por processamento nos sistemas de HPC e, somente são possíveis com o emprego de programação paralela. Com paralelismo o processamento das aplicações é dividido em partes e executado em paralelo nas unidades de processamento disponíveis nos atuais de sistemas de HPC.

Para atender às grandes demandas de processamento, diferentes arquiteturas paralelas tem sido projetadas e construídas empregando processadores compostos de múltiplas unidades de processamento. No entanto, a maioria das aplicações paralelas apresentam características, como comportamento dinâmico, que acabam gerando desbalanceamento de cargas, ou excessiva comunicação entre os objetos. Tais características dificultam a eficiente utilização dos sistemas de computação que geralmente possuem unidades de processamento homogêneos.

Nesse sentido, este artigo apresenta a proposta de uma nova estratégia de balanceamento de carga denominado SMARTLB. Nossa proposta almeja a redução do tempo total de execução e do consumo de energia por meio da migração de tarefas considerando a carga computacional média das unidades de processamento também denominados de cores ou núcleos.

O trabalho está organizado da seguinte forma. A Seção 2 discute o estado da arte das tecnologias a serem utilizadas e os trabalhos relacionados. A Seção 3 apresenta a estratégia, a metodologias de implementação e o algoritmo do balanceador de carga proposto. A Seção 4 descreve a metodologia e o ambiente utilizado na implementação e execução dos testes. Resultados alcançados são discutidos na Seção 5, seguidos das Conclusões e Trabalhos Futuros.

2. Trabalhos Relacionados

Muitas pesquisas tem sugerido o emprego de balanceamento de carga para reduzir o tempo de execução e o consumo de energia das plataformas quando executam aplicações paralelas.

O balanceamento de carga é uma técnica utilizada para a distribuição uniforme da carga computacional e da comunicação entre todos os cores ou processadores de uma máquina paralela. Estas estratégias podem ser empregadas quando as tarefas são criadas ou periodicamente durante a execução das aplicações paralelas [Zheng et al. 2010].

Estratégias como GREEDYLB, REFINELB e RANDCENTLB são consideradas centralizadas, pois fazem decisões de balanceamento de carga em um único processador. Elas coletam informações de carga de toda a máquina em um core específico, o qual que executa um processo de decisão sequencial com base nessas informações [Zheng et al. 2010]. Outras estratégias, visando uma melhor escalabilidade quando aplicadas em sistemas de grande escala, adotam abordagens distribuídas onde os processadores distribuídos tomam decisões usando sua visão local do sistema. Nessas estratégias, os processadores trocam dados de balanceamento de carga apenas entre os vizinhos, como forma de descentralizar o processo de decisão [Cybenko 1989].

3. Balanceador de Carga SmartLB

Nossa proposta, busca reduzir o *overhead* de balanceamento a partir da implementação de melhorias nas estratégias utilizadas nos algoritmos GREEDYLB, REFINELB e AVERAGELB. Utilizando uma abordagem centralizada almeja-se alcançar balanceamento de carga levando em consideração a média aritmética das cargas de cada unidade de processamento, reduzindo o número total de migrações, o tempo de execução e consequentemente o consumo de energia.

3.1. Metodologias de Implementação

Para implementar a estratégia de balanceamento de carga proposta foi utilizado o ambientes de programação paralela CHARM++. A escolha por este ambiente foi motivada pela sua madura estrutura de balanceamento de carga, a qual permite tanto a criação de novos balanceadores de carga, quanto a utilização dos balanceadores de carga disponibilizados pelo ambiente para comparações de resultados.

Dentre as informações armazenadas pelo SMARTLB destacam-se o número total de tarefas, a carga de cada tarefa, o número total de núcleos e a carga total de cada núcleo. Estes dados são utilizados pela estratégia na tomada de decisões para definir quais tarefas devem ser migradas e qual o novo mapeamento de cada tarefa para que seja mitigado o desequilíbrio de carga.

3.2. Algoritmo

O SMARTLB adota um *threshold* para definir um limite de desbalanceamento de carga aceitável. Caso o desbalanceamento seja maior que o *threshold* previamente definido, o algoritmo busca atingir o balanceamento levando em consideração a diferença de carga entre os

cores com carga acima da média e abaixo da média. Quando o balanceador é chamado, ele usa informações disponibilizadas pelo CHARM++ e calcula as cargas computacionais de cada core.

Quando a estratégia é aplicada, o algoritmo (*linhas 1 a 3*) busca informações sobre a quantidade de objetos mapeados em cada core e suas cargas para calcular o core com maior carga (PM), o core com menor carga (Pm) e a média das cargas (avg).

Tendo computado estes valores, a estratégia (*linha 4*) compara o desbalanceamento entre a carga do core mais carregado e o core menos carregado com o *threshold*. Caso essa razão for menor, o desbalanceamento de carga presente é menor do que o limite aceitável, assim nenhuma tarefa é migrada. Por outro lado, se a razão for maior que o *threshold* definido, o balanceador busca tarefas (*linha 6*) dos cores com cargas acima da média (*linha 8*) e realiza a migração desta tarefa para o core menos carregado (*linha 13*).

A estratégia (*linha 11*) também verifica antes de migrar se esta migração não deixará o core que recebe a tarefa com carga acima da média, para isso testa se a carga da tarefa é menor ou igual a diferença entre o core subcarregado e core atual.

Algoritmo 1: Implementação do SMARTLB

```

1   $PM = \text{getCargaMaior}();$ 
2   $Pm = \text{getCargaMenor}();$ 
3   $avg = \text{getMediaCores}();$ 
4  if  $((Pm/PM) > \text{Threshold})$  {
5      for  $(i = 1; i \leq nTarefas; i++)$  {
6           $core = \text{getCoreTarefa}(i);$ 
7           $cc = \text{getCargaCore}(core);$ 
8          if  $(cc \geq avg)$  {
9               $ct = \text{getCargaTarefa}(i);$ 
10
11              if  $(ct \leq (Pm - cc))$  {
12                   $numMigracoes++;$ 
13                   $migrarProcesso(i, core, Pm);$ 
14                   $Pm = \text{getCargaMenor}();$ 
15              }
16          }
17      }
18 }
```

4. Metodologia

Nossos experimentos foram realizados em uma plataforma Altix UV 2000 projetada pela SGI. A plataforma é composta por 24 nós NUMA. Cada nó tem um processador Intel Xeon E5-4640 modelo Sandy Bridge-EP x86-64 com 8 núcleos físicos de 2,40 GHz.

Cada núcleo do Intel Xeon E5-4640 tem memórias caches L1 de 32 KB para instrução e 32 KB para dados e 256 KB de cache L2. Todos os 8 núcleos compartilham uma memória cache de 20 MB L3. Cada nó possui 32 GB de memória DDR3, que é compartilhada com outros nós de uma forma cc-NUMA através do NUMALink6 da SGI. Em geral, esta plataforma tem 192 núcleos físicos e 768 GB de memória DDR3, dos quais foram utilizados somente 24, um core de cada processador.

A plataforma executa um sistema operacional SUSE Linux Enterprise com o kernel 3.0.101 – 0.29. Todos os benchmarks e balanceadores de carga foram compilados com GCC 4.8.2. A versão do CHARM++ utilizada para implementação foi a 6.5.1.

Para analisar o desempenho, a demanda de potência, o consumo de energia foi utilizada a ferramenta EMonDaemon [Padoin et al. 2014]. Esta ferramenta possibilita a coleta e análise do tempo de execução e da demanda de potência de cada processador durante a execução dos testes. Para realização dos testes a ferramenta EMonDaemon foi configurada para realizar medições de demanda de potência do processador a cada 1s.

Cada um dos testes realizados neste trabalho foram repetidos 10 vezes, para atingirmos um erro relativo menor que 5% e 95% de confiança estatística para uma Student's t-distribution. Entre cada um dos testes foi deixado o sistema em *idle* por no mínimo 20 segundos, de modo que a demanda de potência do sistema se estabilizasse.

4.1. Benchmarks

Para avaliar o desempenho, demanda de potência e o consumo de energia do balanceador de carga proposto foram selecionados três benchmarks. Eles foram escolhidos devido à sua variada gama de padrões de comunicação e características da carga de trabalho. A descrição de cada benchmark é apresentada na sequência :

- ***lb_test***: benchmark que apresenta desequilíbrio de carga onde pode-se usar diferentes padrões de comunicação. Neste artigo, foi utilizado um grafo de comunicação aleatória.
- ***kNeighbor***: benchmark iterativo onde cada tarefa se comunica com "k" outras tarefas em cada etapa ou iteração. Seu padrão de comunicação é um anel.
- ***ComprehensiveBench***: benchmark sintético configurável usado para criar aplicativos iterativos com diferentes parâmetros, como o número de tarefas, iterações, gráfico de comunicação, carga de tarefas e tamanhos de mensagens.

A Tabela 1 apresenta as características dos benchmarks e parâmetros utilizados em nossos experimentos. Diferentes frequências de balanceamento de carga foram escolhidas para diferentes aplicações, a fim de estabelecer um equilíbrio entre os benefícios das tarefas de remapeamento e os custos de mover tarefas entre cores bem como a computação de um novo mapeamento de tarefas. Nos testes com o balanceador de carga proposto foi adotado um *threshold* de valor igual a 5%.

Tabela 1. Parâmetros de entrada dos Benchmarks.

Benchmarks	Tarefas	Iterações	Frequência de LB
<i>lb_test</i>	200	150	10
<i>kNeighbor</i>	1600	50	10
<i>ComprehensiveBench</i>	85	50	5

4.2. Balanceadores de carga

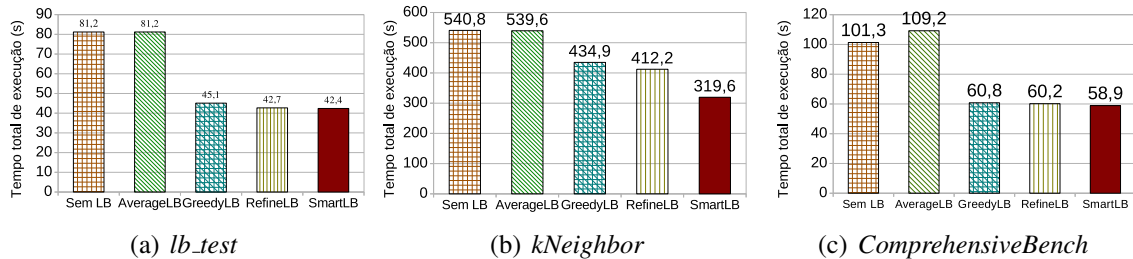
Os resultados de tempo de execução e a economia de energia alcançados com o BC proposto foram comparados com os balanceadores de carga GREEDYLB, REFINELB e AVERAGELB.

5. Resultados

Nesta seção são apresentados as avaliações de desempenho, as redução do tempo de execução, do consumo de energia alcançadas com o emprego do nosso balanceador de carga SMARTLB na plataforma experimental apresentada na última seção.

Na Figura 1 são apresentados os tempos de execução dos testes realizados com o benchmark *lb_test* e *kNeighbor* para diferentes quantidades de tarefas.

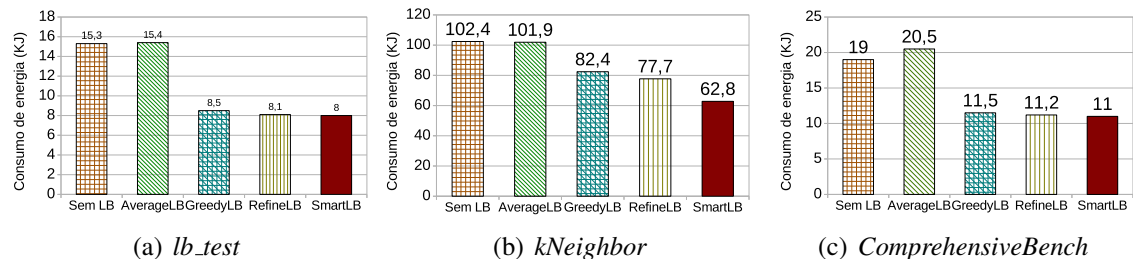
Figura 1. Tempos de execução mensurados durante a execução



O BC SMARTLB apresentou melhor desempenho para ambos os benchmarks testados. Para *lb_test* com 200 tarefas o SMARTLB conseguiu reduzir o tempo de 81,2 para 42,4 segundos, o que representa uma redução de 47,8% em relação à execução sem balanceador.

Quando foi executado o benchmark *kNeighbor*, o SMARTLB conseguiu uma redução significativa no tempo de execução. Com 1600 tarefas o tempo total de execução foi reduzido de 540,8 para 319,6 segundos, o que representa uma redução de 40,9% em relação à execução sem balanceador. Para este benchmark, o SMARTLB apresentou tempos de 22,4% menor que o balanceador REFINELB e de 26,5% menor que o balanceador GREEDYLB.

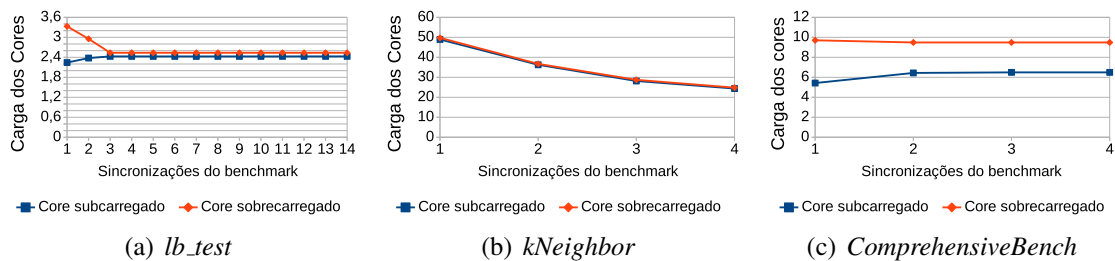
Figura 2. Consumo de energia mensurado durante a execução dos benchmarks



Uma redução equivalente e significativa foi também observada no consumo de energia quando aplicado o SMARTLB com o benchmark *kNeighbor*. O consumo de energia foi reduzindo em 38,6% em relação à execução sem balanceador, ou seja uma economia de 39,6KJ (de 102,4 KJ para 62,8 KJ).

O tempo de execução e o consumo total de energia das execuções também foi reduzido quando o SMARTLB foi empregado na execução do benchmark *ComprehensiveBench*. O tempo total de execução apresentou uma redução de 41,8% em relação à execução sem balanceador. Para este benchmark, o uso da estratégia apresentou desempenho em média de 3,1% melhor que os alcançado pelos balanceador GREEDYLB e REFINELB. Ganhos equivalentes foram observados no consumo de energia quando aplicado a nossa proposta.

Na Figura 3 são apresentados os desbalanceamentos de carga mensurados a cada chamada do balanceador de carga proposto. Em todos os testes, o SMARTLB conseguiu concluir a execução dos benchmarks apresentando o menor de desbalanceamento.

Figura 3. Desbalanceamento mensurado durante a execução dos benchmarks

Nos testes com o *lb_test*, na primeira chamada do balanceador a diferença entre o core mais carregado e o core menos carregado era de 3,34 unidades. O SMARTLB conseguiu reduzir, até a última sincronização, essas diferenças para 0,12. Nos testes realizados com os benchmarks *kNeighbor* e *ComprehensiveBench*, o desbalanceamentos iniciais eram menores, de 0,10 e 0,28 unidades. Mesmo assim, com a utilização do SMARTLB, os desbalanceamentos foram reduzidos, finalizando a execução com 0,07 e 0,11 respectivamente.

6. Conclusões e trabalhos futuros

Este trabalho apresentou a proposta de um novo balanceador de carga denominado SMARTLB. Nossos resultados demonstram que o uso do balanceador de carga SMARTLB reduziu o tempo de execução e a quantidade de energia gasta quando aplicado em aplicações iterativas. O tempo total de execução e o consumo de energia foram reduzido em média 43,5% e 42,8% nos testes executados com três benchmarks. Os resultados alcançados pela nossa estratégia proposta foram melhores que os alcançados pelos balanceadores GREEDYLB e REFINELB.

Como futuros trabalhos, pretende-se realizar melhorias no algoritmo de tomada de decisão do SMARTLB de modo a melhorar o controle de migrações de tarefas. Pretende-se também realizar testes em sistemas paralelos maiores utilizando problemas reais de computação científica bem como comparar com outros balanceadores de carga do estado da arte.

Agradecimentos

Trabalho apoiado por CNPq e CAPES com recursos do programa EU H2020 e do MCTI/RNP-Brasil sob o projeto HPC4E de número 689772 e do edital de bolsa PIBIC da UNIJUI.

Referências

- Cybenko, G. (1989). Dynamic load balancing for distributed memory multiprocessors. *Journal of parallel and distributed computing*, 7(2):279–301.
- Padoin, E. L., Pilla, L. L., Castro, M., Boito, F. Z., Navaux, P. O. A., and Mehaut, J.-F. (2014). Performance/energy trade-off in scientific computing: The case of ARM big.LITTLE and Intel Sandy Bridge. *IET Computers & Digital Techniques*, 2(3):1–14.
- Pilla, L. L., Bozzetti, T. C., Castro, M., Navaux, P. O. A., and Méhaut, J.-F. (2015). Comprehensivebench: a benchmark for the extensive evaluation of global scheduling algorithms. *Journal of Physics: Conference Series*, 649(1):012007.
- Zheng, G., Meneses, E., Bhatele, A., and Kale, L. V. (2010). Hierarchical load balancing for charm++ applications on large supercomputers. In *Parallel Processing Workshops (ICPPW), 2010 39th International Conference on*, pages 436–444. IEEE.

Construção de um Modelo de Simulação para Escalonamento de Processos não-preemptivos

Jhonatan T. Cabral Néry¹, Franciny M. Barreto¹, Joslaine C. Jeske de Freitas¹

¹Bacharelado em Ciência da Computação –
Universidade Federal de Goiás - Regional Jataí (UFG - Jataí)
BR 364, km 195, nº 3800 CEP 75801-615 – Jataí – GO – Brazil

{jhonatanhallisson, francinymedeiros, joslaine}@gmail.com

Abstract. *Modern society has increasingly demanded technology requirements, some of them agility, availability and the ability to handle different tasks simultaneously. Thus, one of the researches to meet the current scenario has its focus on the treatment of running applications (so-called processes) that may be in a functioning Operating System. On a computer, the element responsible for handling these processes is the scheduler. This research aims to construct a simulator that specifically shows the behavior of the First-in First-time scheduler in scenarios with one, two and three processors with random duration processes.*

Resumo. *A sociedade moderna tem exigido cada vez mais requisitos da tecnologia, sendo alguns deles agilidade, disponibilidade e a possibilidade de tratar diferentes tarefas simultaneamente. Assim, uma das pesquisas para atender o atual cenário tem seu foco no tratamento das aplicações em execução (os chamados processos) que podem estar em um Sistema Operacional em funcionamento. Em um computador, o elemento responsável por tratar estes processos é o escalonador. Esta pesquisa tem por objetivo construir um simulador que mostre especificamente o comportamento do escalonador First-in First-out em cenários com um, dois e três processadores com processos de duração aleatória.*

1. Introdução

Todo computador possui recursos finitos, sendo isto verdade, todos estão sujeitos a atrasos e escassez de modo geral. Por este motivo, é importante que se utilizem políticas de alocação bem estruturadas e que não se deixe evidente a carência de recursos. O gerenciamento destes computadores atuais é feito pelo Sistema Operacional (SO) e este realiza tarefas complexas e uma delas é o escalonamento de processos, que de acordo com [Maziero 2014] é o mecanismo que aloca recursos de processamento aos programas da máquina, ditando quem usará, em que momento e por quanto tempo.

Atualmente existem várias técnicas de escalonamento diferentes, cada uma delas propondo uma abordagem distinta. Tendo isto em vista, manifesta-se uma pergunta: Quais os possíveis comportamentos de um escalonador e qual sua conduta quando o número de processadores é alterado? Então, este artigo tem como objetivo responder a esta indagação, mostrando através de simulação, como um escalonador First-Come First-Served (FCFS), também chamado de First-In First-Out (FIFO) pode se comportar quando aplicado a cenários distintos e como isto pode contribuir para a otimização do uso do processador.

2. Conceitos

2.1. Sistemas Operacionais

[Deitel et al. 2005, Silberschatz 2009] definem SO como sendo um *software* que possibilita a interação entre as aplicações e o *hardware* de um computador, além disto fornece serviços que permitem que cada aplicação seja executada com segurança e efetividade. [Machado and Maia 2013] ainda reiteram dizendo que aplicações projetadas sobre um SO são mais racionais, pois detêm o foco em sua tarefa central (alto nível de abstração), não precisam se comunicar diretamente com o hardware, o que as torna quase sempre portáteis, além de exigirem menor custo nas etapas de criação e manutenção.

2.2. Redes de Petri

As redes de Petri (ou RdP) surgiram em 1962, por iniciativa de Carl Adam Petri. Redes de Petri são consideradas como uma ferramenta gráfica e matemática de representação formal que permite modelagem, análise e controle de sistemas a eventos discretos que comportam atividades paralelas, concorrentes e assíncronas [Murata 1989]. Os elementos básicos que permitem a definição das redes de Petri são:

- **Lugar:** Representado por um círculo. Um lugar pode ser interpretado como uma condição, um estado parcial, um procedimento, a existência de um recurso, etc..
- **Transição:** Representada graficamente por uma barra ou retângulo. As transições são associadas a eventos que ocorrem em um sistema.
- **Ficha:** É representada por um ponto em um lugar. É um indicador significando que a condição associada ao lugar é verificada, como, por exemplo, um recurso disponível em um certo processo.

As redes de Petri Coloridas foram idealizadas por Kurt Jensen, pesquisador da Universidade de Aarhus, na Dinamarca. Seu artigo, datado do ano de 1981, intitulado *Coloured Petri Nets and the Invariant Method* foi um trabalho promissor, que induziu uma série de estudos sobre coloração das fichas. As redes de Petri coloridas foram projetadas para tornar mais compactos os modelos de sistemas complexos, permitindo a individualização de fichas (usando as cores que lhe são atribuídas). Assim, diferentes processos ou recursos podem ser representados em uma mesma estrutura gráfica.

CPN Tools é uma ferramenta desenvolvida pelo grupo CPN da Universidade de Aarhus (2000-2010), capaz de trabalhar com vários tipos de redes de Petri, como as redes Temporizadas, redes Clássicas e as redes de Petri Coloridas, possibilitando a criação, simulação de modelos e extração de relatórios e probabilidades que determinam se estes modelos são viáveis, ou se são possíveis [Jensen and Kristensen 2009].

3. Implementação

3.1. Definição do Modelo

[Maziero 2014] define o escalonador de processos como sendo o nível mais baixo, complexo e o processo com maior prioridade em um SO. [Tanenbaum 2003] cita diferentes tipos de SO multiprogramados possíveis. Dentre estes, a presente pesquisa buscará mostrar o escalonador FIFO, que se encaixa no grupo dos escalonadores não preemptivos.

Para criar um modelo de simulação será necessário encontrar um cenário de escalonamento base, com o objetivo de retirar alguns dados fundamentais, como tempo de chegada de processos e tempo de execução de cada um deles.

[Stallings 2012] descreve um cenário de escalonamento de cinco processos que, como ilustrado na tabela 1, é composto pelos dados de hora de chegada (tempo de chegada de cada processo) e tempo de serviço (tempo de execução), que são exatamente os dados necessários para dar andamento as simulações. [Stallings 2012] especifica que neste cenário os processos exigem uso alternativo de processador e E/S de forma repetitiva.

Processo	Hora de Chegada	Tempo de Serviço
A	0	3
B	2	6
C	4	4
D	6	5
E	8	2

Table 1. Exemplo de escalonamento de processo,[Stallings 2012].

Ainda usando este cenário, é possível identificar o tempo médio de chegada e o tempo médio de execução dos processos. Usando a tabela 1 como base, tem-se o seguinte:

- Média chegada = $(0+2+4+6+8)/5 = 4$
- Média execução = $(3+6+4+5+2)/5 = 4$

[Maziero 2014] descreve o algoritmo de escalonamento FIFO apresentado a seguir.

```

01 ALGORITMO FIFO
02   t = 0
03   ENQUANTO (t < tmax), FAÇA
04     PARA cada processo i
05       SE (o processo i chegou agora), ENTÃO
06         coloque o processo i no fim da fila de prontos
07       fim SE
08     fim PARA
09     SE ((o processador estiver livre) E
10       (houver processo na fila de prontos)),
11       ENTÃO
12         encontre o primeiro processo da fila de prontos
13         migre o processo para o estado "executando"
14       fim SE
15     SE (o processo executando chegou ao fim da execução),
16       ENTÃO
17       libere o processador
18     SENÃO
19       SE (o processo solicitou E/S),
20         ENTÃO
21         migre o processo para o estado "bloqueado"
22         libere o processador
23       fim SE
24     fim SE
25     t <-- t+1      /*incrementando o tempo*/
26   fim ENQUANTO
27 fim ALGORITMO

```

A Figura 1 mostra a rede de Petri gerada baseada no funcionamento do escalonador FIFO.

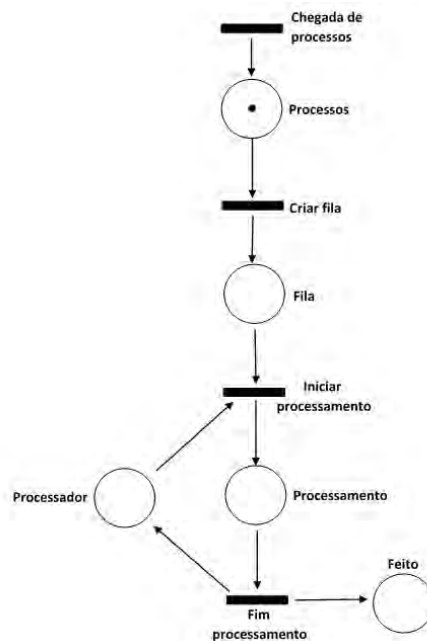


Figure 1. Rede de Petri do escalonador FIFO.

Baseado no cenário proposto por [Stallings 2012] e no algoritmo apresentado por [Maziero 2014] este trabalho propõe construir o algoritmo de escalonamento FIFO utilizando CPN Tools e simulá-lo em diferentes cenários. Esta proposta difere-se do apresentado por [Stallings 2012] pois deseja-se mostrar uma situação mais próxima da realidade. Para tanto, funções exponenciais serão utilizadas para gerar valores aleatórios de tempo de chegada e tempo de serviço de cada processo.

3.2. Desenvolvimento

A figura 2 mostra o modelo de simulação completo do escalonador FIFO, criado na ferramenta *CPN Tools*. As funções utilizadas e definidas como **iat** e **Mtime** são funções prontas da ferramenta e segundo [CPNTools 2018] são definidas pelo seguinte:

- **fun iat(ET) = round (exponential(1.0/ET))** : Cria valores aleatórios levando em conta uma média (ET), neste caso 4.
- **fun Mtime() = IntInf.toInt(time())** : Retorna o tempo atual no modelo como um número inteiro.

Para a simulação do funcionamento do escalonador FIFO com 2 e 3 processadores o lugar *Processador* recebeu o valor 2() e 3() respectivamente, demonstrando que o mesmo trabalhará com mais de um processo por vez.

O *CPN Tools* possui uma ferramenta eficiente para a observação do comportamento de modelos simulados, os chamados monitores. Foram criados neste trabalho os monitores **Tamanho da Fila**, **Tempo de Espera** e **Tempo de Processamento** para observar o comportamento do escalonador FIFO.

4. Análise dos resultados

À partir das simulações geradas pela ferramenta *CPN Tools* é possível identificar vários aspectos da execução e suas nuances quando alterado o número de processadores.

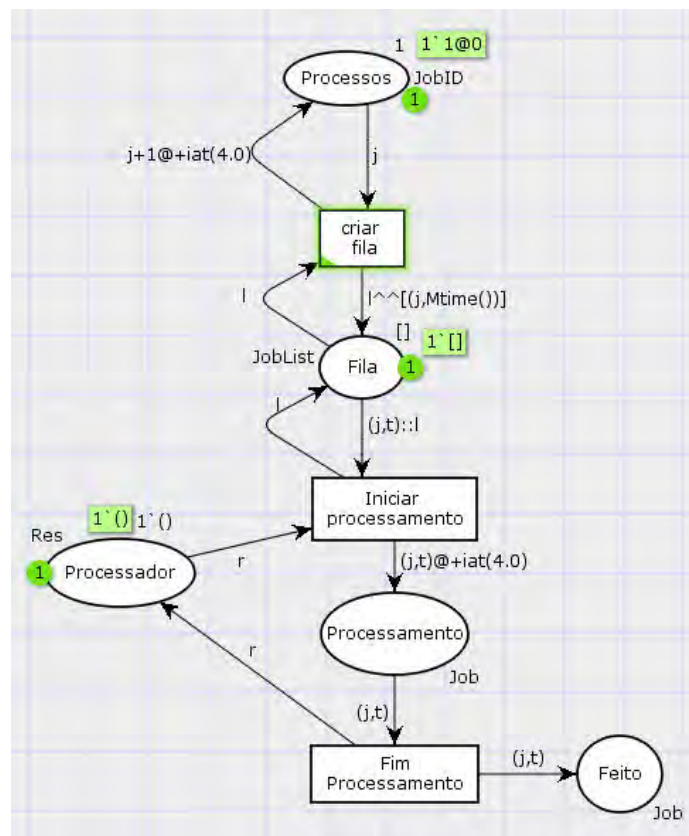


Figure 2. Modelo de simulação completo do escalonador FIFO.

Com o monitor **Tamanho da Fila** foi possível observar que a média de elementos na fila diminui substancialmente com o aumento do número de processadores.

O monitor **Tempo de Espera** mostra que no modelo com 1 processador esta espera penaliza as aplicações, sendo em média 178,585786 unidades de tempo o que cada processo aguarda na fila. O modelo com 2 processadores já mostra uma redução deste tempo em 99,24% e com 3 processadores a redução é de 99,90% em relação ao modelo com 1 processador.

Por fim o monitor **Tempo de Processamento** que visa expressar o tempo total que cada processo gastou no escalonador (tempo de espera + tempo de execução), consegue abstrair conhecimentos bastante pertinentes, já que o tempo total de processamento é fator determinante para avaliar uma arquitetura de escalonamento. Nestas estatísticas é possível ver que quanto menos processadores, mais sobrecarregado o sistema fica, gerando grandes filas e com isto uma espera proporcional, o que faz com que o tempo total de processamento seja maior.

O monitor em questão mostra os seguintes valores: O tempo médio total de processamento com 1 processador é de 182,566413 unidades de tempo, com 2 esta média fica em 5,33624 unidades e se colocando 3 processadores para trabalhar em paralelo este número decresce para 4,150723 unidades de tempo. Sabendo estas informações é possível alegar que o cenário menos atrativo é o de 1 processador, o de 2 e 3 processadores quase se equiparam em eficiência, à vista que o modelo de 2 processadores é 97,08% mais

rápido que o modelo com 1 e o modelo de 3 processadores é pouco diferente, ficando com 97,73% de rapidez a mais que o de 1 processador apenas.

5. Considerações Finais

Sabendo da diferença próxima de desempenho do escalonador FIFO com 2 e 3 processadores é possível concluir que com o aumento da quantidade de processadores, esta diferença de performance tenderá a diminuir e a ser cada vez menos significativa dentro de um escalonador FIFO. Com isto indivíduos e instituições podem usar estes dados para comparação com outros escalonadores e também para decidir a respeito do melhor custo/benefício ante a questão quantidade de processadores *versus* ganho de desempenho.

Sobre a ferramenta *CPN Tools*, pode-se afirmar que esta se mostrou maleável e prática, conseguindo implementar elementos teoricamente complexos de forma simples e isto foi decisivo para a realização deste trabalho. Quanto a visão do poder computacional das simulações para explicar e auxiliar na análise de situações e arquiteturas, é correto dizer que este método é eficaz quando se consegue gerar um modelo assertivo. Assim sendo, as simulações conseguem fornecer informações precisas e relevantes sobre situações hipotéticas ou reais, dando plenas condições para a tomada de decisão.

6. Trabalhos futuros

Algumas sugestões de trabalhos futuros vem de uma das vertentes que não foi abordada aqui, que é a comparação entre o escalonador FIFO e outros escalonadores. Onde a ferramenta *CPN Tools* poderia facilmente gerar relatórios e provocar discussões construtivas a respeito do melhor escalonador para determinados cenários. Trabalhos posteriores podem também explorar simulações com um número maior de processadores, podendo talvez levar a novas descobertas sobre escalonamentos. Pode-se também usar a metodologia exposta aqui para criar novas abordagens que tratem da eficiência do uso de processadores e da relação custo x benefício que estas novas abordagens podem oferecer.

References

- CPNTools (2018). Documentation - <http://cpntools.org/2018/01/16/documentation-2/>.
- Deitel, H., Deitel, P., and Choffnes, D. (2005). *Sistemas Operacionais*. PRENTICE HALL BRASIL.
- Jensen, K. and Kristensen, L. M. (2009). *Coloured Petri Nets*. Springer, Germany.
- Machado, F. B. and Maia, L. P. (2013.). *Arquitetura de sistemas operacionais*. LTC,, Rio de Janeiro-RJ., 5.ed. edition.
- Maziero, C. A. (2014). Sistemas operacionais: Conceitos e mecanismos. *Livro aberto*.
- Murata, T. (1989). Petri nets: Properties, analysis and applications. *Proceedings of the IEEE*, 77(4):541–580.
- Silberschatz, A. (2009). *Operating System Concepts*. John Wiley & Sons Software, 8th edition.
- Stallings, W. (2012). *Operating Systems: Internals and Design Principles*. Prentice Hall.
- Tanenbaum, A. (2003). *Sistemas Operacionais Modernos*. Prentice-Hall do Brasil.

Comparing Parallel Algorithms for Van der Waals Energy with Cell-List Technique for Protein Structure Prediction *

Daniel R. F. Bonetti^{1†}, Gesiel Rios Lopes^{1†}, Alexandre C. B. Delbem^{1†},
Paulo S. L. Souza^{1†}, Kalinka C. Branco^{1†}, Gonzalo Travieso^{1‡}

¹ University of São Paulo (USP – ICMC[†] – IFSC [‡])
São Carlos-SP – Brazil

daniel.bonetti@gmail.com, gesielrios@usp.br

{acbd, pssouza, kalinka}@icmc.usp.br, gonzalo@ifsc.usp.br

Abstract. *This paper compares the runtime of three distinct parallel algorithms for the evaluation of an ab initio and full-atom approach based on GA and cell-list technique, in order to minimize the van der Waals energy. The three parallel algorithms are developed in C and use one of these programming models: MPI, OpenMP or hybrid (MPI+OpenMP). Our preliminary results show that van der Waals Energy are executed faster and with better speedups when using hybrid and more flexible parallel algorithms to predict the structure of larger proteins. We also show that for small proteins the communication of MPI imposes a high overhead for the parallel execution and, thus the OpenMP presents a better relation cost x benefit in such cases.*

1. Introduction

The protein structure prediction (PSP) from its amino acid sequence is a complicated and expensive task, since that, according to Levinthal's paradox [Levinthal 1968], there are a vast number of conformations possible to reach the correct native state, which take an long time to evaluate, whereas in real proteins take only a few seconds or less to reach their native state. In addition, Anfinsen's thermodynamic hypothesis [Anfinsen 1972] states that, at least for small globular proteins, the native structure is a unique, stable, and kinetically accessible minimum of the free energy. Levinthal's paradox and Anfinsen's hypothesis allow us to formulate *ab initio* PSP as an optimization problem. In this context, Genetic Algorithms (GAs) have produced relevant results [Lima 2006, Dorn et al. 2011].

In the recent literature, there are different parallel solutions available aiming to minimize the execution time of such algorithms [Benítez and Lopes 2010, Bonetti et al. 2010, Bonetti et al. 2013]. Parallel algorithms in this context usually have distinct and limited performance, mainly because they are specific for one programming model and/or computer architecture. In [Bonetti et al. 2010] and [Bonetti et al. 2013], instead by just making parallel the van der Waals energy from its $O(n^2)$ algorithm, the authors first improved the efficiency of the energy using the cell-list algorithm, enabling the complexity reduction to $O(n)$.

This paper compares the performance (runtime) of three specific parallel algorithms for the evaluation of an *ab initio* and full-atom approach based on GA and cell-list technique, to minimize the van der Waals energy. The three parallel algorithms are

*This research project is sponsored by FAPESP (2013/07375-0).

developed in C and use one of these programming models: MPI, OpenMP or hybrid (MPI+OpenMP). We show, in our experiments, the importance to develop adaptive algorithms to explore the benefits of different molecules, geometry, architectures and programming paradigms. Indeed, our preliminary results show that van der Waals Energy evaluation is faster when using hybrid parallel algorithms for larger proteins, even when they use as basis algorithms specific for determined programming paradigm (as message passing). A hybrid version is capable to minimize negative aspects of such specific programming paradigms. For small proteins, for example, the MPI communication imposes a high overhead with communication and, thus, in these cases, OpenMP presents better results. On the other hand, OpenMP does not maintain its performance for larger structures. Hybrid algorithms can act in this context, mitigating such problems.

The remaining of this paper is structured as follows. Section 2 introduces concepts related to the van der Waals calculation. Section 3 shows the method of cell-list. The configuration of the experiments and their results are presented in Section 4. Finally, Section 5 concludes this paper.

2. Van der Waals Energy

Van der Waals energy frequently describes the energy of a molecule. The Lennard-Jones potential (also known as Lennard-Jones 12-6) allows to calculate the van der Waals energy of a molecule [Jones 1924]. The van der Waals energy varies according to the distance of the pair of atoms and the type of atoms (hydrogen, carbon, nitrogen, oxygen, etc.), as shown in Equation 1, where r_{ij} is the relative distance.

$$r_{ij} = \frac{d_{i,j}}{R_i + R_j}. \quad (1)$$

The Lennard-Jones potential used in Protein Structure Prediction (PSP) is shown in Equation 2:

$$f_{LJ}(r_{ij}) = \begin{cases} Ar_{ij}^{-12} - Br_{ij}^{-6} & \text{if } r_{ij} > 0.8, \\ C & \text{if } r_{ij} \leq 0.8, \end{cases} \quad (2)$$

where A and B are constants experimentally determined based on characteristics of the environment, and C is given by $Ar_{ij}^{-12} - Br_{ij}^{-6}$ with $r_{ij} = 0.8$.

A molecule's van der Waals energy can be obtained by summing of the interaction of all pairs of atoms. It results in $\frac{n^2-n}{2}$ interactions, where n is the number of atoms of the molecule, showed in Equation 3:

$$E_{vdw} = \sum_{i=1}^{n-1} \sum_{j=i+1}^n f_{LJ}(r_{ij}). \quad (3)$$

3. Method of Cell-Lists

Cell-list is a general technique to improve the efficiency of algorithms responsible to calculate pairs of particles separated by a cutoff [Allen and Tildesley 1987]. It creates cells with least the cutoff length enabling the interaction of only atoms inside the cell

and neighboring cells. In this study, the cell-list technique is adapted to the van der Waals evaluation, in which the particles are represented by the atoms in the molecule configuration. We will use Cell-list Algorithm (CA) to describe the sequential cell-list algorithm developed, as well as Cell-list Parallel Algorithm (CPA) to describe the parallel version. CPA is divided into CPA with OpenMP, CPA with MPI and hybrid CPA.

4. Results and Discussion

We used the cluster belonging to LCR¹ to evaluate the runtime of the proposed approaches. The cluster has 14 nodes and is divided into two groups according to its characteristics. The first group has 10 nodes with AMD Dual-Core 64 bits 2.8 GHz processors and 4 GB of RAM. The second group has 4 nodes with Intel Core i7 64 bits 2.67 GHz processors and 12 GB of RAM. The operation system is GNU/Linux Ubuntu with kernel 2.6.26-2. All nodes have two network adapters: one for the file system and another for messages in MPI, both connected to two independent 3Com Gigabit Ethernet switches.

The inputs of the algorithms are based on proteins that differ in structure and size. Eight different sizes of proteins were chosen from PDB. Protein 1A11 was selected to be the lower bound, with only 390 atoms. Protein 1HTO represents the upper bound of experiments, with 147,900 atoms. Other proteins were selected to cover the range of proteins to evaluate the scalability of the proposed techniques. The accuracy reported from our algorithms remained unchanged for all executions, and for space reason, it will not be shown in this paper.

The methods were statistically compared using the p -value of the Welch Two Sample t-test with 95% confidence interval. The comparisons made were: CA with Quadratic Algorithm (QA) in Equation 3; CPA OpenMP with 8 and 16 processors with CA; CPA MPI, processors ranging from 2 to 18 with CA; CPA hybrid, processors ranging from 8 to 32 with CA; CPA hybrid, processors ranging from 8 to 32 with CPA MPI; and finally, CPA hybrid, processors ranging from 8 to 16 with CPA OpenMP. All tests were performed for all 8 proteins, rendering in 176 tests. The highest p -value obtained was 0.002 and occurred between techniques CPA hybrid with CPA OpenMP, both with 8 processors.

4.1. Speedup of Cell-list Algorithm

Figure 1(a) shows the speedup achieved using the proposed CA in comparison to QA. Points represent the experimental data, and the line represents the predicted linear model. Indeed, the CA reduced the complexity from $O(n^2)$ to $O(n)$. The speedup line predicted linear increases according to the size of the protein. Even for small proteins, the speedup is significant. For protein 1AI0 with 4,728 atoms, the speedup is 5. Larger proteins did produce speedups more impressive, as 1HTO, which resulted in a speedup of 127. The improvement relies on the size of the cell grid (which also depends on the number of atoms). The larger the number of atoms, the larger will be the cell grid.

4.2. Parallel cell-list with OpenMP

Although the simplest parallel implementation of the van der Waals uses OpenMP, it can produce good results. Figure 1(b) shows the speedup achieved for CPA using OpenMP

¹Reconfigurable Computer Laboratory, group of research of Embedded and Evolutionary and Systems at Institute of Mathematics and Computer Sciences at São Paulo University, Brazil.

about CA. The experiment was performed in a node containing an i7 processor with 4 physical cores. The speedup is close to 4, indicating that the tasks were properly distributed among processors and the computational time of the van der Waals calculation was proportionally reduced by the number of cores.

The high number of atoms inside a single cell could be a small disadvantage of cell-list since it will have to compute more interactions, always performed in QA. Points four and five of Figure 1(b) show a depression in the speedup since these proteins are more globular than the others used. However, this speedup is still good.

4.3. Parallel cell-list with MPI

Figure 1(c) shows the speedup achieved when the van der Waals energy was computed using the CPA with MPI. The experiments were performed in 9 nodes of AMD processors. For small proteins, such as 1BFI (1,753 atoms), the speedup was not significant, due to inter-process communication. The cell-list procedure is so fast that the communication time strongly influences the total computational time, and the parallelization for small proteins is not viable. On the other hand, for proteins above 4,728 atoms, the speedup is significant for a small number of processors.

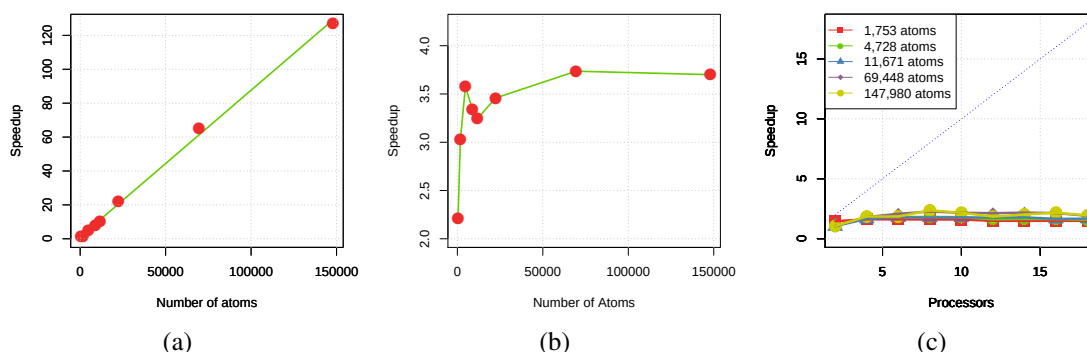


Figure 1. (a) Speedup achieved with CA in relation to QA. (b) Speedup achieved with CPA with OpenMP in relation to CA. (c) Speedup achieved for CPA with MPI in relation to CA.

4.4. Parallel cell-list with OpenMP and MPI (hybrid)

An expected good way to take advantage of both paradigms, OpenMP and MPI, is to use a hybrid paradigm, in which we can explore features of i7 processors with OpenMP. Besides, it can be used on several nodes with MPI. Figure 1(c) shows that for above 5 processors the efficiency is frozen. We ran the hybrid in 4 nodes of the cluster that contain the i7 processor, performing only four communication calls. After receiving the atoms by MPI, each node computes the specific region of the cell grid, splitting the tasks through OpenMP, i.e., each node could compute several tasks (8 in the i7 processor) using only one communication.

Figures 2(a), 2(b), 3(a) and 3(b) show the speedup achieved by the proposed methods (CPAs). For small proteins such as 1A11, the OpenMP paradigm is more adequate (Figure 2(a)). That happens since the number of cores of one node is higher than the number of tasks. For protein 1AI0 (Figure 2(b)), the number of processors is significant when considering the hybrid and the OpenMP alone, since, for above 15 processors, the

hybrid approach is fastest. In Figure 3(a), the hybrid is the fastest in all cases. The use of more processors than tasks will again produce the same speedup. Figure 3(b) shows the increase in the speedup for the hybrid approach. In all four cases, the use of MPI isolated is not a good approach. However, when combined with OpenMP, it produces better results.

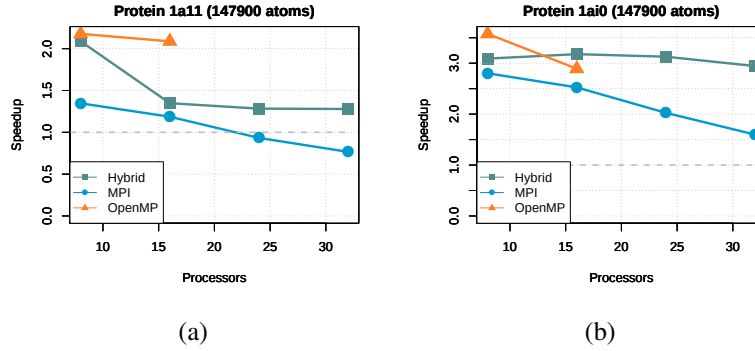


Figure 2. (a) Speedup achieved with the three proposed algorithms for protein 1A11 (390 atoms). (b) Speedup achieved with the three proposed algorithms for protein 1AI0 (4,728 atoms).

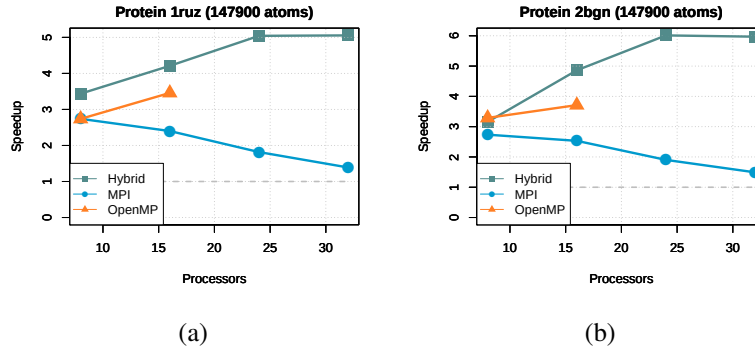


Figure 3. (a) Speedup achieved with the three proposed algorithms for protein 1RUZ (22,380 atoms). (b) Speedup achieved with the three proposed algorithms for protein 2BGN (69,448 atoms).

5. Conclusions

The van der Waals energy used to evaluate the quality of proteins in GA can be efficiently computed using the cell-list technique, which evaluates proteins using an accurate algorithm with linear complexity.

Moreover, flexible parallel techniques applied to cell-list can be used to reduce the run time of the GA. By flexible, we mean the use of distinct parallel programming paradigms in a same algorithm, which, together, can explore diverse benefits available in parallel platforms.

This paper compares the results of three cell-list parallel programs: OpenMP version developed in [Bonetti et al. 2013], MPI developed in [Bonetti et al. 2010] and the new hybrid parallel implementation of cell-list using MPI and OpenMP concomitantly. All parallel versions reduced the running time of the van der Waals energy calculation,

when compared to the sequential version. The hybrid version, however, shows significant speedups independently from the size of the protein.

Our results show that the trade-off between communication and computation times of the MPI algorithm was not good and its speedup is limited. On the other hand, the openMP implementation is more suitable for small proteins, since its communication time is very short when compared to networks in a cluster. Therefore, the hybrid program using MPI and OpenMP has enough flexibility to speedup both small and large sizes of proteins. Results in our experiments show that the hybrid approach presents speedups, regarding the running time and maintain the same accuracy of the results. However, for smaller proteins, as expected, OpenMP using only one node with four cores can achieve speedups near to the speedups of the hybrid solution, offering, in such cases, a better relation cost x benefit.

This paper presented preliminary results belonging to an ongoing research project, focused on discovering new flexible parallel algorithms for PSP. As future work, it will be applied cell-list technique to reduce the complexity of other energy functions such as electrostatic, solvation and hydrogen bond energies, in order to produce more accurate and efficient GA for PSP. Furthermore, the results presented in this paper will be compared with tools like CHARMM, GROMACS, NAMD and AMBER.

References

- Allen, M. P. and Tildesley, D. J. (1987). *Computer Simulation of Liquids*. Oxf. Un. Press.
- Anfinsen, C. B. (1972). Studies on the principles that govern the folding of protein chains. *Nobel Lecture*, pages 103–119.
- Benítez, C. M. V. and Lopes, H. S. (2010). Protein structure prediction with the 3d-hp side-chain model using a master–slave parallel genetic algorithm. *Journal of the Brazilian Computer Society*, 16(1):69–78.
- Bonetti, D. R., Delbem, A. C., Travieso, G., and Souza, P. S. L. (2013). Enhanced van der waals calculations in genetic algorithms for protein structure prediction. *Concurrency and Computation: Practice and Experience*, 25(15):2170–2186.
- Bonetti, D. R. F., Delbem, A. C. B., Travieso, G., and de Souza, P. S. L. (2010). Optimizing van der waals calculi using cell-lists and mpi. In *Evolutionary Computation (CEC), 2010 IEEE Congress on*, pages 1–7.
- Dorn, M., Buriol, L., and Lamb, L. (2011). A hybrid genetic algorithm for the 3-d protein structure prediction problem using a path-relinking strategy. In *Evolutionary Computation (CEC), 2011 IEEE Congress on*, pages 2709–2716.
- Jones, J. E. (1924). On the determination of molecular fields. ii. from the equation of state of a gas. *Proceedings of the Royal Society of London. Series A*, 106(738):463–477.
- Levinthal, C. (1968). Are there pathways for protein folding? *Journal de chimie physique*, 65:44–45.
- Lima, T. W. (2006). Algoritmos evolutivos para predição de estruturas de proteínas. Master's thesis, ICMC-USP, São Carlos, SP.

Geração de carga de trabalho ciente do desempenho dinâmico do sistema

Lourenço Alves Pereira Júnior^{1,2}, Regina H. Carlucci Sanatana²,
Marcos José Santana², Francisco José Monaco²

¹Instituto Tecnológico de Aeronáutica — ITA
ljr@ifsp.edu.br

²Universidade de São Paulo — USP
{ljr,rsc,mjs,monaco}@icmc.usp.br

Abstract. *Workload characterization synthesizes the load behavior of a system by statistical methods whose inputs are commonly from empirical data. Thus, in another moment, one can excite a system with synthetical different workloads that share statistical properties. In this context, to the best of our knowledge, no other study has yet considered the dynamical performance model in the process of workload generation. We take a performance model of a large-scale e-commerce system as a Transfer Function, which allows us to analyze the system behavior in Frequency-domain. We demonstrate cases in which two workloads generated by different distributions lead to similar results and two statistical equivalent workloads lead to inconsistent results.*

Resumo. *A metodologia aplicada à caracterização de carga de trabalho convencional considera dados obtidos empiricamente por meio de métodos estatísticos com a finalidade de criar um modelo capaz de sintetizar o caso real. Dessa forma, pode-se reproduzir a carga original, gerando diferentes instâncias que se equivalem em suas características estatísticas. Neste artigo, apresentamos os benefícios de considerar o desempenho dinâmico de um sistema de e-commerce por meio de um modelo de função de transferência. Apresentamos um caso em que cargas aplicadas ao sistema em estudo apresentam resultados diferentes do esperado, e uma explicação para esse fenômeno está na análise no domínio da frequência proporcionada pela abordagem adotada.*

1. Introdução

A caracterização de carga de trabalho em sistemas computacionais é uma área bem estabelecida e que produz resultados significativos para a avaliação de desempenho de sistemas computacionais [Calzarossa et al. 2016]. Essa técnica aliada a Teoria de Filas formam um arcabouço importante para o planejamento de capacidade de sistemas computacionais. Provêem um modelo analítico capaz de representar, em regime estacionário, características dos sistemas (tais como: taxa de chegada de requisições, tempo de processamento, quantidade de centros de serviço, tamanho da fila, distribuição de probabilidade etc.) e o desempenho do sistema (tempo de resposta, taxa de utilização, *throughput* etc.). Porém, com a efetividade de recursos computacionais ofertados como utilidade (computação em nuvem) [Mell and Grance 2011], um sistema computacional pode aumentar ou diminuir em tempo de execução sua potência. Desse modo, constatamos que os sistemas estão mais suscetíveis a condições de transição (de um estado *a* antes da

adaptação para um estado b após sua efetivação), e os modelos estacionários não capturam esses momentos.

Dado a importância da geração de carga sintética para o projeto de sistemas computacionais de grande escala, neste trabalho buscamos entender como uma carga de trabalho impacta em um sistema computacional, considerando seu comportamento dinâmico. Nesse sentido, este artigo objetiva demonstrar (1) como cargas estatisticamente diferentes podem impactar de modo semelhante dependendo do comportamento dinâmico do sistema em estudo e (2) como a análise de resposta em frequência pode revelar quais componentes impactam mais no sistema em estudo. Ao apresentar estas questões, este trabalho contribui para o estado da arte ao evidenciar os ganhos obtidos por modelar um sistema computacional com uma Função de Transferência, mostrando qual impacto o ruído presente na entrada do modelo gera na variável de saída.

Em estudos anteriores [Pereira et al. 2017b, Pereira et al. 2017a], demonstramos a importância de se conduzir experimentos para identificação de modelos de Função de Transferência e, a partir disso, como realizar análises de desempenho. Se por um lado o foco foi em como mensurar e prever o comportamento enquanto o sistema encontra-se em estado de transição, neste artigo, verificamos como o entendimento do modelo dinâmico impacta em um sistema em estado estacionário. Demonstramos como uma carga estacionária, gerada a partir de uma função de probabilidade, é processada pelo sistema. Evidenciamos que a absorção da carga depende da dinâmica do processamento de requisições do sistema, e que alterações bruscas de curta duração minimamente impactam na variável de saída — respeitada a resposta em frequência do sistema em estudo.

A seguir, na Seção 2 é apresentada a formulação para representação do modelo de desempenho dinâmico como Função de Transferência. A Seção 3 apresenta o modelo considerado para a análise de resposta em frequência. Os trabalhos que se relacionam com o presente são descritos na Seção 4. Por fim, na Seção 5 conclui-se este estudo.

2. Representação do desempenho dinâmico

Para a representação do comportamento dinâmico, é necessário um modelo de desempenho que leve em consideração a variável tempo. Conceitualmente, durante as transições que ocorrem no regime operacional, o desempenho do sistema computacional em estudo é influenciado pelas alterações na carga de trabalho, mas também pelo seu estado interno (i.e: preenchimento de *buffers* e provisionamento de recursos). Dessa forma, precisamos de um modelo matemático capaz de representar como a evolução dinâmica do desempenho ocorre temporalmente. Como apresentado em [Ljung 1999], o modelo autorregressivo atende esses requisitos e pode ser expresso na forma

$$A(q)y(k) = \frac{B(q)}{D(q)}u(k) + \frac{C(q)}{F(q)}v(k), \quad (1)$$

em que $y(k)$ é a variável de desempenho no instante de tempo k (discreto), $u(k)$ é a variável de entrada, $v(k)$ é a representação do ruído. $A(q)$, $B(q)$, $C(q)$ e $D(q)$ e $F(q)$ são polinômios que permitem ponderar como as variáveis irão impactar em $y(k)$; q é um operador de atraso e serve para especificar o estado do sistema, quanto maior o valor de q , maior será a influência de valores passados no valor atual. A razão entre os polinômios é chamada de Função de Transferência.

3. Análise de carga de trabalho sintética

Nesta seção descrevemos o estudo que demonstra a utilização da análise de resposta em frequência como uma ferramenta que auxilia na geração de carga de trabalho em sistemas computacionais. A noção explorada nesta abordagem é que, dependendo da FT do sistema em estudo, a diferença entre duas cargas está na análise de seus espectros. O Diagrama de Bode quantifica como o sistema absorve as flutuações na carga de trabalho, evidenciando como o sistema atenua as frequências da variável de entrada. Dessa forma, as frequências muito atenuadas da entrada provocam pouca influência na variável de saída. Portanto, se duas cargas de trabalho diferem apenas nas componentes de frequência que são muito atenuadas, então, para efeitos práticos, essas duas cargas podem ser consideradas similares.

Iniciamos com a descrição do modelo de desempenho utilizado, seguido pela especificação de um filtro que mantém as frequências que impactam na variável de saída do sistema. A análise segue ao considerar três cargas diferentes: 1) gerada por uma distribuição de probabilidade exponencial com média 300; 2) corresponde à carga 1 (exponencial) filtrada pelo filtro desenvolvido; 3) normal (média 300 e dispersão 67), aproximação da carga 2 (exponencial filtrada).

A Função de Transferência foi o modelo dinâmico adotado para a representação do sistema em estudo, cujos parâmetros foram identificados como

$$G(z) = \frac{b}{1 + az^{-1}} = \frac{0.00273}{1 - 0.9037z^{-1}}. \quad (2)$$

Detalhes do processo de identificação estão descritos em [Pereira et al. 2017b]. Trata-se de um sistema de *e-commerce* de três camadas: apresentação (representado pelos softwares clientes—*browser*), aplicação (composto de 8 servidores) e dados (banco de dados SQL). A taxa de amostragem é de 1 segundo. A variável de entrada do sistema é a quantidade de requisições por segundo (req/s), a saída é a taxa de utilização média dos servidores de aplicação. O ganho DC é 0,0283, o que significa que uma requisição impacta na taxa de utilização em 0,02%. O tempo de assentamento do sistema é de 39 s.

A partir da FT (Eq. 2) é possível gerar sua representação em Diagrama de Bode. Nele podemos quantificar o quanto do sinal de entrada (req/s) será atenuado no sinal de saída. Se há uma quantidade de variação grande e brusca no valor de entrada, a frequência com que alterações ocorre é alta. Se mudanças no sinal de entrada são duradouras, a frequência é baixa. Pelo Diagrama de Bode (inspecionar Figura 1(a)), percebemos que baixas frequências são absorvidas pelo sistema, e isso significa que quando elas acontecem, novos patamares estacionários são alcançados. Por outro lado, à medida que se aumenta a frequência com que mudanças acontecem, elas tendem a não ter efeito na saída. Frequências altas caracterizam-se por picos e vales seguidos, que, em efeitos práticos, anulam o impacto no sistema.

3.1. Análise das cargas de trabalho sintéticas

Gerou-se um sinal oriundo de um gerador de números pseudo-aleatório (PRNG); o tamanho desse sinal será de 2^{20} segundos (aproximadamente 1 milhão de amostras), ou seja, uma simulação de 291 horas, pois cada amostra corresponde a 1 segundo. Especificamos dois sinais entrada: 1) **Exponencial**: sinal gerado por uma distribuição de probabilidade

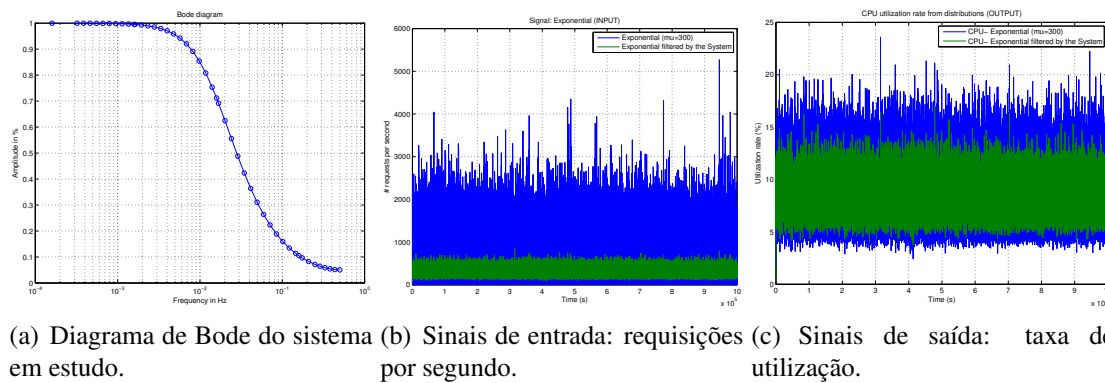


Figura 1. O desempenho do sistema Eq. 2 é semelhante ao receber duas cargas de trabalho diferentes que possuem as mesmas componentes de baixa frequência.

exponencial de média 300 e 2) **Filtrado**: produzido pelo filtro que representa a dinâmica do sistema, cuja entrada é o sinal exponencial ([Pereira et al. 2017b]). A Figura 1(b) apresenta os sinais exponencial e filtrado no domínio do tempo. Esses dois sinais foram passados para a Função de Transferência do sistema (Eq. 2), produzindo os sinais de saída apresentados na Figura 1(c).

O sinal exponencial possui média 299,82 e desvio padrão 299,90, enquanto o sinal filtrado 299,81 e 67,33 para média e desvio padrão, respectivamente. Os sinais de saída para o exponencial possuem média 8,49 e desvio padrão de 1,90, ao passo que para a saída do filtrado 8,49 de média e 1,35 para desvio padrão. Ao dividir a média dos sinais de saída pela média dos sinais de entrada, obtém-se o ganho de 0,02835, o que é condizente com o sistema. Ao dividir o desvio padrão dos sinais de entrada, obtém-se 0,22452 ($67,33/299,9$), fazendo a mesma operação para os sinais de saída, tem-se como resultado 0,7105 ($1,35/1,9$). Isso significa que o sinal filtrado possui uma dispersão correspondente a 22,45% do sinal exponencial original. Porém, nos sinais de saída, o resultado dessa razão aumenta para 70,75%, ou seja, boa parte da dispersão é filtrada pelo sistema, o que faz com que os sinais de saída sejam menos dispersos do que os de entrada em cerca de 50% ($70,75 - 22,45$).

Seguindo adiante, podemos aproximar o sinal da exponencial filtrada ao de uma normal e novamente passar os dois sinais (normal e filtrado) pela FT. A intuição neste instante, é que o resultado seria equivalente, pois teríamos dois sinais que compartilham de propriedades estatísticas. Foi gerado um sinal com 1 milhão de amostras oriundo de um PRNG com distribuição normal com média 300 e dispersão 67; denominamos esse sinal como normal. Os sinais de entrada filtrado e normal foram passados pelo sistema o que gera o sinal de saída. O sinal normal possui média 299.96 e desvio padrão 66.97. Para o sinal normal, ao dividir a média de sua saída pela média do respectivo sinal de entrada, obtém-se o ganho de 0,02835, o que é condizente com o sistema. Ao dividir a dispersão dos sinais de entrada (dispersão do normal pelo filtrado), obtém-se 0,99361, enquanto que, para a mesma operação considerando os sinais de saída, tem-se 0,31719. O que significa que há uma semelhança estatística entre os sinais de entrada, porém os de saída são diferentes. O que leva à evidência de que duas cargas de trabalho semelhantes apresentam resultados diferentes. O que é contraditório ao esperado.

Para observar a semelhança dos sinais filtrado e normal e a diferença apresentada nos resultados, calculou-se a Função Densidade de Probabilidade (PDF) com a função `ksdensity` disponível no Matlab ©. Essa função recebe uma série de números e estima como seria a PDF da série. Observou-se uma boa aproximação para os sinais de entrada, porém uma distribuição diferente para a saída, muito embora a média seja a aproximadamente a mesma.

Até o momento, tem-se que para duas distribuições diferentes os resultados produzidos foram semelhantes, e que para duas distribuições semelhantes os resultados foram diferentes. Ou seja, o resultado sugere que as estatísticas de média e dispersão da distribuição, nesse caso, influencia pouco nesse sistema. A análise da resposta em frequência que considera tanto o sinal de entrada quanto o comportamento do sistema pode revelar o ponto que explica tais resultados.

Os três sinais considerados até o momento (exponencial, filtrado e normal) foram passados para o algoritmo Fast Fourier Transform (FFT). Do vetor resultante, obteve-se o valor correspondente aos harmônicos que geram o sinal no domínio do tempo. Observou-se que o sinal exponencial é composto pela média (300), correspondente ao harmônico fundamental e um ruído branco, isto é, uma grande quantidade de harmônicos distribuídos aproximadamente de forma isonômica na faixa de frequência. Ao passar o sinal exponencial pelo filtro especificado, parte das frequências são atenuadas, de modo que as componentes mais altas (próximas a 0,5 Hz) tendessem a zero, conforme previsto pelo Diagrama de Bode (Figura 1(a)). Há uma concentração de harmônico de baixa ordem.

A FFT do sinal normal difere dos outros dois sinais. É diferente da FFT do sinal exponencial pois as frequências do sinal normal possuem amplitude menor. Muito embora os dois sinais tenham a mesma média, as diferenças fazem com que os dois sinais tenham probabilidade de distribuição diferentes. A exponencial com harmônicos de maior amplitude propiciam picos maiores no domínio do tempo, o que é característico de sua cauda. A FFT do sinal filtrado quando comparada à FFT do sinal normal possui harmônicos de baixa frequência de maior amplitude. Esse fato impacta na saída do sistema, pois a presença de baixas frequências significa uma maior excursão do sinal resultante, e isso foi observado nos resultados, uma vez que a dispersão do sinal de saída do filtrado foi maior que o respectivo do normal.

4. Trabalhos relacionados

Não encontramos outros trabalhos que consideram Função de Transferência (FT) como modelo de desempenho e empregue Análise de Resposta em Frequência para entender como o sistema processa uma carga de trabalho caracterizada. A aplicação comum de FT tem sido empregada como base para implementação de controladores de recursos baseados em teoria de controle [Shevtsov et al. 2017]. No entanto, a FT pode ser empregada como modelo de desempenho complementado as melhores práticas correntes [Yang and Liu 2012]. Nesse sentido, há trabalhos que focam nessa temática [Pereira et al. 2017b, Pereira et al. 2015, Mamani et al. 2015, Pereira et al. 2017a], porém nenhum deles apresentou o impacto da resposta em frequência de um sistema em regime estacionário, como apresentado neste artigo. Nosso trabalho avança o estado da arte por prover uma visão da FT como modelo de desempenho e analisar como é impacto de cargas de trabalho estacionárias em um sistema computaci-

onal previamente modelado.

5. Conclusões

A conclusão desse estudo é que o fato de haver casos em que o sistema é excitado por (1) distribuições diferentes que geram resultados semelhantes e (2) distribuições semelhantes que geram resultados diferentes, depende não somente da distribuição estatística utilizada, mas como o sistema absorve a carga de trabalho imposta a ele. Os resultados evidenciam a importância da abordagem da modelagem de sistemas computacionais através de modelos dinâmicos e destaca o uso da análise em frequência, pelos Diagrama de Bode e FFT, para obter comportamentos importantes a respeito do desempenho do sistema. Como trabalhos futuros, pretende-se avaliar sistemas com modelos não lineares (NARX) e de múltiplas entradas e saídas (MIMO).

Agradecimentos

Agradecemos CAPES, FAPESP, CNPq, LaSDPC/USP e IFSP pelo apoio financeiro.

Referências

- Calzarossa, M. C., Massari, L., and Tessera, D. (2016). Workload characterization: A survey revisited. *ACM Comput. Surv.*, 48(3):48:1–48:43.
- Ljung, L. (1999). *System Identification: Theory for the User*. Pearson Education.
- Mamani, E. L. C., Pereira, L. A., Santana, M. J., Santana, R. H. C., Nobile, P. N., and Monaco, F. J. (2015). Transient performance evaluation of cloud computing applications and dynamic resource control in large-scale distributed systems. In *High Performance Computing Simulation (HPCS), 2015 International Conference on*, pages 246–253.
- Mell, P. M. and Grance, T. (2011). Sp 800-145. the nist definition of cloud computing. Technical report, National Institute of Standards & Technology, Gaithersburg, USA.
- Pereira, L. A., dos Santos de Souza, F. L., Mamani, E., and Monaco, F. J. (2017a). Geração de carga de trabalho transiente para aplicações de e-commerce multicamadas. In *XIV Workshop de Computação em Clouds e Aplicações*, Belém, PA. SBC.
- Pereira, L. A., Mamani, E., Santana, R., Santana, M., and Monaco, F. J. (2017b). Análise de resposta em frequência para modelagem e geração de carga de trabalho em aplicações de e-commerce. In *XXXV Simpósio Brasileiro de Redes de Computadores e Sistemas Distribuídos (SBRC)*, pages 888–901.
- Pereira, L. A., Mamani, E. L. C., Santana, M. J., Santana, R. H. C., Nobile, P. N., and Monaco, F. J. (2015). Non-stationary simulation of computer systems and dynamic performance evaluation: A concern-based approach and case study on cloud computing. In *Computer Architecture and High Performance Computing (SBAC-PAD), 2015 27th International Symposium on*, pages 130–137.
- Shevtsov, S., Berekmeri, M., Weyns, D., and Maggio, M. (2017). Control-theoretical software adaptation: A systematic literature review. *IEEE Transactions on Software Engineering*, PP(99):1–1.
- Yang, F. and Liu, J. (2012). Simulation-based transfer function modeling for transient analysis of general queueing systems. *European Journal of Operational Research*.

Patrocinador Diamante



GOVERNO
DO RIO GRANDE DO NORTE

Patrocinadores Bronze



Apoio Financeiro



MINISTÉRIO DA
EDUCAÇÃO

