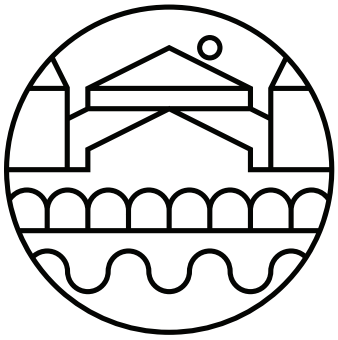
XXXV
SIMPÓSIO BRASILEIRO DE
REDES DE COMPUTADORES
E SISTEMAS DISTRIBUÍDOS

15 a 19 de maio de 2017
Belém - Pará

Anais XV WCGA 2017



XXXV
SIMPÓSIO BRASILEIRO DE
REDES DE COMPUTADORES
E SISTEMAS DISTRIBUÍDOS

15 a 19 de maio de 2017
Belém - Pará

Anais do XV WCGA 2017
Workshop em Clouds e Aplicações

Editora
Sociedade Brasileira de Computação (SBC)

Organização
Bruno Richard Schulze (LNCC)
Luiz Fernando Bittencourt (UNICAMP)
Tiago Coelho Ferreto (PUCRS)
Ronaldo Alves Ferreira (UFMS)
Antônio Jorge Gomes Abelém (UFPA)
Eduardo Coelho Cerqueira (UFPA)

Realização
Sociedade Brasileira de Computação (SBC)
Universidade Federal do Pará (UFPA)
Laboratório Nacional de Redes de Computadores (LARC)

Capa: Catarina Nefertari (PCT-UFPA)
Produção Editorial: Denis Lima do Rosário (UFPA)

Cópias Adicionais:
Sociedade Brasileira de Computação (SBC)
Av. Bento Gonçalves, 9500- Setor 4 - Prédio 43.412 - Sala 219
Bairro Agronomia - CEP 91.509-900 - Porto Alegre - RS
Fone: (51) 3308-6835
E-mail: sbc@sbc.org.br

XV Workshop em Clouds e Aplicações (15: 2017: Belém, Pa).

Anais / XV Workshop em Clouds e Aplicações – WCGA; organizado por Antônio Jorge Gomes Abelém, Eduardo Coelho Cerqueira, Ronaldo Alves Ferreira, Bruno Richard Schulze, Luiz Fernando Bittencourt, Tiago Coelho Ferreto - Porto Alegre: SBC, 2017

171 p. il. 21 cm.
Vários autores
Inclui bibliografias

1. Redes de Computadores. 2. Sistemas Distribuídos. I. Abelém, Antônio Jorge Gomes II. Cerqueira, Eduardo Coelho III. Ferreira, Ronaldo Alves IV. Schulze, Bruno Richard V. Bittencourt, Luiz Fernando VI. Ferreto, Tiago Coelho VII. Título.

Sociedade Brasileira da Computação

Contato
Av. Bento Gonçalves, 9500
Setor 4 - Prédio 43.412 - Sala 219
Bairro Agronomia
91.509-900 – Porto Alegre RS
CNPJ: 29.532.264/0001-78
http://www.sbrc.org.br

Laboratório Nacional de Redes de Computadores (LARC)

Diretora do Conselho Técnico-Científico
Rossana Maria de C. Andrade (UFC)

Vice-Diretor do Conselho Técnico-Científico
Ronaldo Alves Ferreira (UFMS)

Diretor Executivo
Paulo André da Silva Gonçalves (UFPE)

Vice-Diretor Executivo
Elias P. Duarte Jr. (UFPR)

Membros Institucionais
SESU/MEC, INPE/MCT, UFRGS, UFMG, UFPE, UFCG (ex-UFPB Campus Campina Grande), UFRJ, USP, PUC-Rio, UNICAMP, LNCC, IME, UFSC, UTFPR, UFC, UFF, UFSCar, IFCE (CEFET-CE), UFRN, UFES, UFBA, UNIFACS, UECE, UFPR, UFPA, UFAM, UFABC, PUCPR, UFMS, UnB, PUC-RS, PUCMG, UNIRIO, UFS e UFU.

Contato
Universidade Federal de Pernambuco - UFPE
Centro de Informática - CIn
Av. Jornalista Anibal Fernandes, s/n
Cidade Universitária
50.740-560 - Recife - PE
http://www.larc.org.br

Organização do SBRC 2017

Coordenadores Gerais
Antônio Jorge Gomes Abelém (UFPA)
Eduardo Coelho Cerqueira (UFPA)

Coordenadores do Comitê de Programa
Edmundo Roberto Mauro Madeira (UNICAMP)
Michele Nogueira Lima (UFPR)

Coordenador de Palestras e Tutoriais
Edmundo Souza e Silva (UFRJ)

Coordenador de Painéis e Debates
Luciano Paschoal Gaspary (UFRGS)

Coordenadores de Minicursos
Heitor Soares Ramos (UFAL)
Stênio Flávio de Lacerda Fernandes (UFPE)

Coordenadora de Workshops
Ronaldo Alves Ferreira (UFMS)

Coordenador do Salão de Ferramentas
Fabio Luciano Verdi (UFSCar)

Comitê de Organização Local
Adailton Lima (UFPA)
Alessandra Natasha (CESUPA)
Davis Oliveria (SERPRO)
Denis Rosário (UFPA)
Elisangela Aguiar (SERPRO)
João Santana (UFRA)
Josivaldo Araújo (UFPA)
Marcos Seruffo (UFPA)
Paulo Henrique Bezerra (IFPA)
Rômulo Pinheiro (UNAMA)
Ronedo Ferreira (META)
Thiêgo Nunes (IFPA)
Vagner Nascimento (UNAMA)

Comite Consultivo
Allan Edgard Silva Freitas (IFBA)
Antonio Alfredo Ferreira Loureiro (UFMG)
Christian Esteve Rothenberg (UNICAMP)
Fabíola Gonçalves Pereira Greve (UFBA)
Frank Augusto Siqueira (UFSC)
Jussara Marques de Almeida (UFMG)
Magnos Martinello (UFES)
Antonio Marinho Pilla Barcellos (UFRGS)
Moisés Renato Nunes Ribeiro (UFES)
Rossana Maria de Castro Andrade (UFC)

Mensagem dos Coordenadores Gerais

Sejam bem-vindos ao 35o Simpósio Brasileiro de Redes de Computadores e Sistemas Distribuídos (SBRC 2017) e a acolhedora cidade das mangueiras - Belém / Pará.

Organizar uma edição do SBRC pela segunda vez no Norte do Brasil é um desafio e um privilégio por poder contribuir com a comunidade de Redes de Computadores e Sistemas Distribuídos do Brasil e do exterior. O SBRC se destaca como um importante celeiro para a discussão, troca de conhecimento e apresentação de trabalhos científicos de qualidade.

A programação do SBRC 2017 está diversificada e discute temas relevantes no cenário nacional e internacional. A contribuição da comunidade científica brasileira foi de fundamental importância para manter a qualidade técnica dos trabalhos e fortalecer a ciência, tecnologia e inovação no Brasil.

Após um cuidadoso processo de avaliação, foram selecionados 77 artigos completos organizados em 26 sessões técnicas e 10 ferramentas para apresentação durante o Salão de Ferramentas. Além disso, o evento contou com 3 palestras e 3 tutoriais proferidos por pesquisadores internacionalmente renomados, 3 painéis de discussões e debates, todos sobre temas super atuais, 6 minicursos envolvendo Big Data, sistemas de transportes inteligentes, rádios definidos por software, fiscalização e neutralidade da rede, mecanismos de autenticação e autorização para nuvens computacionais e comunicação por luz visível, bem como 10 workshops.

O prêmio “Destaque da SBRC” e uma série de homenagens foram prestadas para personalidades que contribuíram e contribuem com a área. O apoio incondicional da SBC, do LARC, do Comitê Consultivo da SBRC e da Comissão Especial de Redes de Computadores e Sistemas Distribuídos da SBC foram determinantes para o sucesso do evento. A realização do evento também contou com o importante apoio do Comitê Gestor da Internet no Brasil (CGI.br), do CNPq, da CAPES, do Parque de Ciência e Tecnologia Guamá, da Connecta Networking, da Dantec Telecom, da RNP e do Google. Nosso especial agradecimento à Universidade Federal do Pará (UFPA) e ao Instituto Federal do Pará (IFPA) pelo indispensável suporte à realização do evento.

Nosso agradecimento também para os competentes e incansáveis coordenadores do comitê do programa (Michele Nogueira/UFRP – Edmundo Madeira/UNICAMP), aos coordenadores dos minicursos (Stênio Fernandes/UFPE – Heitor Ramos/UFAL), ao coordenador dos workshops (Ronaldo Ferreira/UFMS), ao coordenador de painéis e debates (Luciano Gaspary/UFRGS), ao coordenador do Salão de Ferramentas (Fabio Verdi/UFSCar) e ao coordenador de palestras e tutoriais (Edmundo Souza e Silva/UFRJ). Destacamos o excelente trabalho do comitê de organização local coordenado por Denis do Rosário.

Por fim, desejamos a todos uma produtiva semana em Belém.

Antônio Abelém e Eduardo Cerqueira

Coordenadores Gerais do SBRC 2017

Mensagem do Coordenador de Workshops

É com grande prazer que os convido a prestigiar os workshops do Simpósio Brasileiro de Redes de Computadores e Sistemas Distribuídos (SBRC) nos dias 15, 16 e 19 de maio de 2017. Tradicionalmente, os workshops abrem e fecham a semana do SBRC e são responsáveis por atrair uma parcela expressiva de participantes para o Simpósio. Como coordenador de workshops, dividi com os coordenadores gerais do SBRC a nobre tarefa de selecionar os workshops que melhor representam a comunidade e que fortaleçam novas linhas de pesquisa ou mantenham em evidência linhas de pesquisa tradicionais.

Em resposta à chamada aberta de workshops, recebemos dez propostas de alta qualidade, das quais nove foram selecionadas. Além disso, mantivemos a longa colaboração com a RNP por meio da organização do WRNP, que já é uma tradição na segunda e terça-feira da semana do SBRC. Dentre as propostas aceitas, sete são reedições de workshops tradicionais do SBRC que já são considerados parte do circuito nacional de divulgação científica nas várias subáreas de Redes de Computadores e Sistemas Distribuídos, como o WGRS (Workshop de Gerência e Operação de Redes e Serviços), o WTF (Workshop de Testes e Tolerância a Falhas), o WCGA (Workshop em Clouds, Grids e Aplicações), o WP2P+ (Workshop de Redes P2P, Dinâmicas, Sociais e Orientadas a Conteúdo), o WPEIF (Workshop de Pesquisa Experimental da Internet do Futuro), o WoSiDA (Workshop de Sistemas Distribuídos Autonômicos) e o WoCCES (Workshop de Comunicação de Sistemas Embarcados Críticos). Como novidade, teremos dois novos workshops com programação diversificada e grande apelo social, o CoUrb (Workshop de Computação Urbana) e o WTICp/D (Workshop de TIC para Desenvolvimento).

Temos certeza que 2017 será mais um ano de sucesso para os workshops do SBRC pelo importante papel de agregação que eles exercem na comunidade científica de Redes de Computadores e Sistemas Distribuídos no Brasil.

Aproveitamos para agradecer o apoio recebido de diversos membros da comunidade e, em particular, a cada coordenador de workshop, pelo brilhante trabalho. Como coordenador dos workshops, agradeço imensamente o apoio recebido da Organização Geral do SBRC 2017.

Esperamos que vocês aproveitem não somente os workshops, mas também todo o SBRC e as inúmeras atrações de Belém.

Ronaldo Alves Ferreira

Coordenador de Workshops do SBRC 2017

Mensagem dos Coordenadores do XV WCGA 2017

Sejam bem-vindos ao XV Workshop de Computação em Clouds e Aplicações (WCGA 2017), evento realizado em Belém-PA em conjunto com o XXXV Simpósio Brasileiro de Redes de Computadores e Sistemas Distribuídos (SBRC 2017).

A computação em nuvem continua sendo um tema de interesse da comunidade científica, pois com a sua consolidação também são identificados novos problemas e desafios, como garantias de rede, segurança e execução de aplicações com qualidade de serviço, além de limitações que são contornadas com novas propostas de arquiteturas distribuídas complementares às nuvens, como por exemplo a computação em névoa. A variedade de tópicos que a computação em nuvem tem atraído tem reflexo no programa do WCGA, que conta com artigos relacionados a temas como: elasticidade de aplicações, alocação de máquinas virtuais, eficiência energética, migração entre nuvens, entre outros. Os coordenadores do WCGA agradecem todos os autores que submeteram seus trabalhos e que permitem a continuidade deste workshop com trabalhos de qualidade que permitem a troca de experiência e discussão de pesquisas em um nível elevado.

Agradecemos os coordenadores do SBRC e a coordenação de workshops pelo apoio na realização do WCGA, que vem, desde sua quarta edição em 2006, sendo executado em conjunto com o SBRC. As primeiras edições do WCGA (2003, 2004, 2005) foram realizadas no LNCC, em Petrópolis-RJ e, em conjunto com o SBRC, as edições seguintes foram realizadas em Curitiba-PR (2006), Belém-PA (2007), Rio de Janeiro-RJ (2008), Recife-PE (2009), Gramado-RS (2010), Campo Grande-MS (2011), Ouro Preto-MG (2012), Brasília-DF (2013), Florianópolis-SC (2014), Vitória-ES (2015), Salvador-BA (2016), culminando nesta décima quinta edição de 2017 em Belém do Pará.

O WCGA 2017 recebeu um total de 29 submissões, dentre as quais 12 artigos completos foram aceitos para publicação e apresentação oral no evento. Reiteramos o nosso agradecimento aos autores de artigos submetidos, e enaltecemos o trabalho dos membros do comitê de programa e revisores externos nas avaliações dos artigos. A excelente qualidade do Comitê de Programa do WCGA permitiu uma avaliação extensiva e criteriosa dos trabalhos submetidos. Por fim, agradecemos as entidades que acolhem e apoiam os organizadores do WCGA e SBRC: UNICAMP, LNCC, PUCRS, UFPA e SBC.

Luiz Bittencourt, Bruno Schulze e Tiago Ferreto

Coordenadores do WCGA 2017

Comitê de Programa

- Andrey Brito (UCG)
- Antônio Tadeu Azevedo Gomes (LNCC)
- Antonio Mury (LNCC)
- Artur Ziviani (LNCC)
- Carlos Ferraz (UFPE)
- Christian Esteve Rothenberg (UNICAMP)
- Claudio Geyer (UFRGS)
- Cristina Boeres (UFF)
- Daniel Batista (USP)
- Daniel de Oliveira (UFF)
- Fabrício da Silva (CETEX)
- Frederico Lopes (UFRN)
- Hélio Guardia (UFSCar)
- Hermes Senger (UFSCAR)
- Lisandro Granville (UFRGS)
- Lourenço Alves Pereira Júnior (IFSP)
- Luis Carlos de Bona (UFPR)
- Marco Netto (IBM Research)
- Marcos Assunção (INRIA, França)
- Nabor Mendonça (Unifor)
- Nelson Fonseca (UNICAMP)
- Noemi Rodrigues (PUC-Rio)
- Paulo Ferreira (INESC-ID, PT)
- Philippe Navaux (UFRGS)
- Raphael Camargo (UFABC)
- Rodrigo Righi (UNISINOS)
- Rossana Andrade (UFC)
- Stenio Fernandes (UFPE)
- Thaís Batista (UFRN)

Sumário

XV Workshop de Computação em Clouds e Aplicações

SBRC 2017

Sessão Técnica 1

EMA-Bench: Um Benchmark para a Avaliação de Mecanismos de Alocação Elástica de Memória

Guilherme Galante[1], **Cristiane Andrade**[1],
Luiz Antonio Rodrigues[1], **Rodrigo da Rosa Righi**[2]

[1]Ciência da Computação – Unioeste
Cascavel – PR – Brasil

[2]Programa Pós-Graduação em Computação Aplicada – Unisinos
São Leopoldo – RS – Brasil

guilherme.galante@unioeste.br

Abstract. *Several works have addressed the development of mechanisms for elastic allocation of memory, creating a need for a precise way to evaluate such solutions. Analyzing the technical literature, we found a gap in the state of the art in the evaluation of vertical elasticity, since the proposals of metrics and methodologies focus on the evaluation of horizontal elasticity. In this sense, this work contributes with the development of the EMA-Bench, a benchmark for evaluating elastic memory allocation mechanisms. The benchmark enables to analyse a mecanism in terms of accuracy and time proportions spent in over-provisioning and overprovisioning states, as well as the financial costs involved. The results show that the proposed tool is able to assist the user to define the best mechanism to be adopted, ensuring cost reduction and the maintenance of quality of service. In addition, the EMA-Bench can be used by other researchers in the comparison and refinement of experiments and elastic solutions.*

Resumo. *Diversos trabalhos têm sido realizados com o objetivo de apresentar mecanismos para alocação de elástica de memória, se fazendo necessário métricas e metodologias capazes de avaliar tais soluções. Analisando a literatura técnica sobre o assunto, observa-se que as propostas de métricas e metodologias se concentram na avaliação de mecanismos de elasticidade horizontal, havendo uma lacuna no estado-da-arte para a avaliação da elasticidade vertical. Nesse sentido, este trabalho tem como principal contribuição o desenvolvimento do EMA-Bench, um benchmark para a avaliação de mecanismos de alocação elástica de memória. O benchmark fornece um conjunto de métricas para a avaliação da precisão e dos períodos dispendidos em períodos de sub-provisionamento e superprovisionamento, bem como o cálculo dos custos envolvidos. Os resultados mostram que a ferramenta proposta é capaz de auxiliar o usuário na definição de qual é o melhor mecanismo a ser adotado, garantindo redução de custo e a manutenção da qualidade do serviço. Além disso, o EMA-Bench pode ser utilizado por outros pesquisadores na comparação e refinamento de experimentos e de soluções elásticas.*

1. Introdução

A elasticidade é uma das principais características da computação em nuvem. Esta característica consiste na capacidade de adicionar ou remover recursos, sem interrupções e em tempo de execução, de acordo com uma demanda específica [Li 2017]. A capacidade de elasticamente expandir e contrair a base de recursos é interessante tanto para o provedor quanto para o usuário final da nuvem. Do ponto de vista do provedor, a elasticidade garante um melhor uso dos recursos de computação, fornecendo economia de escala e permitindo que mais usuários possam ser atendidos simultaneamente, uma vez que os recursos liberados por um usuário podem instantaneamente ser alocados por outro. Da perspectiva do usuário, a elasticidade pode ser usada para diversos fins, tais como manutenção da qualidade de serviço, complementação de recursos e redução de custos.

Dependendo da forma como a nuvem implementa o provisionamento de recursos, pode-se classificar a sua elasticidade em horizontal ou vertical [Galante and Bona 2012]. Uma nuvem com elasticidade horizontal possibilita apenas a adição ou a remoção dinâmica de instâncias alocadas por um usuário. Por sua vez, a elasticidade vertical é caracterizada pela possibilidade de se alterar a capacidade de VMs em execução. Tipicamente, a elasticidade vertical é implementada através da adição ou remoção de CPUs e memória, mas também pode ser utilizada no contexto de redes e armazenamento.

A elasticidade vertical é considerada uma das tecnologias chave para a concretização do conceito de Recurso Como Serviço (RaaS) [Ben-Yehuda et al. 2012] e uma das principais características da segunda geração de Infraestrutura como Serviço (IaaS 2.0), na qual os usuários pagam apenas pelos recursos que eles realmente usam, e os provedores de nuvem podem usar seus recursos de forma mais eficiente e servir mais usuários [Farokhi et al. 2016]. Embora atualmente não esteja implementada na maioria dos provedores de nuvem, a elasticidade vertical já é suportada em hipervisores como Xen e KVM e implementado por diversos mecanismos desenvolvidos pela academia [Galante and Bona 2012, Galante et al. 2016].

Nesse contexto, se fazem necessárias métricas e metodologias capazes de avaliar os mecanismos de elasticidade disponíveis com o intuito de auxiliar o usuário a definir qual é o mais apropriado para suas demandas. Em uma revisão da literatura técnica sobre o assunto, percebe-se que as propostas de métricas e metodologias se concentram na avaliação de mecanismos de elasticidade horizontal, havendo uma lacuna no estado-da-arte para a avaliação da elasticidade vertical. Nesse sentido, este trabalho tem como principal contribuição o desenvolvimento do *Elastic Memory Allocation Benchmark* (EMA-Bench), um *benchmark* para a avaliação de mecanismos de alocação elástica de memória. O *benchmark* implementa um conjunto de métricas para a avaliação da precisão e dos períodos dispendidos em períodos de subprovisionamento e superprovisionamento, bem como o cálculo dos custo envolvidos. Dessa forma, a ferramenta auxilia o usuário a definir qual é o mecanismo mais adequado a sua demanda, garantindo redução de custos e a manutenção da qualidade do serviço.

O restante do trabalho está organizado como segue. A Seção 2 apresenta os trabalhos relacionados. A Seção 3 apresenta uma breve revisão sobre a elasticidade vertical de memória. Na Seção 4, descreve-se o *benchmark* proposto. Na Seção 5, apresenta-se a metodologia de avaliação, os experimentos, bem como a discussão dos resultados. Por fim, a Seção 6 conclui este trabalho.

2. Trabalhos Relacionados

Diversas estratégias para a avaliação da elasticidade foram propostas na literatura. De modo geral, os trabalhos propõem metodologias e métricas para mensurar a elasticidade de nuvens de infraestrutura (IaaS).

Islam et al. (2009) define uma métrica para elasticidade baseadas em dois elementos: tempo e custo. A métrica reflete a penalidade financeira para um determinado usuário nos casos de subprovisionamento, que pode resultar em demandas não atendidas ou Acordo de Nível de Serviço (*Service Level Agreement* - SLA) não cumprido, ou de superprovisionamento, onde paga-se mais que o necessário pelos recursos para suportar uma carga de trabalho. No trabalho de Coutinho et al. (2014), os autores apresentam métricas para a medição da elasticidade horizontal baseada em conceitos da física, como estresse e tensão. Faz análise apenas sobre o uso de CPU e número de instâncias alocadas.

Hwang et al. (2016) propõem uma métrica de desempenho da nuvem em termos de eficiência e produtividade. Os experimentos são realizados na nuvem Amazon EC2, na qual os recursos são dimensionados pela quantidade e tipos de instâncias de máquina virtual. Por sua vez, Beltrán (2016) define uma métrica de elasticidade para ambientes de computação em nuvem que considera quatro componentes: escalabilidade, precisão, tempo e custo, além de descrever um procedimento para analisar a elasticidade em contextos de nuvem.

Por fim, Herbst et al. (2015, 2016) propõem um conjunto de métricas para caracterizar a elasticidade de uma plataforma de nuvem, e uma metodologia de *benchmarking*, denominada BUNGEE, para avaliar a elasticidade horizontal de plataformas de nuvem IaaS. As métricas pretendem avaliar os mecanismos de elasticidade em termos de acurácia e tempo dispendido em estados de sub e superprovisionamento.

Considerando os trabalhos apresentados, observa-se que todos se concentram na avaliação da elasticidade horizontal em nuvens IaaS, havendo uma lacuna no estado-da-arte para a avaliação da elasticidade vertical. Nesse sentido, propõem-se um *benchmark* para a avaliação da elasticidade vertical de memória. O *benchmark* se baseia no trabalho de Herbst et al. (2016), porém adaptado para elasticidade vertical de memória.

3. Elasticidade Vertical de Memória

Estimar a quantidade exata de memória necessária por uma determinada aplicação ou serviço não é uma tarefa trivial. Usuários tendem a superestimar seus requisitos de memória baseando-se nos cenários de pior caso, que na prática, pode ser necessário apenas em curtos períodos de tempo. Isso afeta diretamente o aproveitamento da infraestrutura da nuvem, pois resulta em memória não utilizada e que poderia ser dedicada a instâncias adicionais rodando em uma mesma máquina física. Como resultado, técnicas de *overbooking*, nas quais mais recursos são alocados do que fisicamente existem, tem se tornado prática comum [Farokhi et al. 2016]. Embora a abordagem permita aproveitar melhor a capacidade dos recursos, ela também pode causar impacto no desempenho de aplicações e violações de SLA.

Dessa forma, a exploração da elasticidade vertical pode apresentar diversas vantagens. É possível fornecer continuamente a quantidade necessária de memória considerando que aplicações distintas têm necessidades de memória diferentes e que suas car-

gas de trabalho podem modificar-se significativamente ao logo de sua execução. Assim, remove-se o ônus colocado sobre os usuários para que estimem com precisão os recursos a serem utilizados, como também possibilita que os provedores aumentem a eficiência do uso de sua infraestrutura e reduzam o consumo de energia, despesas de capital e custos de administração.

Os hipervisores modernos oferecem suporte à elasticidade de memória usando técnicas como compartilhamento de páginas, *hotplug* virtual e *ballooning* [Zhang et al. 2017] e embora haja esse suporte, a decisão de quando e como as alocações de memória devem ser feitas é tomada por mecanismos de elasticidade com controle mais elaborado. Nesse contexto, diversos trabalhos têm sido realizados com o objetivo de apresentar soluções para alocação de elástica de memória. Basicamente, as propostas se dividem em dois grupos: reativas e proativas [Galante and Bona 2012].

Soluções reativas empregam mecanismos do tipo Regra-Condição-Ação, onde cada condição considera um evento ou uma métrica do sistema que é comparado com um limiar determinado e quando uma condição definida em alguma dessas regras é satisfeita, uma ação é disparada. Por sua vez, as abordagens proativas utilizam técnicas analítico-matemáticas e heurísticas, para obter uma previsão sobre o comportamento das cargas do sistema e, a partir desses resultados, tomar decisões de como e quando escalar os recursos. Essas técnicas incluem análises de séries temporais [Spinner et al. 2015], Transformada de Fourier [Gong et al. 2010], cadeias de Markov e redes Bayesianas [Tan et al. 2012]. A abordagem proativa é mais apropriada para os casos onde a carga de trabalho apresenta padrões bem definidos com periodicidade uniforme, facilitando a previsão de cargas futuras.

4. EMA-Bench

Para possibilitar a avaliação de soluções de elasticidade vertical de memória deve-se haver uma abordagem bem definida, com o objetivo de fornecer suporte às decisões do usuário quanto a eficácia dos método e se satisfazem as suas demandas. É com este objetivo que o EMA-Bench foi proposto.

O EMA-Bench foi implementado em C++ e é executado em linha de comando. Conforme ilustrado na Figura 1, possui como entrada (1) um perfil de consumo de memória, (2) a indicação de qual método implementado no *benchmark* será executado e (3) os seus respectivos parâmetros de configuração. Após o processamento, fornece três resultados: (4) alocações dinâmicas de memória realizadas pelo mecanismo selecionado, fornecido como um arquivo texto, (5) resultado das métricas e (6) informações sobre o custo. Os detalhes das entradas e saídas da ferramenta são descritos na Seção 4.3.

Atualmente, o *benchmark* tem implementado três mecanismos de elasticidade (apresentados em detalhes na Seção 4.1), porém não se limita a esses mecanismos, sendo que novas soluções podem ser adicionadas diretamente no código-fonte, de acordo com um *template* de código pré-definido. As métricas implementadas baseiam-se nos trabalho de Herbst et al. (2015,2016) e fornecem informações sobre a precisão do mecanismo para determinado perfil e sobre os intervalos dispendidos em períodos de subprovisionamento e superprovisionamento. Mais detalhes sobre a metodologia são apresentados na Seção 4.2.

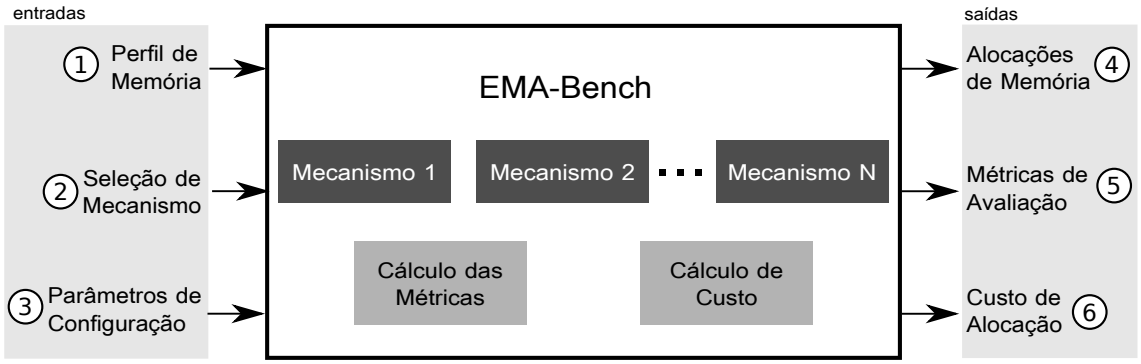


Figura 1. EMA-Bench: estrutura, entradas e resultados fornecidos.

4.1. Mecanismos Elasticidade Vertical de Memória

Três soluções reativas de alocação elástica disponíveis na literatura estão implementadas no EMA-Bench. A primeira solução é a proposta por Heo et al. (2009), na qual a próxima alocação de memória $u_{mem}(k+1)$ é calculada baseando-se na última alocação de memória $u_{mem}(k)$ e na memória consumida $v_{mem}(k)$. Nesse mecanismo há ainda dois parâmetros configuráveis, λ que é a constante de incremento que determina a agressividade do atuador e u_{ref}^{mem} que estabelece a quantidade desejável de memória ocupada na VM.

O cálculo da quantidade de memória a ser alocada no instante $k+1$ é dado por:

$$u_{mem}(k+1) = u_{mem}(k) + \lambda \times v_{mem}(k) \times (u_{ref}^{mem} - r_{mem}(k))/u_{ref}^{mem} \quad (1)$$

onde,

$$r_{mem} = \frac{v_{mem}(k)}{u_{mem}(k)} \quad (2)$$

A segunda solução é descrita por Moltó et al. (2013), na qual os autores apresentam um mecanismo que realiza o monitoramento do uso de memória da VM e caso o percentual de memória livre esteja fora de determinada faixa, pode-se alocar ou liberar memória. As regras de elasticidade são aplicadas quando a porcentagem de memória livre no instante k for menor que 80% ou maior que 120% do *Memory Overprovisioning Percentage* (MOP), que corresponde à porcentagem de memória que deseja-se deixar disponível. Por exemplo, se o usuário escolher um MOP = 10%, o mecanismo de alocação será acionado quando a memória livre da VM for menor ou maior do que 10% do total de memória disponível. Assim, se a quantidade de memória livre for inferior a 8% o mecanismo aloca mais memória. Se superior a 12%, o mecanismo libera memória.

A quantidade de memória a ser alocada no tempo $k+1$ é dada por:

$$u_{mem}(k+1) = u_{mem}(k) \times (1 + MOP) \quad (3)$$

O terceiro mecanismo implementado é descrito no trabalho de Jenitha e Veeramani (2014), que calcula a alocação de memória para o instante $k+1$ usando uma equação baseada em médias móveis exponenciais ponderadas por meio da equação:

$$u_{mem}(k+1) = \alpha \times u_{mem}(k) + (1-\alpha) \times v_{mem}(k) \quad (4)$$

onde α é obtido através de:

$$\alpha = \frac{v_{mem}(k)}{u_{mem}(k)} \quad (5)$$

4.2. Métricas Implementadas

O EMA-Bench utiliza o conjunto de métricas propostas por Herbst et al. (2016), uma vez que se mostra apropriado para a avaliação da alocação elástica de memória. Nesta proposta dois aspectos principais são avaliados: precisão (*accuracy*) e o tempo dispendido nos estados de sub e superprovisionamento. A precisão do dimensionamento de recursos é calculada em relação à quantidade de recursos provisionados e a quantidade de recursos realmente utilizados. O tempo de provisionamento está relacionado à proporção de tempo em cada estado.

Dado o resultado referente às alocações fornecido ao final da execução de um mecanismo de elasticidade, calcula-se:

$\sum A$ - tempo acumulado no estado de subprovisionamento;
$\sum U$ - quantidade acumulada de recursos faltantes;
$\sum B$ - tempo acumulado no estado de superprovisionamento;
$\sum O$ - quantidade acumulada de recursos excedentes.

Os valores para $\sum A$, $\sum U$, $\sum B$ e $\sum O$ são obtidos a partir da comparação do valor de demanda no tempo k com o total de memória alocada pelo método no tempo k. Se o total alocado é menor que o demandado, indicando subprovisionamento, incrementa-se $\sum U$ pelo valor absoluto da diferença entre o alocado e demandado e $\sum A$ é incrementado em 1. Se o total alocado é maior que o demandado, indicando superprovisionamento, incrementa-se $\sum O$ pelo valor absoluto da diferença entre o alocado e demandado e $\sum B$ é incrementado em 1.

A partir desses valores é possível definir a precisão, calculada através da média dos desvios absolutos entre os recursos alocados e suas respectivas demandas reais de recursos, normalizadas pelo tempo de avaliação T. Neste caso define-se duas métricas: P_u que corresponde à precisão de alocação e P_d correspondente à precisão de desalocação.

$$P_u = \frac{\sum U}{T} \qquad P_d = \frac{\sum O}{T} \tag{6}$$

Similarmente, pode-se calcular o tempo de provisionamento somando a quantidade total de tempo gasto em um estado de subprovisionamento ($\sum A$) ou superprovisionamento ($\sum B$) normalizado pela duração do período de medição T. Assim, o tempo total gasto em estados sub ou superprovisionados é dado, respectivamente, por:

$$Q_u = \frac{\sum A}{T} \qquad Q_d = \frac{\sum B}{T} \tag{7}$$

Usando as métricas propostas para avaliar os mecanismos de alocação elástica de memória, espera-se que Q_d apresente valores próximo de 1 (consequentemente com Q_u próximo de zero), e P_d apresente valores pequenos, indicando que não há, ou há poucos, pontos de subprovisionamento e que não há alocação excessiva de recursos.

4.3. Parâmetros de Entrada e Saídas

Conforme apresentado na Seção 4 e ilustrado na Figura 1, o EMA-Bench possui um conjunto de entradas e saídas bem definidos. Os parâmetros de entrada são fornecidos por meio de linha de comando na forma:

```
ema-bench perfil_memoria mecanismo [parâmetros]
```

O parâmetro `perfil_memoria` refere-se ao nome do arquivo que contém o perfil de memória obtido a partir de um mecanismo de monitoramento. O arquivo, de texto e sem formatação, consiste de n linhas contendo valores inteiros, cada um associado ao uso de memória no instante k da coleta. O parâmetro `mecanismo` é utilizado para a escolha de uma solução para a avaliação. As atuais opções são: `heo`, `molto` e `jenitha`, correspondendo aos três mecanismos implementados até o momento. Dependendo do mecanismo selecionado tem-se um conjunto de parâmetros de configuração próprios. Por exemplo, optando pelo mecanismo `heo` os parâmetros λ e u_{ref}^{mem} devem ser informados; o parâmetro MOP deve ser informado para `molto`; e nenhum parâmetro adicional é necessário para `jenitha`.

Após a execução o EMA-Bench retorna um arquivo de saida (*output.dat*) com n linhas contendo um número inteiro que corresponde a alocação realizada pelo mecanismo escolhido para o instante k. O arquivo de alocações pode ser usado em uma ferramenta de plotagem, permitindo a visualização gráfica dos resultados. Além disso, são retornadas em tela duas tabelas contendo os valores para as métricas e com os cálculos de custo.

Na tabela de cálculo de custos, apresenta-se um comparativo entre os custos de alocação estática e elástica. Para o cálculo no caso da alocação estática, considera-se a quantidade de memória necessária para satisfazer o maior pico de demanda para todo o período avaliado multiplicado pelo custo por unidade de tempo (dólar por MB/min., por exemplo). Na contabilização dos custos da alocação elástica, considerou-se o custo por unidade de tempo multiplicado pelo montante alocado a cada instante k.

5. Avaliação Experimental

Nessa seção apresenta-se um conjunto de experimentos conduzidos com o *benchmark* proposto. Para a realização dos testes foram utilizados 5 perfis de consumo de memória (nomeados de R1 a R5) obtidos de de máquinas virtuais hospedadas no *data center* de uma empresa de tecnologia localizada na cidade de Cascavel-PR. O monitoramento do consumo de memória foi realizado a cada minuto por aproximadamente 6 dias, totalizando 8.747 coletas para cada um dos 5 perfis.

Os perfis foram selecionados de modo a apresentar comportamentos bastante heterogêneos, fornecendo situações distintas para a avaliação dos mecanismos. O perfil R1 corresponde a um controlador de domínios do *data center*, R2 é um servidor de impressão, R3, R4 e R5 correspondem a servidores de aplicação. O hipervisor utilizado no *data center* para a criação das VMs é o Microsoft Hiper-V.

Ao todo foram realizados 25 testes utilizando o EMA-Bench, nos quais realizou-se cinco testes para cada um dos cinco perfis, com o intuito de se verificar se os métodos apresentados na literatura são apropriados para estes cenários. Utilizou-se nos testes um computador com processador Intel Core i5-2450M @ 2.50 GHz e 6 GB de memória RAM com Sistema Operacional Linux Ubuntu 14.04 LTS o compilador utilizado é o GCC 5.4.0.

Cada um dos perfis foi testado com o mecanismo proposto por Heo (2009) utilizando os parâmetros $u_{ref}^{mem} = 90\%$ e 70% e $\lambda = 1$, empregados no trabalho original. Os resultados obtidos para cada um dos perfis são apresentados graficamente na Figura 2, onde é possível verificar que a alocação de memória realizada satisfaz as demandas du-

rante todo o tempo para todos os perfis. Os valores apresentados nas Tabelas 1 e 2 para a métrica $Q_d = 1$, significando que em 100% do tempo houve alguma sobra de memória, atesta isso.

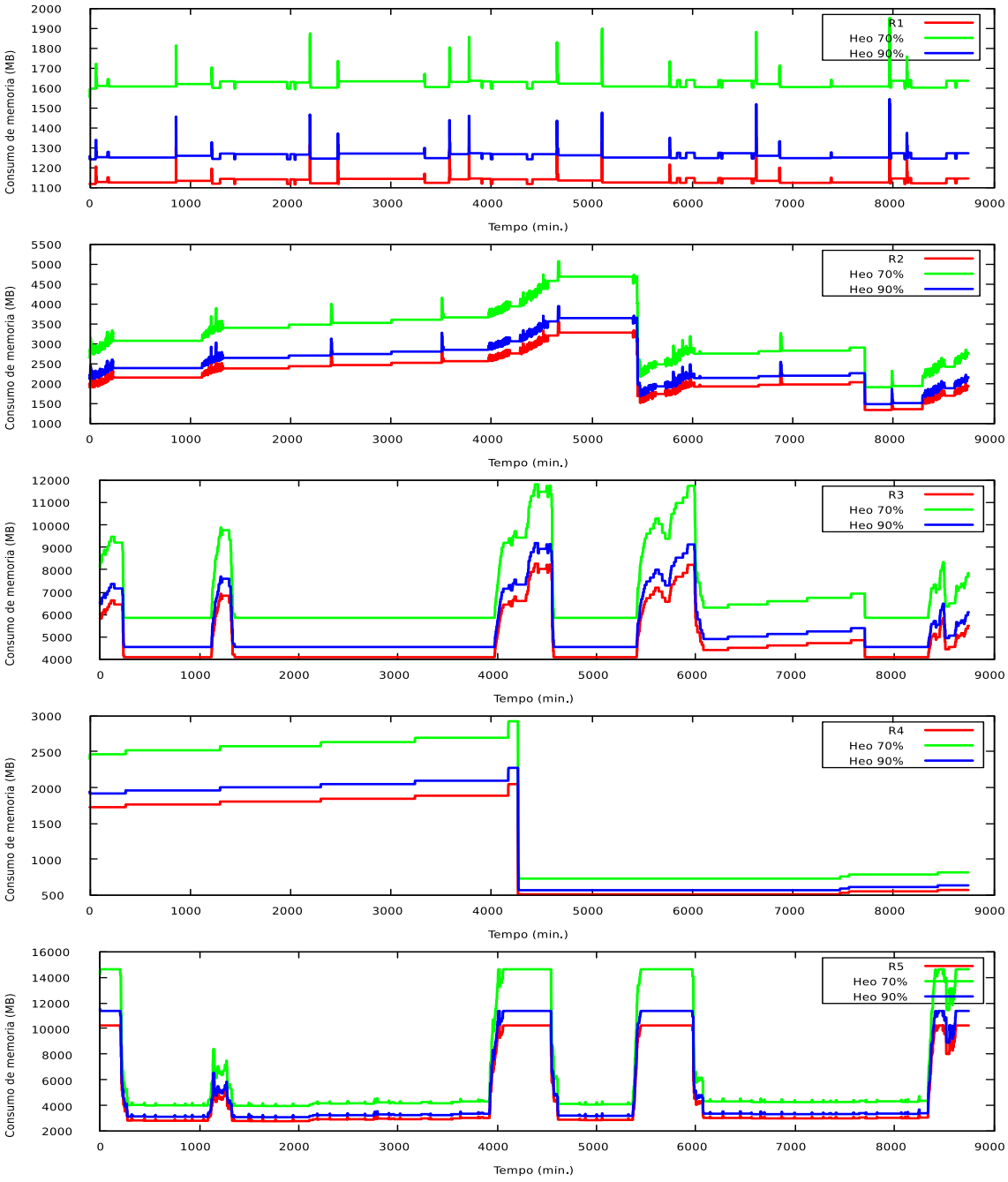


Figura 2. Heo: Alocações realizadas.

Considerando que a solução de Heo determina uma porcentagem alvo para o uso de memória (90 % e 70%, nos testes), é esperado que se tenha superprovisionamento em grande parte das execuções. Isso é evidenciado pela métrica $\sum O$ que mostra o somatório das quantias de memória superprovisionadas ao longo do período considerado. Como esperado, os resultados com $u_{ref}^{mem} = 90\%$ apresentaram valores inferiores para $\sum O$ quando comparados com os testes empregando $u_{ref}^{mem} = 70\%$.

Tabela 1. Heo 90%: Resultado das métricas.

Perfil	$\sum A$	$\sum U$	$\sum B$	$\sum O$	P_u	P_d	Q_u	Q_d
R1	0	0	8.747	1.099.400	0,0000	125,6744	0,0000	1
R2	0	0	8.747	2.216.153	0,0000	253,3325	0,0000	1
R3	0	0	8.747	4.615.865	0,0000	527,6480	0,0000	1
R4	0	0	8.747	1.121.759	0,0000	128,2303	0,0000	1
R5	0	0	8.747	4.323.920	0,0000	494,2753	0,0000	1

Tabela 2. Heo 70%: Resultado das métricas.

Perfil	$\sum A$	$\sum U$	$\sum B$	$\sum O$	P_u	P_d	Q_u	Q_d
R1	0	0	8.747	4.249.098	0,0000	485,7222	0,0000	1
R2	0	0	8.747	8.544.932	0,0000	976,7869	0,0000	1
R3	0	0	8.747	17.801.887	0,0000	2.034,9667	0,0000	1
R4	0	0	8.747	4.337.233	0,0000	495,7971	0,0000	1
R5	0	0	8.747	16.664.225	0,0000	1.904,9182	0,0000	1

Assim, de modo geral, a solução Heo consegue fornecer uma alocação elástica de memória satisfatória e dada sua configurabilidade, pode se adaptar com sucesso a diferentes perfis.

Os experimentos com a solução proposta por de Moltó et al. (2013) foram realizados com o parâmetro MOP assumindo valores 10% e 30%, assim como apresentado no trabalho original. Assim, no primeiro caso espera-se ter aproximadamente 10% memória livre e 30% no segundo. As alocações realizadas usando esta abordagem podem ser observados na Figura 3 e os resultados das métricas são apresentados nas Tabelas 3 e 4.

Tabela 3. Moltó 10%: Resultado das métricas.

Perfil	$\sum A$	$\sum U$	$\sum B$	$\sum O$	P_u	P_d	Q_u	Q_d
R1	7	467	8.740	989.004	0,0534	113,0549	0,0008	0,9991
R2	10	404	8.737	1.990.936	0,0462	227,5876	0,0011	0,9987
R3	0	0	8.747	4.150.330	0,0000	474,4319	0,0000	1
R4	0	0	8.747	883.272	0,0000	100,9685	0,0000	1
R5	7	741	8.740	3.870.026	0,0847	442,3898	0,0008	0,9991

Tabela 4. Moltó 30%: Resultado das métricas.

Perfil	$\sum A$	$\sum U$	$\sum B$	$\sum O$	P_u	P_d	Q_u	Q_d
R1	0	0	8.747	2.972.113	0,0000	339,7477	0,0000	1
R2	0	0	8.747	5.977.232	0,0000	683,2684	0,0000	1
R3	0	0	8.747	12.455.647	0,0000	1.423,8280	0,0000	1
R4	0	0	8.747	3.030.940	0,0000	346,4724	0,0000	1
R5	0	0	8.747	11.615.460	0,0000	1.327,7847	0,0000	1

Diferentemente dos resultados apresentados pela solução de Heo, o mecanismo de Moltó não obteve resultados totalmente positivos para todos testes. Pode-se observar que para $MOP = 10\%$ há períodos de subprovisionamento para os perfis R1, R2 e R5, mesmo que pouco significativos e breves, como mostram as métricas apresentadas na Tabela 3. O subprovisionamento ocorre nos momentos onde as demandas de memória aumentam

rapidamente e como há pouca memória excedente (cerca de 10%), o mecanismo apresenta um intervalo de tempo entre a detecção da demanda e a alocação efetiva da memória demandada. O mesmo não ocorre com $MOP = 30\%$, pois a quantidade extra de memória alocada mascara o atraso na alocação. Isso fica claro ao comparar a métrica P_d, que mostra a quantidade média de memória excedente, dos testes empregando diferentes valores para MOP.

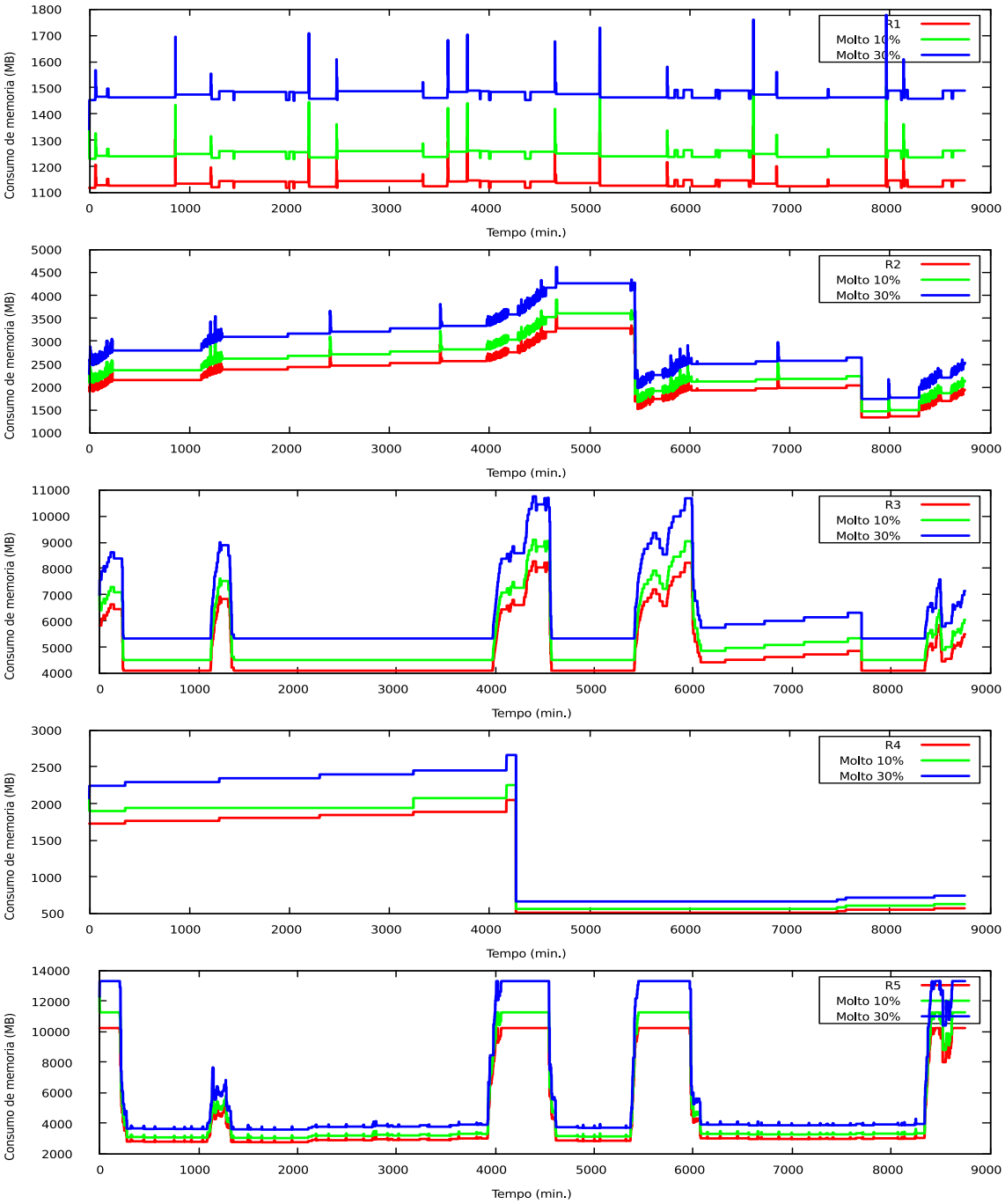


Figura 3. Moltó: Alocações realizadas.

O mecanismo proposto por Jenitha e Veeramani possui um comportamento diverso dos apresentados pelos demais. Por não prever alocação de memória livre excedente, a solução oferece uma alocação muito próxima da demanda, como pode-se observar na Figura 4 e na Tabela 5. Com isso, há períodos de subprovisionameto para todos os perfis ($\sum A \neq 0$), embora a métrica P_u apresente valores baixos. Destaca-se o comportamento desta solução para o perfil R4, onde em 35% $(1 - (Q_u + Q_d))$ do tempo do experimento a quantia de memória foi exatamente o demandado pela aplicação.

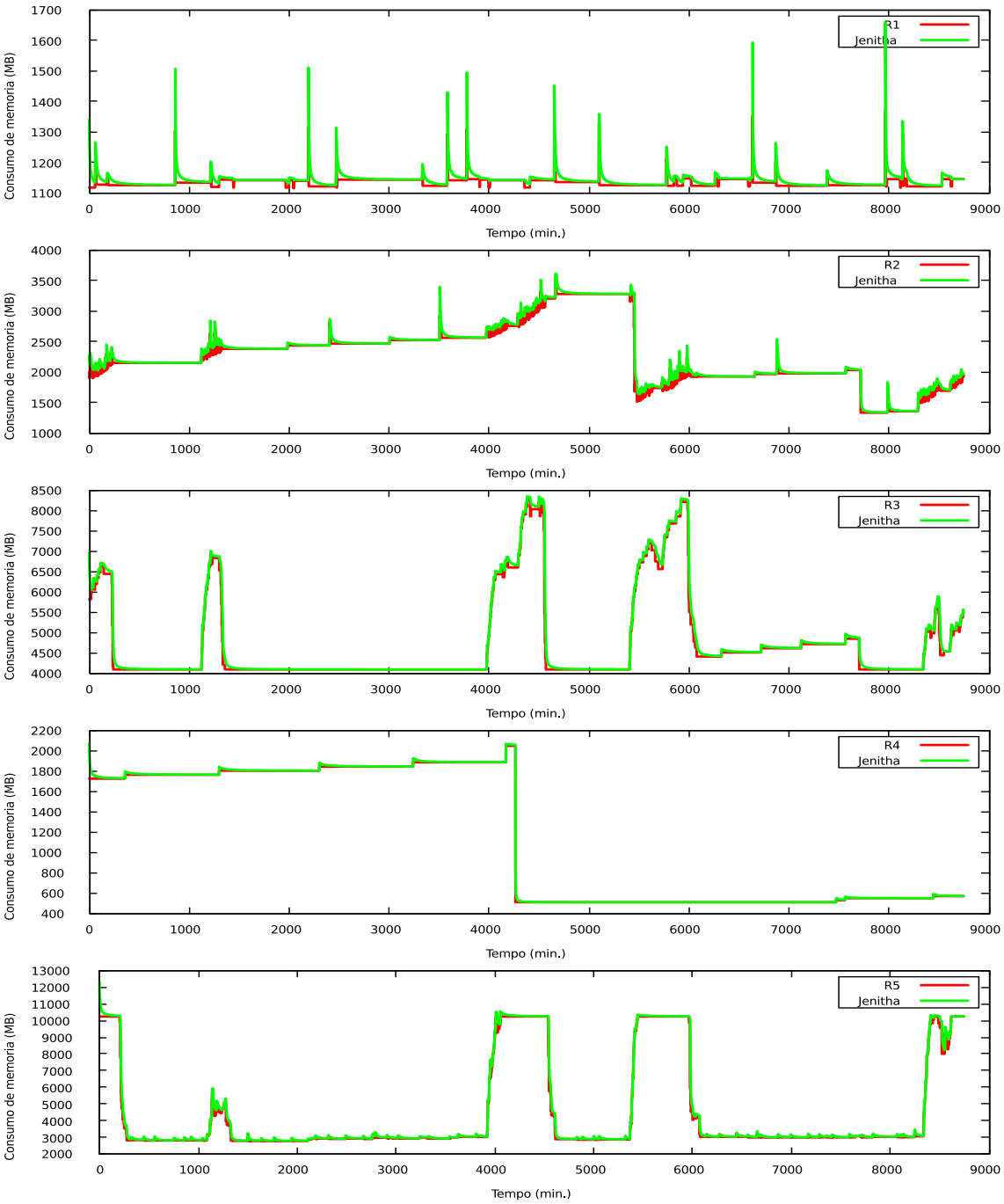


Figura 4. Jenitha: Alocações realizadas.

Tabela 5. Jenitha e Veeramani: Resultado das métricas.

Perfil	$\sum A$	$\sum U$	$\sum B$	$\sum O$	P_u	P_d	Q_u	Q_d
R1	31	1.936	8.597	92.804	0,2213	10,6086	0,0035	0,9827
R2	150	6.584	8.592	266.141	0,7526	30,4231	0,0171	0,9822
R3	119	9.050	8.628	407.279	1,0345	46,5568	0,0136	0,9863
R4	8	298	5678	3.6974	0,0341	4,2266	0,0009	0,6491
R5	163	19.949	8.584	694.881	2,2804	79,4331	0,0186	0,9813

Considerando que sob a óptica do cliente, a principal vantagem da alocação elástica é a manutenção do SLA com redução de custos, apresenta-se na Tabela 6 resultados de custo total para cada um dos perfis apresentados utilizando alocação estática e elástica.

Os valores foram calculados tomando como base os valores cobrados pelo provedor Profitbricks[1], que é de US$ 0,0070 por GB/hora ou US$ 0,000117 GB/minuto. É possível notar na tabela que há apenas dois casos em que o valor total é superior ao alcançado pela alocação estática. Ambos os casos ocorrem para o perfil R1 com mecanismos que preservam o nível de uso de memória em aproximadamente 70% (Heo 70% e Moltó 30%), já que em ambos os casos é previsto superprovisionamento de recursos. Nos demais casos, pode-se observar reduções bastante significativas nos custos, em alguns casos superior a 50%, o que mostra que o uso da elasticidade vertical de memória pode ser realmente efetiva.

Tabela 6. Comparativo de custos: alocação estática versus alocação elástica. Valores em Dólares Americanos.

Perfil	Estática	Heo 90%	Heo 70%	Moltó 10%	Moltó 30%	Jenitha
R1	1,37	1,26 (92%)	*1,61 (118%)*	1,24 (90%)	*1,47 (107%)*	1,14 (83%)
R2	3,55	2,52 (71%)	3,24 (91%)	2,50 (70%)	2,95 (83%)	2,30 (65%)
R3	8,25	5,26 (64%)	6,76 (82%)	5,21 (63%)	6,15 (75%)	4,78 (58%)
R4	2,04	1,28 (63%)	1,65 (81%)	1,25 (61%)	1,50 (74%)	1,16 (57%)
R5	10,22	4,92 (48%)	6,33 (62%)	4,87 (48%)	5,75 (56%)	4,50 (44%)

6. Conclusão

Considerando a falta de métricas e metodologias para a avaliação de mecanismos de elasticidade vertical de memória, a contribuição deste trabalho foi a proposição de um *benchmark* para este fim. Perante os trabalhos relacionados, o diferencial da proposta apresentada foi implementar um conjunto de métricas para a avaliação da eficácia e eficiência dos mecanismos, bem como informações sobre os custo envolvidos na adoção de uma determinada solução.

Os resultados mostram que a ferramenta proposta é capaz de auxiliar o usuário na definição de qual é o melhor mecanismo a ser adotado, garantindo redução de custos e a manutenção da qualidade do serviço. Além disso, o EMA-Bench pode ser utilizado por outros pesquisadores na comparação e refinamento de experimentos e de soluções elásticas.

[1] `https://www.profitbricks.com/`

Os trabalhos futuros incluem a implementação de novos mecanismos de elasticidade de memória, incluindo mecanismos preditivos, que por terem implementação mais complexa ficaram fora do escopo deste trabalho. Pretende-se ainda melhorar a qualidade das informações retornadas ao usuário por meio da geração automática de gráficos de alocação, bem como de relatórios mais completos. Está previsto também a amplição do escopo do trabalho, adicionando também ao *benchmark* a possibilidade de avaliar a elasticidade vertical para outros tipos de recurso, tal como CPU e armazenamento.

Agradecimentos

Os autores agradecem à empresa Constel Tecnologia por fornecer os perfis de utilização de memória dos servidores virtualizados e ao CNPq pelo financiamento parcial desta pesquisa.

Referências

Beltran, M. (2016). Defining an elasticity metric for cloud computing environments. In *Proceedings of the 9th EAI International Conference on Performance Evaluation Methodologies and Tools*, VALUETOOLS'15, pages 172–179. ICST.

Ben-Yehuda, A. O., Ben-Yehuda, M., Schuster, A., and Tsafrir, D. (2012). The resource-as-a-service (raas) cloud. In *Proceedings of the 4th USENIX conference on Hot Topics in Cloud Computing*, HotCloud'12, pages 1–5. USENIX.

Coutinho, E. F., Rego, P. A. L., Gomes, D. G., and de Souza, J. N. (2014). Métricas para avaliação da elasticidade em computação em nuvem baseadas em conceitos da física. In *Anais do XII Workshop de Computação em Clouds e Aplicações*, WCGA 2014, pages 55–66. SBC.

Farokhi, S., Jamshidi, P., Bayuh Lakew, E., Brandic, I., and Elmroth, E. (2016). A hybrid cloud controller for vertical memory elasticity. *Future Gener. Comput. Syst.*, 65(C):57–72.

Galante, G. and Bona, L. C. E. (2012). A survey on cloud computing elasticity. In *Proceedings of the International Workshop on Clouds and eScience Applications Management*, CloudAM'12, pages 263–270. IEEE.

Galante, G., Erpen De Bona, L. C., Mury, A. R., Schulze, B., and da Rosa Righi, R. (2016). An analysis of public clouds elasticity in the execution of scientific applications: a survey. *Journal of Grid Computing*, 14(2):193–216.

Gong, Z., Gu, X., and Wilkes, J. (2010). Press: Predictive elastic resource scaling for cloud systems. In *Proceedings of the 6th International Conference on Network and Service Management*, CNSM'10, pages 9–16. IEEE.

Heo, J., Zhu, X., Padala, P., and Wang, Z. (2009). Memory overbooking and dynamic control of xen virtual machines in consolidated environments. In *Proceedings of the IFIP/IEEE International Symposium on Integrated Network Management*, pages 630–637. IEEE.

Herbst, N., Krebs, R., Oikonomou, G., Kousiouris, G., Evangelinou, A., Iosup, A., and Kounev, S. (2016). Ready for rain? A view from SPEC research on the future of cloud metrics. Technical Report SPEC-RG-2016-01, SPEC Research Group - Cloud Working Group.

Herbst, N. R., Kounev, S., Weber, A., and Groenda, H. (2015). Bungee: An elasticity benchmark for self-adaptive iaas cloud environments. In *Proceedings of the 10th International Symposium on Software Engineering for Adaptive and Self-Managing Systems*, SEAMS '15, pages 46–56. IEEE.

Hwang, K., Bai, X., Shi, Y., Li, M., Chen, W., and Wu, Y. (2016). Cloud performance modeling with benchmark evaluation of elastic scaling strategies. *IEEE Trans. Parallel Distrib. Syst.*, 27(1):130–143.

Islam, S., Lee, K., Fekete, A., and Liu, A. (2012). How a consumer can measure elasticity for cloud platforms. In *Proceedings of the 3rd ACM/SPEC International Conference on Performance Engineering*, pages 85–96. ACM.

Jenitha, V. H. A. and Veeramani, R. (2014). Dynamic memory allocation using ballooning and virtualization in cloud computing. *IOSR Journal of Computer Engineering*, 16(2):19–23.

Li, K. (2017). Quantitative modeling and analytical calculation of elasticity in cloud computing. *IEEE Transactions on Cloud Computing*, PP(99):1–14.

Moltó, G., Caballer, M., Romero, E., and de Alfonso, C. (2013). Elastic memory management of virtualized infrastructures for applications with dynamic memory requirements. *Procedia Computer Science*, 18:159–168.

Spinner, S., Herbst, N., Kounev, S., Zhu, X., Lu, L., Uysal, M., and Griffith, R. (2015). Proactive memory scaling of virtualized applications. In *Proceedings of the 8th International Conference on Cloud Computing*, pages 277–284. IEEE.

Tan, Y., Nguyen, H., Shen, Z., Gu, X., Venkatramani, C., and Rajan, D. (2012). Prepare: Predictive performance anomaly prevention for virtualized cloud systems. In *Proceedings of the 32nd International Conference on Distributed Computing Systems*, ICDCS, pages 285–294. IEEE.

Zhang, W.-Z., Xie, H.-C., and Hsu, C.-H. (2017). Automatic memory control of multiple virtual machines on a consolidated server. *IEEE Transactions on Cloud Computing*, 5(1):2–14.

Um Modelo de Estimação para Provisionamento de Rede de Máquinas Virtuais em Nuvens IaaS

Jonatas A. Marques[1], **Rafael R. Obelheiro**[1]

[1] Programa de Pós-Graduação em Computação Aplicada
Universidade do Estado de Santa Catarina, Joinville, Brasil

jonatas.marques@ufrgs.br, rafael.obelheiro@udesc.br

Abstract. *In Infrastructure-as-a-Service (IaaS) clouds, customers lease virtual resources (e.g., CPU, memory, network) offered by cloud providers, paying for the reserved capacity of resources, regardless of their effective use. In this context, it is in the interest of customers to reserve resources with sufficient capacity so that their applications achieve good performance whilst, at the same time, seeking to minimize their expenses with idle capacities. In this paper, we present a novel model for network bandwidth estimation to guide cloud users in the provisioning of their virtual machines. The model is application-agnostic and dynamically adapts to changes in the virtual machine's workload profile through automatic monitoring and classification of its network flows. Our experimental evaluation shows that the estimates provided by the model are useful for guiding the provisioning of virtual machines, providing accurate bandwidth estimates even when the network is currently under-provisioned.*

Resumo. *Em nuvens computacionais do tipo Infrastructure-as-a-Service (IaaS), clientes alugam recursos virtuais (processador, memória, rede) fornecidos por provedores de nuvem, pagando pela capacidade reservada, independente da efetiva utilização dos recursos. Nesse contexto, é de interesse dos clientes reservar recursos com capacidade suficiente para que suas aplicações atinjam um bom desempenho, buscando ao mesmo tempo minimizar gastos com capacidade ociosa. Neste trabalho, é proposto um novo modelo para estimação de largura de banda de rede adequada para provisionamento de máquinas virtuais em nuvens IaaS, considerando a perspectiva do cliente. O modelo é agnóstico de aplicação e adapta-se dinamicamente à evolução do perfil de carga da máquina virtual, sendo baseado na monitoração e classificação automática dos seus fluxos de rede. A avaliação experimental mostra que as estimativas fornecidas pelo modelo são úteis para orientar o provisionamento de máquinas virtuais, sendo capaz de gerar estimativas acuradas de largura de banda necessária mesmo em cenários onde a rede está subprovisionada.*

1. Introdução

A computação em nuvem é um modelo que permite o acesso a recursos computacionais configuráveis (como redes, servidores, armazenamento, aplicações e serviços) por seus clientes [Mell and Grance 2011], que pagam conforme a capacidade reservada dos recursos, independente da efetiva utilização destes [Suleiman et al. 2012]. Nesse contexto, deve-se ressaltar que (i) o cliente deve especificar a capacidade desejada para cada recurso

individualmente e (ii) em nuvens, capacidade em excesso induz um custo desnecessário com o aluguel de recursos ociosos (diferente de infraestruturas físicas onde o custo de aquisição de recursos é fixo), portanto o cliente deseja reservar a capacidade mínima que ainda proporcione um bom desempenho às suas aplicações.

De modo geral, em nuvens IaaS (*Infrastructure-as-a-Service*) a rede é compartilhada entre os clientes, sem garantia da largura de banda (LB) disponível para as máquinas virtuais. Essa ausência de garantia prejudica não apenas os clientes, pois o desempenho de muitas aplicações é sensível ao desempenho da rede e este se torna imprevisível [Ballani et al. 2011], mas também os provedores, que podem arcar com custos decorrentes de violações de acordos de nível de serviço (*service level agreements*, SLAs) e deixar de atrair clientes que demandem previsibilidade da rede [Mogul and Popa 2012]. Abstrações mais refinadas, tais como infraestruturas virtuais [Anhalt et al. 2010], permitem reservar LB para enlaces virtuais, mas exigem que o cliente especifique a capacidade desejada, o que não é uma tarefa trivial. Portanto, um mecanismo que seja capaz de estimar a LB adequada para enlaces virtuais seria útil tanto para os clientes, que poderiam procurar serviços com garantia da capacidade de rede, quanto provedores, pois facilitaria a oferta de serviço garantido com bom custo-benefício ao permitir uma utilização otimizada das infraestruturas físicas com mínimo risco de problemas de desempenho.

O dimensionamento de enlaces virtuais para ambientes de nuvem ainda é pouco explorado na literatura. Alguns trabalhos focam em mecanismos para oferecer garantias de LB em nuvens, com base em demandas especificadas pelos clientes [Popa et al. 2012, Anastasi et al. 2016, Marcon et al. 2016]. Em [Pfitscher et al. 2013] é proposto um modelo de diagnóstico de provisionamento de rede que identifica se a LB está subprovisionada, superprovisionada ou adequada, mas que, nos casos de sub- e superprovisionamento, não estima a LB adequada. Cicada [LaCurts et al. 2014] propõe um mecanismo para predição de LB entre pares de MVs considerando a demanda média ou de pico em um dado intervalo, mas não trata o tráfego entre a MV e *hosts* Internet e nem é capaz de medir acuradamente a LB necessária quando a rede está subprovisionada.

Visando a contornar essas limitações, este trabalho traz como contribuição uma nova abordagem para estimação da LB adequada para enlaces virtuais. As interfaces de rede de uma MV são monitoradas para observar o volume de dados transmitido por todas as aplicações, sem a necessidade de conhecê-las. Algoritmos de aprendizado de máquina são usados para identificar grupos de fluxos com demanda de rede semelhante, e o tráfego de rede é classificado segundo esses grupos para estimar a LB adequada para essa MV. Resultados experimentais mostram que essa abordagem fornece estimativas suficientemente acuradas mesmo quando a rede está subprovisionada.

O restante deste artigo está organizado como segue. A Seção 2 discute os trabalhos relacionados. A Seção 3 descreve o modelo proposto, e a Seção 4 apresenta sua avaliação experimental. Finalmente, a Seção 5 conclui o artigo.

2. Trabalhos Relacionados

Cicada [LaCurts et al. 2014] é um mecanismo para predição da LB de enlaces virtuais sob o modelo *pipe* [Mogul and Popa 2012], em que são especificadas as capacidades dos enlaces entre pares de MVs. A monitoração do tráfego se baseia na análise dos fluxos entre MVs em um *datacenter*, sendo explicitamente desconsiderado o tráfego entre MVs

e a rede externa (clientes na Internet). A matriz de tráfego é observada a intervalos regulares, e a LB prevista é dada por uma combinação linear de observações anteriores, com pesos que vão sendo ajustados automaticamente em função dos erros de previsão. O mecanismo pode prever tanto a LB média quanto a LB de pico durante um dado período. O trabalho adota a premissa de que a rede está suficientemente provisionada durante o período de monitoração, não tratando o caso em que ela está subprovisionada. Cicada também incorpora um algoritmo para mapeamento dos recursos virtuais na infraestrutura física. O presente trabalho tem um escopo mais estreito, concentrando-se no problema de estimação da LB adequada para uma dada MV, mas considera o tráfego externo e é capaz de fornecer estimativas acuradas mesmo quando a rede está subprovisionada. Adaptar nossa abordagem para o modelo *pipe* seria uma tarefa trivial, bastando separar os fluxos de acordo com os seus pares origem-destino.

Em [Pfitscher et al. 2013] é proposto um modelo de diagnóstico de provisionamento de rede capaz de identificar se a largura de banda (LB) está subprovisionada, superprovisionada ou adequada. Esse modelo é baseado na monitoração do consumo de banda e da fila das interfaces de rede de uma máquina virtual (MV). Apesar do modelo ser capaz de identificar o estado de provisionamento de LB, ele não estima a LB nos casos de sub- e superprovisionamento. O presente trabalho vale-se da monitoração de fluxos para estimar a LB necessária em uma MV. Essa estimativa pode ser comparada com a LB alocada para diagnosticar o provisionamento de rede.

Alguns trabalhos, como [Gong et al. 2010, Engelbrecht and van Greunen 2015, Nguyen et al. 2013, Moreira Jr and Obelheiro 2016], propuseram estratégias de previsão do nível de utilização de recursos. Esses trabalhos são baseados em algoritmos de regressão (e.g., auto-regressão, auto-correlação, ARMA, ARIMA, suavização exponencial) que consideram históricos de medições de utilização de recursos com o objetivo de prever valores futuros. Em contraste com o presente trabalho, esses trabalhos focam no provisionamento adequado de processador e memória. Além disso, nenhuma dessas estratégias consideram o impacto do subprovisionamento na medição da utilização dos recursos. O modelo proposto no presente trabalho poderia ser usado em conjunto com uma das estratégias propostas nesses trabalhos para melhorar a qualidade do histórico de consumo de LB de rede para previsão de necessidades futuras.

[Popa et al. 2012, Anastasi et al. 2016, Marcon et al. 2016] propuseram mecanismos e estratégias para oferecer garantias de LB em nuvens. Esses trabalhos assumem que os clientes de nuvens especificam a LB desejada no momento da reserva de recursos. Nossa proposta é ortogonal a esses trabalhos e os complementa auxiliando os clientes de nuvem na tarefa de determinar qual a LB adequada para que suas aplicações atinjam bom desempenho sem que a reserva incorra em custos supérfluos.

3. Modelo Proposto

O fluxo de execução do modelo proposto é mostrado na Figura 1, e suas etapas são detalhadas na sequência. O modelo produz uma estimativa da largura de banda necessária para a carga aplicada sobre uma máquina virtual em um dado período. Estimar a LB necessária é um problema particularmente difícil quando a rede está subprovisionada, pois não é possível observar diretamente uma demanda superior à capacidade alocada. Nosso modelo traz como contribuição uma nova abordagem que permite estimar indiretamente

o valor real da demanda quando a rede está subprovisionada.

As estimativas geradas pelo modelo não são, por si sós, previsões da LB necessária, mas em conjunto podem ser utilizadas por um algoritmo de previsão para esse fim. No presente trabalho, intui-se que a qualidade das estimativas de LB necessária passadas tem impacto significativo sobre a acurácia das previsões futuras. Usar as estimativas produzidas em uma sequência de períodos para obter previsões da LB necessária futura enquadra-se no escopo de outro trabalho em andamento.

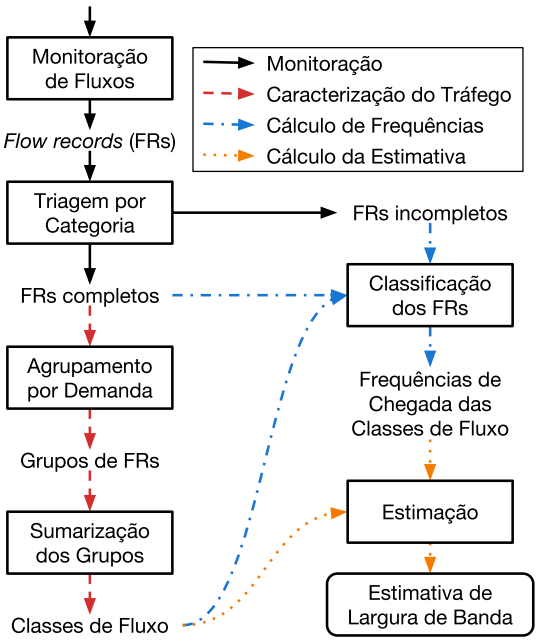


Figura 1. Fluxo de execução do modelo proposto.

3.1. Monitoração de Fluxos

O fluxo de execução do modelo proposto inicia pela monitoração de fluxos nas interfaces de rede da máquina virtual (MV) da qual deseja-se estimar a largura de banda (LB) necessária. Neste trabalho, essa monitoração é realizada usando o Argus[1]. Após um período de monitoração, o Argus fornece um relatório do tráfego que consiste em um conjunto de *flow records* (FRs), sendo registrado um FR para cada fluxo (bidirecional) observado. Um FR consiste em um registro que traz diversas informações sobre um fluxo. Alguns exemplos dessas são: os instantes inicial e final em que o fluxo foi observado; os endereços de origem e destino das aplicações comunicantes (i.e., endereço IP e porta); o protocolo de rede (e.g., UDP e TCP); as *flags* que foram observadas em pelo menos um dos pacotes do fluxo (e.g., SYN, ACK, FIN); e a quantidade de pacotes e bytes transmitidos em cada sentido do fluxo.

Realizada a monitoração de fluxos, pode-se então realizar a caracterização do tráfego das aplicações. Caracterizar o tráfego das aplicações consiste em identificar quais são as diferentes demandas de rede típicas dos fluxos das aplicações. A demanda de rede de um fluxo é expressa pelo par:

$$\text{Demanda} = \{\text{quantidade de bytes enviados}, \text{quantidade de bytes recebidos}\} \quad (1)$$

[1] http://www.qosient.com/argus/

Par esse formado pelos valores da quantidade de bytes transmitido em seus dois sentidos. A hipótese do modelo proposto é que conhecendo as demandas típicas dos fluxos e a frequência com que fluxos de cada demanda ocorrem na MV, pode-se estimar a LB necessária da rede para que minimizar a ocorrência de filas nas interfaces de rede da MV. Visto que a formação de filas causa atraso na transmissão (e eventualmente a perda) de pacotes, prejudicando o desempenho da aplicação.

3.2. Triagem por Categoria

Os *flow records* (FRs) gerados pela monitoração podem ser categorizados como completos ou incompletos. Um FR é dito completo quando traz informações referentes a um fluxo que iniciou e terminou dentro do intervalo de monitoração. Quando um fluxo ocorre (em parte) fora do intervalo de monitoração (i.e., iniciou antes e/ou terminou após o período de monitoração), o FR que traz informações sobre este fluxo é dito incompleto, pois durante o período de monitoração observou-se apenas parte de sua demanda de rede. Para a finalidade de caracterização do tráfego apenas os FRs completos são de interesse, visto que somente esses possuem informações totais sobre a demanda de rede dos fluxos.

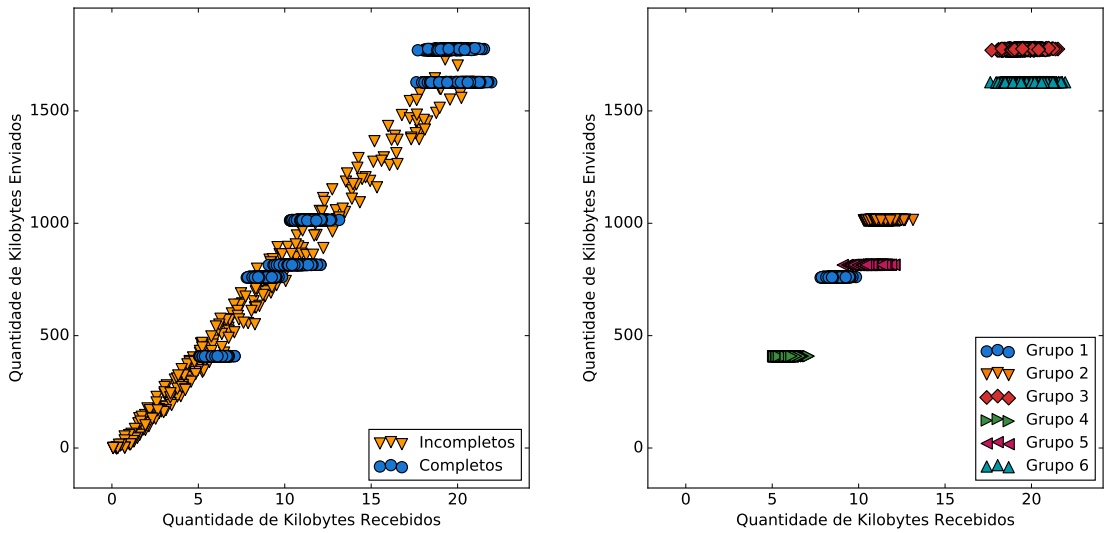
A etapa de triagem categoriza os FRs e os separa entre completos e incompletos. Os FRs de fluxos TCP são categorizados pelas *flags* observadas nos pacotes do fluxo. Caso as *flags* SYN (que indica o início de uma sessão TCP) e FIN (que indica o fim de uma sessão TCP) tenham sido usadas em alguns dos pacotes do fluxo, esse é categorizado como completo. Noutro caso, em que apenas uma ou nenhuma das duas *flags* seja observada, o fluxo TCP é categorizado como incompleto. Se a *flag* SYN não foi observada, conclui-se que o fluxo teve início anterior ao período de monitoração. Se a *flag* FIN não foi observada, conclui-se que o fluxo continuou além do período de monitoração ou foi abortado. Por sua vez, os FRs de fluxos UDP são todos categorizados como completos, visto que esse protocolo não apresenta sessões e todo datagrama transmitido representa uma transação/mensagem completa.

A Figura 2(a) apresenta a categorização do processo de triagem sobre um conjunto de FRs. Observa-se que os FRs completos formam grupos bem caracterizados, enquanto os incompletos estão espalhados ao longo da linha formada entre a origem do gráfico e os grupos de FRs completos. Em virtude desta observação, a fase de caracterização do tráfego considera apenas os FRs completos.

3.3. Agrupamento por Demanda

As aplicações voltadas ao atendimento de requisições (e.g., servidores de páginas web, bancos de dados) normalmente possuem um conjunto finito de requisições que são realizadas regularmente. A fase de caracterização do tráfego tem por objetivo identificar as demandas de rede dessas requisições observando unicamente os fluxos transitando nas interfaces de rede, sem a necessidade de entender o funcionamento das aplicações ou o que cada requisição representa para a aplicação. Essa metodologia permite ao modelo ser agnóstico às aplicações executando na MV.

Por experimentação, observou-se que requisições idênticas para uma mesma aplicação podem apresentar variações de quantidade de bytes recebidos e enviados (Figura 2(a)). Isso ocorre devido a fatores como o encapsulamento de mensagens de aplicação em segmentos TCP, perdas, reordenação de pacotes e ACKs atrasados (*delayed*



(a) *Flow records* (b) Grupos correspondentes

Figura 2. Fluxos observados e grupos identificados para uma carga de trabalho.

ACKs). Essas variações, no entanto, não têm magnitude suficiente para descaracterizar a demanda das requisições.

Para administrar essas variações de demanda para requisições idênticas, o modelo proposto aplica a técnica de *clustering* para identificar grupos de fluxos (que representam requisições) com demandas de rede próximas. A etapa de agrupamento busca identificar os grupos de fluxos com demandas de rede diferentes para cada uma das aplicações executando na MV. Essa etapa é realizada em duas partes. Primeiramente, os *flow records* (FRs) são agrupados quando possuem exatamente os mesmos endereços IP e portas de destino e protocolos de rede. No modelo proposto, cada tripla única $[\text{endereço IP de destino}, \text{porta de destino}, \text{protocolo de rede}]$ representa uma "aplicação" diferente. Por exemplo, uma aplicação servidora Apache[2] com um *virtual host* configurado para atender a porta 80 e outro para atender a porta 443 é considerado pelo modelo como duas aplicações diferentes, pois utiliza portas diferentes.

Após o agrupamento por aplicação, cada grupo de fluxos de aplicação é subdividido (por um algoritmo de *clustering*) em grupos de fluxos com demanda de rede semelhante. Dessa forma, para cada requisição de cada aplicação é identificado indiretamente um grupo de fluxos. O algoritmo de *clustering* escolhido para implementação do protótipo do modelo proposto foi o Mini-Batch K-Means, disponível através da biblioteca SciKit-Learn [Pedregosa et al. 2011]. Mini-Batch K-Means apresentou, quando comparado com outros cinco algoritmos, o melhor compromisso entre complexidade de ajuste, complexidade de tempo e representatividade de agrupamento. O número de grupos usado no algoritmo é selecionado automaticamente, de forma iterativa: o algoritmo é executado várias vezes com diferentes números de grupos, e o melhor agrupamento – aquele que minimiza a variância da distância intragrupo – é o escolhido. A Figura 2(b) mostra os grupos identificados automaticamente pela etapa de agrupamento dos FRs da Figura 2(a).

[2]https://httpd.apache.org/

Como comentado anteriormente, essa etapa considera apenas os *flow records* categorizados como completos. A consideração dos incompletos acarretaria na formação de grupos que não representam adequadamente as demandas totais típicas da aplicação. Considerar somente os *flow records* completos (durante o agrupamento) é importante principalmente em casos onde a rede está subprovisionada, pois nesses, o percentual de *flow records* incompletos dentre todos os observados costuma ser maior do que em casos de provisionamento adequado e superprovisionamento. Logo, a triagem proposta nesse trabalho é essencial para obter grupos de fluxos com demandas representativas e boa acurácia na estimação de LB.

3.4. Sumarização dos Grupos

Esta última etapa da fase de caracterização do tráfego tem por objetivo criar protótipos para os grupos de *flow records* (FRs) identificados pela etapa de agrupamento. Para cada grupo de FRs é criado um protótipo que o resume. Um protótipo, ou doravante chamado classe de fluxo, é composto de sete informações, sendo as três primeiras a tripla de identificação de uma aplicação: endereço IP de destino, porta de destino e protocolo de rede. As outras quatro informações expressam a demanda de rede típica da classe de fluxo, são estas: a quantidade média de bytes enviados em cada fluxo, a margem de erro da quantidade de bytes enviados em cada fluxo (com nível de confiança de 99%), a quantidade média de bytes recebidos em cada fluxo e a margem de erro da quantidade de bytes recebidos em cada fluxo (com nível de confiança de 99%).

A Tabela 1 exemplifica o resultado da etapa de sumarização e da fase de caracterização do tráfego sobre os grupos de FRs da Figura 2(b). A partir dessa tabela (e das classes por ela apresentada) conclui-se que existem duas aplicações em execução na MV sendo monitorada, uma operando na porta 8000 e outra na porta 9000. A primeira aplicação (porta 8000) atende a requisições com demandas de envio típicas de aproximadamente 761 KB, 1 MB e 1,8 MB. A outra aplicação (porta 9000) atende a requisições com demandas de envio típicas de aproximadamente 410 KB, 818 KB e 1,63 MB.

Tabela 1. Classes de Fluxo geradas pela etapa de sumarização dos grupos.

Classe	Endereço IP de Destino	Porta de Destino	Protocolo de Rede	Média Bytes Enviados ± Margem de Erro	Média Bytes Recebidos ± Margem de Erro
1	1.1.1.1	8000	TCP	761.070 ± 352	9.727 ± 213
2	1.1.1.1	8000	TCP	1.013.472 ± 600	12.040 ± 238
3	1.1.1.1	8000	TCP	1.775.569 ± 960	22.170 ± 562
4	1.1.1.1	9000	TCP	409.906 ± 167	7.330 ± 150
5	1.1.1.1	9000	TCP	817.707 ± 461	13.326 ± 410
6	1.1.1.1	9000	TCP	1.633.532 ± 404	25.436 ± 379

3.5. Classificação dos *Flow Records*

As etapas descritas nas subseções anteriores ajustam parcialmente o modelo através das classes de fluxo que caracterizam o tráfego das aplicações em execução na MV. Para estimar a LB necessária é essencial também identificar com que frequência fluxos de cada uma das classes iniciam nas interfaces de rede. Normalmente, a frequência de chegada de fluxos de cada classe é mais dinâmica do que a demanda de rede dessas, visto que é comum essa primeira variar ao longo de um dia (e.g., popularidade das páginas) enquanto a variação dessa segunda costuma estar atrelada a modificação da aplicação (e.g., criação de nova página em um servidor de páginas web). Para identificar a frequência de chegada

de cada classe de fluxo o modelo proposto realizada uma etapa de classificação sobre todos (completos e incompletos) os *flow records* (FRs) observados pela monitoração às classes.

Como comentado na Seção 3.2, os FRs fornecidos no relatório da monitoração de fluxos podem ser categorizados como completos ou incompletos. Para a finalidade de caracterização do tráfego, apenas os FRs considerados completos são observados, visto que somente estes contêm informações totais de demanda de rede. Já para a finalidade de identificar as frequências de chegada das classes de fluxo, ambas as categorias dos FRs devem ser consideradas (ainda que de formas diferentes). A classificação dos FRs completos associa cada um deles à classe de fluxo (de mesma aplicação, considerando os campos de identificação de aplicação) com demanda de rede mais próxima segundo a Distância Euclidiana, calculada através da Equação 2:

$$d_{i,a} = \sqrt{(R_a - R_i)^2 + (E_a - E_i)^2} \tag{2}$$

Sendo $d_{i,a}$ a distância entre um FR i e uma classe de fluxo a; R_a e E_a a quantidade média de bytes recebidos e enviados, respectivamente, da classe a; R_i e E_i a quantidade de bytes recebidos e enviados, respectivamente, pelo fluxo i.

A classificação dos FRs incompletos, por sua vez, não pode ser baseada na distância euclidiana entre as demandas de um fluxo e das classes, pois os FRs incompletos contêm uma demanda de rede parcial do fluxo (apenas a quantidade de bytes observada durante a monitoração). Por exemplo, é possível que um fluxo com demanda total igual a $[1\,\text{MB}, 10\,\text{KB}]$ seja observado apenas pela metade, resultando em um FR incompleto com demanda parcial de aproximadamente $[500\,\text{KB}, 5\,\text{KB}]$. Se o tráfego for caracterizado por três classes com demandas médias iguais a (a) $[100\,\text{KB}, 1\,\text{KB}]$, (b) $[600\,\text{KB}, 6\,\text{KB}]$ e (c) $[1\,\text{MB}, 10\,\text{KB}]$, não é possível determinar categoricamente que quando observado por completo o FR pertenceria a classe (b) ou (c). Entretanto, como a demanda parcial registrada no FR tem valor superior à demanda da classe (a), conclui-se que o fluxo completo não pertenceria a essa classe.

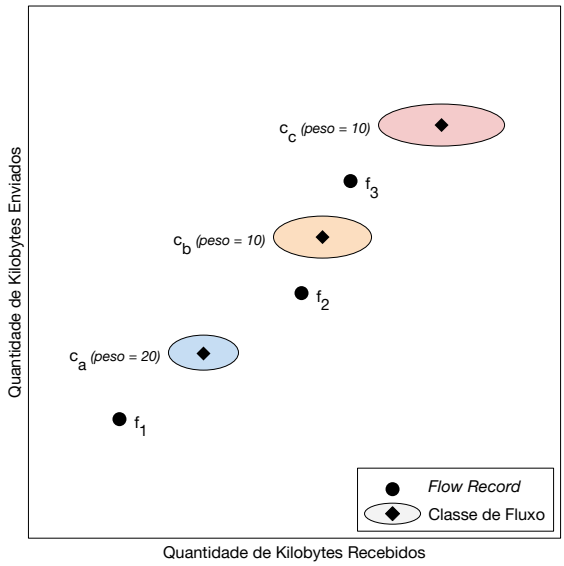


Figura 3. Exemplo de cenários de classificação de *flow records* incompletos.

Diante dessa problemática, o modelo proposto classifica FRs incompletos usando três cenários (otimista, pessimista e intermediário), e faz uma composição das três classificações para chegar à classificação final. No primeiro cenário, cada FR incompleto é classificado como pertencente à classe de mesma aplicação com demanda de rede imediatamente superior à demanda parcial do fluxo. Esse cenário otimista representa a LB mínima que seria necessária para transmitir completamente todos os fluxos observados parcialmente, pois cada fluxo é associado à classe de menor demanda dentre as classes com demanda superior à do fluxo. Nesse cenário os FRs incompletos f_1, f_2 e f_3 da Figura 3 seriam atribuídos às classes c_a, c_b e c_c, respectivamente.

O segundo cenário considerado classifica todos os FRs incompletos como pertencentes à classe de maior demanda da mesma aplicação. Esse cenário pessimista representa a LB máxima que seria necessária para transmitir completamente todos os fluxos observados parcialmente, pois todo fluxo é considerado como pertencente à classe de demanda de rede máxima da aplicação. Neste cenário todos os FRs incompletos da Figura 3 seriam atribuídos à classe c_c.

O terceiro (e último) cenário considerado associa parcelas dos *flow records* incompletos às classes. A parcela referente a cada classe é distribuída segundo uma ponderação feita sobre a frequência de chegada de fluxos completos de cada classe:

$$p_a = \frac{\lambda_a^c}{\sum_{i \in C_>} \lambda_i^c} \tag{3}$$

Sendo p_a o peso da classe a; λ_a^c a frequência de chegada de fluxos completos da classe a; $C_>$ o conjunto de todas as classes de fluxo com demanda superior à do fluxo sob classificação. Neste cenário, à classe c_a da Figura 3 seria atribuído $1/2$ do fluxo f_1; à classe c_b seria atribuído $1/4$ do fluxo f_1 e $1/2$ do fluxo f_2; e à classe c_c seria atribuído $1/4$ do fluxo f_1, $1/2$ do fluxo f_2 e todo o fluxo f_3. Esse cenário intermediário representa a LB que seria necessária para transmitir completamente todos os fluxos observados parcialmente caso a proporção entre as frequências de chegada observadas das classes fosse a mesma para os fluxos completos e para os incompletos.

Tabela 2. Frequências de chegada das classes de fluxo do exemplo.

Classe	*Flow records* completos (un.)	*Flow records* incompletos (un.)	Taxa de Chegada (fluxos/s)
1	436	28,85	3,1
2	719	17,94	4,91
3	697	51,88	4,99
4	1.318	70,13	9,25
5	419	25,15	2,96
6	1.206	132,72	8,92

As frequências de chegada das classes de fluxo considerando os FRs incompletos são calculadas pela média aritmética das frequências calculadas nos três cenários discutidos. Como comentado anteriormente na Seção 3.3, o percentual de FRs incompletos dentre todos os observados costuma ser significativo nos casos de subprovisionamento. Portanto, a classificação correta dos FRs incompletos (além dos completos) é fundamental para estimar a largura de banda necessária nos casos de subprovisionamento de rede. As frequências de chegada totais das classes de fluxo são calculadas somando as frequências

calculadas considerando os FRs completos e as calculadas considerando os incompletos. A Tabela 2 mostra as frequências de chegada de fluxos calculadas para as classes da Tabela 1 pela etapa de classificação dos fluxos.

3.6. Estimação

Após o ajuste completo do modelo proposto através dos processos (i) de caracterização do tráfego (ii) e de cálculo das frequências de chegada das classes de fluxo, estima-se a LB necessária para as aplicações em execução na MV segundo a Equação 4:

$$\mathrm{LB}_{\text{necessária}} = \max\Big(\sum_{i \in C} \lambda_i E_i; \sum_{i \in C} \lambda_i R_i\Big) \tag{4}$$

Sendo C o conjunto de todas as classes de fluxo; λ_i a frequência de chegada de fluxos da classe i; E_i a quantidade média de bytes enviados pela MV nos fluxos da classe i; e R_i a quantidade média de bytes recebidos pela MV nos fluxos da classe i. A LB necessária para atender as classes da Tabela 2, por exemplo, é estimada como 295,3 Mbps (máximo entre 295,3 Mbps de envio e 3,6 Mbps de recepção).

Para garantir o bom desempenho das aplicações, durante o provisionamento de rede da MV recomenda-se a adição de uma folga à LB necessária para tolerar oscilações naturais da taxa instantânea de transmissão de rede. Neste trabalho recomenda-se (após observação experimental) provisionar a rede de forma que a LB necessária use em média entre 75% e 85% da capacidade. Por exemplo, para a estimativa de 295,3 Mbps, a LB a ser provisionada está entre 347,4 Mbps (85%) e 393,7 Mbps (75%). Essa folga pode ser ajustada pelo cliente de nuvem de modo a privilegiar o custo (reduzindo a folga) ou o desempenho (aumentando a folga).

4. Avaliação

4.1. Variação da Popularidade de Requisições

Este primeiro experimento tem por objetivo avaliar a qualidade da fase de caracterização de demanda de rede do tráfego e a acurácia da etapa de classificação de fluxos. Para este propósito, um servidor de páginas web foi configurado para atender requisições a 25 páginas de tamanhos e conteúdos distintos.

Tabela 3. Tamanhos das páginas do servidor web.

Pág.	Tamanho	Pág.	Tamanho	Pág.	Tamanho	Pág.	Tamanho	Pág.	Tamanho
1	68 KB	6	582 KB	11	1,6 MB	16	2,4 MB	21	4,1 MB
2	162 KB	7	682 KB	12	1,6 MB	17	2,6 MB	22	5,9 MB
3	238 KB	8	1,2 MB	13	1,9 MB	18	2,9 MB	23	6,9 MB
4	245 KB	9	1,5 MB	14	2,0 MB	19	3,3 MB	24	7,9 MB
5	381 KB	10	1,5 MB	15	2,4 MB	20	3,5 MB	25	9,4 MB

A Tabela 3 apresenta o tamanho de cada uma das 25 páginas da aplicação. Três conjuntos de requisições de página (U, E, N) foram criados para gerar tráfego de rede (diferentes quanto a demanda total) na máquina onde o servidor foi configurado. Cada conjunto é composto de 2500 requisições, e cada uma dessas requisita uma das 25 páginas do servidor. A diferença entre cada conjunto de requisições está na distribuição de probabilidade do número de página requisitado:

- U: distribuição uniforme entre 1 e 25;
- E: distribuição exponencial $\mathrm{Exp}(25/3)$ (quanto menor o número de uma página, maior a sua popularidade);
- N: distribuição normal $N(25/2, 25/6)$ (páginas com números intermediários são mais populares que páginas com números pequenos/grandes).

A infraestrutura do experimento consistiu em duas máquinas virtuais (MVs) (servidora e cliente) provisionadas sobre o monitor de máquinas virtuais Xen versão 4.4. Cada MV foi provisionada com: 1 VCPU, 512 MB de RAM, 20 GB de espaço em disco, uma interface de rede com largura de banda (LB) de envio limitada a 80 Mbps. O servidor de páginas web foi implantado na MV servidora. Na MV cliente executou-se a ferramenta httperf[3] para requisitar as páginas à servidora segundo os conjuntos U, E e N.

Inicialmente foi caracterizado o tráfego para cada um dos conjuntos de requisições, e posteriormente as requisições foram classificadas usando as classes de fluxos obtidas. O objetivo foi o de avaliar a sensibilidade das estimativas a discrepâncias entre o tráfego usado para caracterização e o observado durante a classificação. A Tabela 4 apresenta a LB necessária estimada para cada par de conjunto de requisições; as linhas representam o conjunto usado para caracterização e as colunas o conjunto usado para classificação e estimação. Os valores de cada coluna devem se aproximar do valor encontrado utilizando o mesmo conjunto de requisições para caracterizar o tráfego, calcular as frequências e estimar a LB necessária. Observa-se que, mesmo quando o tráfego usado na classificação foi diferente do usado na caracterização, as estimativas tiveram boa exatidão: o erro médio (considerando apenas os conjuntos diferentes) foi de 0,62%, e o erro máximo foi de 2,18%. Conclui-se que as classes de fluxos obtidas do processo de agrupamento de *flow records* representam bem as diferentes demandas das requisições feitas ao servidor de páginas, mesmo variando a popularidade de cada requisição considerada no agrupamento dos *flow records*.

Tabela 4. Estimativas e erros para os pares de conjuntos de requisições.

conjunto usado	conjunto usado para classificação e estimativa		
para caracterização	U	E	N
U	17,09 Mbps	10,30 Mbps (2,18%)	22,37 Mbps (0,54%)
E	17,13 Mbps (0,23%)	10,08 Mbps	22,24 Mbps ($-$0,04%)
N	17,11 Mbps (0,12%)	10,14 Mbps (0,60%)	22,25 Mbps

4.2. Variação da Largura de Banda de Rede Reservada

Este experimento tem por objetivo avaliar a qualidade das estimativas dadas pelo modelo proposto quando a largura de banda (LB) reservada no período de monitoração varia entre valores subprovisionados, adequados e superprovisionados. Para este experimento foram criadas quatro cargas de trabalho, cada uma com páginas de tamanhos e taxas de requisição diferentes. As páginas das cargas foram implantadas em um servidor Apache com dois *virtual hosts*, um HTTP e outro HTTPS. A Tabela 5 apresenta o tamanho, a taxa de requisição e o protocolo de aplicação (HTTP ou HTTPS) de cada página de cada carga de trabalho. Realizou-se também experimentos com cargas de trabalho de outras aplicações – como FTP, SSH, e banco de dados SQL – mas seus resultados não são apresentados por serem semelhantes aos observados no presente experimento. Os tamanhos

[3]https://github.com/httperf/httperf

das páginas foram escolhidos aproximando-se de tamanhos de páginas dos websites mais populares da Internet[4]. As frequências de requisições de cada página foram escolhidas para explorar diferentes distribuições de popularidade (e.g., páginas possuem popularidade uniforme, páginas menores são mais populares).

Tabela 5. Cargas de trabalho para o segundo experimento.

Carga 1

Pág.	Tamanho	λ (req/s)	Protocolo
1	750 KB	3	HTTP
2	1 MB	5	HTTP
3	1,75 MB	5	HTTP
4	400 KB	9	HTTPS
5	800 KB	3	HTTPS
6	1,6 MB	9	HTTPS

Carga 2

Pág.	Tamanho	λ (req/s)	Protocolo
1	100 KB	25	HTTP
2	1 MB	7	HTTP
3	4 MB	3	HTTP
4	500 KB	10	HTTPS
5	1,5 MB	5	HTTPS
6	2 MB	5	HTTPS

Carga 3

Pág.	Tamanho	λ (req/s)	Protocolo
1	100 KB	20	HTTP
2	200 KB	25	HTTP
3	500 KB	20	HTTP
4	400 KB	25	HTTPS
5	500 KB	15	HTTPS
6	600 KB	15	HTTPS

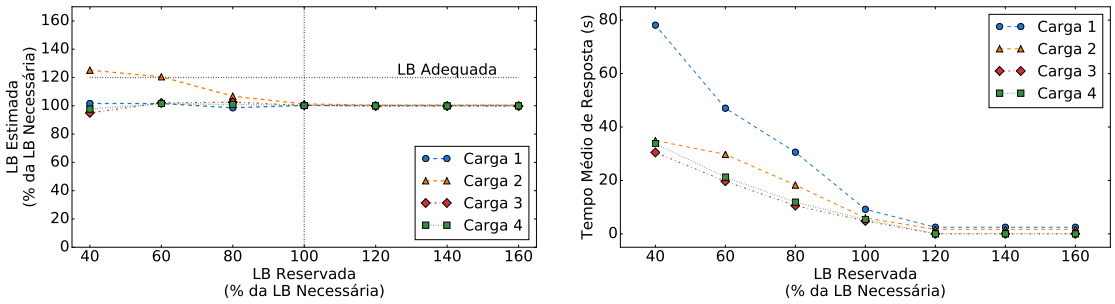
Carga 4

Pág.	Tamanho	λ (req/s)	Protocolo
1	100 KB	20	HTTP
2	200 KB	20	HTTP
3	500 KB	15	HTTP
4	400 KB	20	HTTPS
5	500 KB	15	HTTPS
6	600 KB	15	HTTPS

A infraestrutura do experimento consistiu em duas máquinas físicas (hospedeira e cliente) e uma MV (servidora). Cada máquina física possui 2 processadores Intel Xeon E5520 (4 cores por processador), 24 GB de RAM, 64 GB de armazenamento SSD e uma interface de rede Gigabit Ethernet. Ambas estão configuradas com o sistema operacional Debian 7, e adicionalmente, a máquina hospedeira executa o monitor de máquinas virtuais Xen versão 4.1. A MV servidora é virtualizada na hospedeira com 4 VCPUS, 4 GB de RAM, 4 GB de armazenamento SSD e LB da interface de rede variando de acordo com o cenário de teste do experimento. A máquina cliente faz as requisições das cargas de trabalho (pela ferramenta httperf) ao servidor Apache implantado na MV servidora.

Para cada carga de trabalho o modelo proposto foi executado completamente para sete cenários de reserva de LB, entre 40% e 160% da LB necessária (para determinar a LB necessária, a carga foi executada uma vez sem restrição da LB, ou seja, com 1 Gbps disponível). O tempo médio de execução do modelo foi de 7 segundos. Essa variação da LB reservada busca reproduzir os estados de subprovisionamento, provisionamento adequado e superprovisionamento e avaliar a qualidade das estimativas fornecidas pelo modelo proposto. As Figuras 4(a) e 4(b) apresentam as estimativas do modelo e o tempo médio de resposta para cada carga de trabalho em cada cenário de reserva de LB. Cada curva do gráfico representa os valores para uma carga de trabalho. Cada observação indica a LB estimada pelo modelo proposto ou o tempo de resposta (eixo y) quando executado sob determinada LB reservada (eixo x). O tempo de resposta reflete a duração de cada transação web, do início da requisição até o fim da resposta. Nos gráficos, os valores de LB são expressos como um percentual da LB necessária (quando a LB estimada ou reservada for igual à LB necessária esse percentual é de 100%).

[4]http://www.alexa.com/topsites



(a) LB estimada (b) Tempo de resposta

Figura 4. LB estimada e tempo de resposta em função da LB reservada.

Observa-se que as estimativas para os cenários onde a LB reservada é igual ou superior à LB necessária apresentam erros percentuais de pequena magnitude. Considerando o tempo médio de resposta, conclui-se que a LB mais adequada dentre as experimentadas está em 120% da LB necessária de cada carga, pois a partir deste ponto não há redução no tempo de resposta, mesmo com o aumento da capacidade. Constata-se também que os cenários de subprovisionamento (40%, 60% e 80%) resultam em erros maiores, mas ainda aceitáveis. Esse aumento era esperado visto que a rede fica saturada, o que por sua vez distorce o tráfego imprevisivelmente, favorecendo alguns tipos de fluxos em detrimento de outros. O maior erro observado, uma superestimação de aproximadamente 25%, ocorreu para a Carga 2, no cenário onde a LB reservada é igual a 40% da necessária. Mesmo no cenário mais desfavorável, o modelo consegue produzir uma estimativa útil para o provisionamento adequado da rede. Portanto, pode-se concluir que o modelo proposto é capaz de fornecer boas estimativas da LB necessária independentemente do nível atual de provisionamento da rede.

5. Conclusão

Neste trabalho foi proposto um novo modelo para estimação de largura de banda adequada para o provisionamento de máquinas virtuais em nuvens IaaS, considerando a perspectiva do cliente. O modelo é agnóstico a aplicação e baseia-se na monitoração de fluxos de pacotes nas interfaces de rede de uma máquina virtual. O processamento dos fluxos usando técnicas de aprendizado de máquina permite obter boas estimativas mesmo quando a rede está subprovisionada, um avanço em relação ao estado da arte. A avaliação experimental evidenciou que o modelo proposto é capaz de identificar acuradamente a demanda de rede com diferentes cargas de trabalho e de prover estimativas de largura de banda que são úteis para a tarefa de provisionamento de rede, independente da capacidade reservada durante sua execução. Em continuidade a este trabalho, está-se investigando a obtenção de previsões a partir de sequências de estimativas, e a incorporação do modelo proposto a um mecanismo de gerenciamento automático de escalabilidade.

Agradecimentos

Os autores gostariam de agradecer à CAPES pelo auxílio financeiro e ao Laboratório de Processamento Paralelo e Distribuído (LabP2D) pela disponibilização dos recursos computacionais para a realização do trabalho. Alguns experimentos apresentados neste trabalho foram realizados usando o *testbed* Grid'5000, mantido por um grupo de interesse científico hospedado por Inria e inclui CNRS, RENATER, várias universidades e outras organizações (vide https://www.grid5000.fr).

Referências

[Anastasi et al. 2016] Anastasi, G. F., Coppola, M., Dazzi, P., and Distefano, M. (2016). QoS guarantees for network bandwidth in private clouds. *Procedia Computer Science*, 97:4–13.

[Anhalt et al. 2010] Anhalt, F., Koslovski, G., and Primet, P. V.-B. (2010). Specifying and provisioning virtual infrastructures with HIPerNET. *Int. J. Network Management*, 20(3):129–148.

[Ballani et al. 2011] Ballani, H., Costa, P., Karagiannis, T., and Rowstron, A. (2011). The price is right: Towards location-independent costs in datacenters. In *10th ACM Workshop on Hot Topics in Networks (HotNets)*.

[Engelbrecht and van Greunen 2015] Engelbrecht, H. and van Greunen, M. (2015). Forecasting methods for cloud hosted resources, a comparison. In *Network and Service Management (CNSM), 2015 11th International Conference on*, pages 29–35. IEEE.

[Gong et al. 2010] Gong, Z., Gu, X., and Wilkes, J. (2010). PRESS: Predictive elastic resource scaling for cloud systems. In *2010 International Conference on Network and Service Management*, pages 9–16. IEEE.

[LaCurts et al. 2014] LaCurts, K., Mogul, J. C., Balakrishnan, H., and Turner, Y. (2014). Cicada: Introducing predictive guarantees for cloud networks. In *6th USENIX Workshop on Hot Topics in Cloud Computing (HotCloud 14)*.

[Marcon et al. 2016] Marcon, D. S., Neves, M. C., Oliveira, R. R., Gaspary, L. P., and Barcellos, M. P. (2016). PredCloud: Providing predictable network performance in large-scale OpenFlow-enabled cloud platforms through trust-based allocation of resources. *Computer Communications*, 91:44–61.

[Mell and Grance 2011] Mell, P. M. and Grance, T. (2011). The NIST definition of cloud computing. NIST SP 800-145, NIST, Gaithersburg, MD, United States.

[Mogul and Popa 2012] Mogul, J. C. and Popa, L. (2012). What we talk about when we talk about cloud network performance. *ACM Computer Communication Review*, 42(5):44–48.

[Moreira Jr and Obelheiro 2016] Moreira Jr, D. A. and Obelheiro, R. R. (2016). Previsão de demanda de recursos em nuvens IaaS usando séries temporais. In *VI Simpósio Brasileiro de Engenharia de Sistemas Computacionais (SBESC)*, João Pessoa, PB.

[Nguyen et al. 2013] Nguyen, H., Shen, Z., Gu, X., Subbiah, S., and Wilkes, J. (2013). Agile: Elastic distributed resource scaling for infrastructure-as-a-service. In *Proceedings of the 10th International Conference on Autonomic Computing (ICAC 13)*, pages 69–82.

[Pedregosa et al. 2011] Pedregosa, F., Varoquaux, G., Gramfort, A., Michel, V., Thirion, B., Grisel, O., Blondel, M., Prettenhofer, P., Weiss, R., Dubourg, V., Vanderplas, J., Passos, A., Cournapeau, D., Brucher, M., Perrot, M., and Duchesnay, E. (2011). Scikit-learn: Machine learning in Python. *Journal of Machine Learning Research*, 12:2825–2830.

[Pfitscher et al. 2013] Pfitscher, R. J., Pillon, M. A., and Obelheiro, R. R. (2013). Diagnóstico do provisionamento de recursos para máquinas virtuais em nuvens IaaS. *31o. Simpósio Brasileiro de Redes de Computadores (SBRC)*, pages 599–612.

[Popa et al. 2012] Popa, L., Kumar, G., Chowdhury, M., Krishnamurthy, A., Ratnasamy, S., and Stoica, I. (2012). FairCloud: Sharing the network in cloud computing. In *Proc. ACM SIGCOMM*, pages 187–198.

[Suleiman et al. 2012] Suleiman, B., Sakr, S., Jeffery, R., and Liu, A. (2012). On understanding the economics and elasticity challenges of deploying business applications on public cloud infrastructure. *Journal of Internet Services and Applications*, 3:173–193.

AZOS - Uma ferramenta para migração em *multi-clouds**

Raul Leiria[1], Vinícius Gubiani[1], Fabricio Cordella[1], Thiago Nascimento[1], Miguel da Silva[1], Gabriel Susin[1], Marcelo Drumm[1], Tiago Ferreto[1]

[1] Pontifícia Universidade Católica do Rio Grande do Sul (PUCRS)
Porto Alegre – RS – Brasil

{raul.leiria,vinicius.gubiani,fabricio.cordella}@acad.pucrs.br

{thiago.nascimento,miguel.silva.001}@acad.pucrs.br

{gabriel.susin,marcelo.drumm}@acad.pucrs.br

tiago.ferreto@pucrs.br

Abstract. *Cloud computing paradigm brought to companies and end-users more flexibility in costs, online services and hosting. The possibility in migrate services and infrastructures to the cloud is an attractive, however the lock-in caused by cloud computing providers usually is not convenient to the customers at latency or cost reasons. Therefore, this work presents the AZOS tool for migrating instances in multi-clouds. It was performed a study case where instances were migrated in a multi-cloud composed by OpenStack and Microsoft Azure clouds.*

Resumo. *O paradigma da computação em nuvem possibilitou às empresas e usuários finais maior flexibilidade em custos, serviços e hospedagens online. A possibilidade de migrar serviços e infraestruturas para a nuvem é um atrativo, porém nem sempre o lock-in causado por os provedores de nuvem é interessante a seus clientes, seja por questões de custo ou latência. Em razão disso, este trabalho apresenta a ferramenta AZOS para migração de instâncias em multi-clouds. Foi realizado um estudo de caso onde instâncias foram migradas em uma multi-cloud formada por nuvens OpenStack e Microsoft Azure.*

1. Introdução

Nos últimos anos a popularização da Internet elevou a demanda por serviços *online* [Ren and Zhai 2014]. Ao invés de arquivos e processamentos serem mantidos e feitos localmente em computadores, a tendência é que dispositivos locais, bem como computadores *desktops*, *laptops* e *smartphone*s, sejam utilizados em sua maior parte como terminais de acesso [Jin and Lin 2012]. O paradigma da computação em nuvem possibilita contratar recursos computacionais sob demanda através do sistema *Pay-As-You-Go*. Entretanto, apesar das vantangens desse modelo os clientes podem sofrer o efeito de *lock-in* em relação ao seu provedor de nuvem, o qual os coloca sob regras e preços determinados por ele. As *multi-clouds* possibilitam aos usuários manter suas infraestruturas e serviços em provedores de nuvem distintos [Petcu 2013], todavia a portabilidade entre eles ainda é um desafio em razão dos diferentes padrões e tecnologias utilizados.

*Este trabalho foi apoiado pelo Programa PDTI, financiado pela Dell Computadores do Brasil Ltda (Lei 8.248/91).

A ferramenta proposta neste trabalho denomina-se AZOS e possibilita a migração de instâncias em *multi-clouds* pertencentes ao modelo IaaS (*Infrastructure-as-a-Service*). Seu objetivo é evitar o *lock-in* ocasionado por provedores de nuvem e garantir ao usuário a mobilidade entre eles. As entidades envolvidas no funcionamento desta ferramenta são nuvem de origem, nuvem de destino e usuário de âmbas as nuvens. Primeiramente é realizado um processo de *discovery* na nuvem de origem para gerar um *dump* do *tenant* do cliente contendo descrições do ambiente e de suas instâncias. Por segundo é realizado o processo de *build* que efetua a leitura do arquivo de descrições gerado pelo *discovery* e orquestra a nuvem do cliente no provedor de destino. Os processos de *discovery* e *build* operam sem concomitância e possibilitam ao usuário usufruir de melhores preços e condições conforme ofertas efêmeras dos provedores, graças à portabilidade adquirida com a migração de instâncias entre eles.

A partir da Introdução (seção 1) o presente trabalho está estruturado como segue. Na seção 2 está a fundamentação onde é possível encontrar conceitos chaves para o entendimento das seções seguintes. Na seção 3 estão os trabalhos relacionados a este e por seguinte na seção 4 está a descrição detalhada da ferramenta. Na seção 5 está um estudo de caso realizado em uma *multi-cloud* composta por as nuvens OpenStack [Foundation 2017] e Microsoft Azure [Microsoft 2017]. Na seção 6 está a validação da ferramenta, e por fim na seção 7 estão a conclusão e os trabalhos futuros.

2. Fundamentação

A computação em nuvem envolve diferentes níveis de abstrações e tecnologias. Sua base advém da virtualização que possibilita a divisão dos recursos dos servidores em instâncias para ser possível suportar múltiplos clientes. Em outras palavras, a virtualização permite a execução de sistemas operacionais sobre o mesmo *hardware* de forma concomitante. Sistemas operacionais podem ser virtualizados utilizando uma camada de *software* chamada *hypervisor*, que intermedia a comunicação entre sistemas operacionais *guests* e o *hardware* real. Os *hypervisors* são classificados de acordo com a sua proximidade do *hardware* (tipo 1 ou 2) [Tanenbaum and Bos 2014] e em relação a sua técnica de virtualização (paravirtualização, virtualização completa ou assistida por *hardware*). Dispositivos de entrada e saída costumam ser emulados ao passo que a interação com o processador pode ser realizada por meio de tradução binária, *hypercalls* ou utilizando as extensões de virtualização presentes em Unidades de Processamento Central (CPUs) que ofereçam suporte à tecnologia Intel VT-x ou AMD-V.

Em herança à virtualização, os gerenciadores de nuvem podem utilizar a migração do tipo *live* para migrar instâncias (máquinas virtuais). A maioria dos *hypervisors* oferece esse tipo de migração de forma nativa, bastando ao gerenciador de nuvem oferecer apenas suporte a ela. Nesse tipo de migração é necessário haver um *block storage* compartilhado entre os servidores de *compute* (*hypervisors*) para que somente sejam transferidos de um *hypervisor* para outro os dados em memória principal referentes à instância migrada. Ao contrário da migração do tipo *live*, a migração do tipo *offline* não requisita que os processadores dos servidores de *compute* (virtualização) possuam a mesma arquitetura e tampouco que os *hypervisors* sejam iguais. Essas duas limitações muitas vezes inviabilizam a portabilidade entre diferentes provedores de nuvem, o que leva à migração *offline* a ser a mais propícia para esses casos.

As plataformas de nuvem fornecem a sensação de que não há limites para expansão de *hardware*, fazendo com que muitas vezes o usuário expanda tanto sua infraestrutura virtual a ponto de se tornar inviável uma migração para outro provedor, seja por a sua extensão ou por a incompatibilidade entre eles. Nesses casos, geralmente os usuários acabam optando por realizar uma extensão de sua infraestrutura para outras plataformas de nuvem, porém mantendo ainda contrato com mais de um provedor. Esse cenário especifica o conceito de *multi-cloud*, onde infraestruturas virtuais independentes são mantidas em diferentes provedores de nuvem. As *multi-clouds* em conjunto com ferramentas para migração possibilitam portabilidade entre diferentes plataformas de nuvem, minimizando assim o efeito de *lock-in* ocasionado por elas.

Em vista das *multi-clouds* serem nuvens independentes, elas seguem os mesmos modelos de implantação e de serviço da computação em nuvem. Os modelos de serviço proporcionam diferentes níveis de operação para usuários nos serviços ofertados por as plataformas de nuvem. Nesse sentido, os modelos mais significativos são IaaS (*Infrastructure-as-a-Service*), PaaS (*Platform-as-a-Service*) e SaaS (*Software-as-a-Service*) [Buyya et al. 2013]. O modelo IaaS disponibiliza infraestrutura como serviço, geralmente servidores virtualizados, por meio da virtualização de *hardware*, *storage* e rede. O PaaS por sua vez fornece ambientes para execução, processamento e desenvolvimento de aplicações, geralmente é utilizado por programadores. Já o modelo SaaS é o mais próximo do usuário final e oferece serviços que normalmente são acessíveis via *Application Program Interface* (APIs) ou por o navegador de Internet. Exemplos disso são redes sociais e Internet *bankings*.

Ao passo que os modelos de serviço se referem ao nível de operação dos usuários numa pilha de *software*, os de implantação se referem ao local onde a nuvem está hospedada. Quando hospedada num *data center* próprio da empresa trata-se de uma nuvem privada. Quando em *data centers* de terceiros nuvem pública e quando existe a união de recursos entre nuvens privadas e públicas trata-se de uma nuvem híbrida. Nuvens privadas são interessantes para empresas que priorizam baixa latência e confidencialidade de seus dados, pois no provedor de nuvem pública existe a possibilidade de vários clientes estarem compartilhando o mesmo servidor físico, e caso a segurança de algum deles seja comprometida pode haver o comprometimento da dos demais. Cabe aos *softwares* gerenciadores de nuvem manter o isolamento dos clientes em seus respectivos *tenants*.

Atualmente, dentre os mais populares gerenciadores de nuvem *open-source* está o OpenStack. Esse gerenciador é modularizado e por isso possui seus serviços de forma independente. O Keystone é o seu serviço de identificação, ele autentica e autoriza o acesso para os demais serviços. Imagens e discos virtuais (VHDs) de sistemas operacionais são gerenciados pelos serviços Cinder e Glance, respectivamente. A gerência do *hypervisor* e do ciclo de vida das máquinas virtuais (instâncias) é efetuada pelo serviço Nova *compute*, enquanto que os recursos de rede são gerenciados por o serviço Neutron. O Microsoft Azure é popularmente conhecido por ser uma nuvem pública, entretanto ele também é um gerenciador de nuvem que possui serviços com funcionalidades equivalentes as do OpenStack. Não estão explícitas nomenclaturas para eles e tampouco detalhamentos acerca de seu funcionamento nas literaturas consultadas. Todavia, ambos gerenciadores de nuvem possibilitam acesso aos seus serviços via interfaces de comandos, o que flexibiliza o desenvolvimento de ferramentas para interagir com eles.

Ambientes de nuvem e suas instâncias costumam ser descritos por arquivos de *template*. Esses arquivos contém parâmetros e dados acerca da topologia da nuvem, de seus recursos e *tenants*. Há diferentes implementações para arquivos de *template*, em geral todas objetivam a orquestração, todavia costumam ser incompatíveis entre si. Há propostas de unificação de formatos como por exemplo o formato TOSCA [Binz et al. 2013], mas a maioria das plataformas de nuvem ainda utiliza formatos próprios de *template*. Casos conhecidos são a Microsoft Azure, que utiliza o formato *Azure Resource Manager*, o serviço de orquestração do OpenStack que utiliza o formato *OpenStack HOT* e a Amazon que utiliza o formato *AWS Cloud Formation*. Apesar de serem implementações diferentes, dentre esses três o formato da Amazon é aceito em outras plataformas de nuvem como a OpenStack.

Neste trabalho será efetivada uma *multi-cloud* utilizando o modelo de serviço IaaS e os modelos de implantação nuvem privada para o gerenciador de nuvem OpenStack e nuvem pública para o Microsoft Azure. O OpenStack utilizará o *hypervisor* KVM (*hypervisor* tipo 1 e virtualização assistida por *hardware* [Chirammal et al. 2016]) [Kivity et al. 2007] em conjunto com o emulador de periféricos QEMU [Bellard 2005]. A ferramenta AZOS utilizará em seus processos de *discovery* e *build*, migração *offline* e *templates* para orquestração da nuvem, de suas instâncias e de sua topologia.

3. Trabalhos Relacionados

[Katzmarski et al. 2014] apresentam uma técnica e prova de conceito acerca da migração *offline* de máquinas virtuais entre o OpenStack e dispositivos locais, podendo esses serem de arquitetura x86 e ARM. Para isso foi desenvolvido um *plugin* para o OpenStack que adiciona duas rotinas à API do *compute*, sendo elas para arquivar e restaurar máquinas virtuais. O processo para arquivar a máquina virtual (VM) consiste em o usuário enviar uma requisição ao *compute* node, que por sua vez cria um *snapshot* e o empacota com outros arquivos referentes à VM para futuro *download* via Glance. A fim de que seja possível executar as instâncias em dispositivos locais, os autores criaram um *web service* para migrar e gerenciar máquinas virtuais advindas do OpenStack. Esse *web service* contém as funcionalidades básicas das APIs do OpenStack, *download* e *upload* de instâncias para ir e vir da nuvem com elas.

Em seu trabalho, [Mangal et al. 2015] desenvolveram um método para realizar a integração de nuvens OpenStack públicas e privadas. Em seu experimento, os autores instalaram às nuvens em servidores físicos diferentes, assim simulando no segundo servidor uma nuvem pública. Quando ocorre a solicitação para *placement* de uma instância, o *Load analyzer* (implementado pelos autores) verifica a quantidade de uso dos recursos físicos do *host* e caso ele esteja sobrecarregado, a instância é criada na nuvem pública. Instâncias que estão na nuvem pública geram gastos financeiros e por isso o *Load analyzer* também é executado em períodos de tempo para verificar se o servidor *host* da nuvem privada está com recursos disponíveis para hospedar instâncias que sofreram *placement* na nuvem pública. Se ele estiver disponível, ocorre *live migration* no sentido nuvem pública para nuvem privada.

Em [Awasthi and Gupta 2016], os autores relatam ser possível realizar *live migration* entre nuvens de mesmo *hypervisor* já que eles possuem o mesmo formato de disco

virtual, ou entre nuvens de *hypervisors* diferentes sendo em alguns casos necessário realizar conversão para o formato de disco da nuvem de destino. Segundo os autores, os *hypervisors* suportados pelo OpenStack são KVM, QEMU, Xen, vSphere e Microsoft Hyper-V. No trabalho não é mencionado se houveram implementações ou testes com as afirmações feitas pelos autores, ao que parece é apenas um estudo teórico.

[Desai and Patel 2015] propõem uma abordagem para diminuir o tempo de migração do tipo *live* em ambientes de nuvem. Sua abordagem se baseia em modificar a etapa de pré-copia para reduzir o número de páginas da memória a serem transferidas. Para isso, os autores utiliziram um limiar que aumenta a precisão de quando e quais páginas precisam ser atualizadas no destino. Sobre essas páginas é aplicado o algoritmo *Characteristic Based Compression* (CBC) para compressão dos dados e com isso elas são transferidas. Os autores relataram ter conseguido diminuir o tempo total de migração e o tempo de *downtime* significativamente.

A ferramenta AZOS se diferencia do trabalho de [Mangal et al. 2015] e de [Katzmarski et al. 2014] por não realizar intrusões no código fonte do OpenStack, utilizando apenas as APIs nativas de seus serviços para interação. Ainda, [Mangal et al. 2015] simulam sua nuvem pública na mesma rede local de sua nuvem privada, ao passo que a ferramenta AZOS possui especificação para e foi testada com a nuvem pública Microsoft Azure. [Mangal et al. 2015] relatam o uso, [Desai and Patel 2015] utiliza com modificações para redução de tempos e [Awasthi and Gupta 2016] apenas afirma ser possível utilizar *live migration* para migração de instâncias, enquanto a ferramenta AZOS utiliza de forma prática migração *offline* e pode com isso efetuar as conversões necessárias para compatibilidade entre diferentes formatos.

Optou-se por esse tipo de migração (*offline*) em razão de não ser necessário haver compatibilidade de CPUs entre *host* de origem de destino e tampouco entre *hypervisors* e seus formatos de discos virtuais. A migração *offline* possui apenas limitações quando se trata de *hypervisors* que utilizam diferentes técnicas de virtualização, portanto sistemas operacionais *guests* que executam em virtualização completa podem ter problemas de compatibilidade caso sejam migrados para *hypervisors* que suportem apenas para-virtualização. No estudo de caso realizado neste trabalho, a ferramenta AZOS foi utilizada em ambientes onde não há para-virtualização.

4. Descrição da ferramenta

A ferramenta proposta neste trabalho foi implementada na linguagem de programação *Python* e se comunica com os gerenciadores de nuvem por meio dos *endpoints* de seus serviços. Na arquitetura da aplicação existe um usuário que utiliza a ferramenta AZOS para adquirir mobilidade em sua *multi-cloud*. Para isso são executados processos de *discovery* e *build* em suas nuvens propositando a migração de instâncias. Não há incompatibilidades entre os *hypervisors* nesse caso, pois a migração utilizada é do tipo *offline*. O ambiente de nuvem é representado por arquivos descritores tanto para importação como exportação de dados. O formato utilizado é o *JavaScript Object Notation* (JSON) e por isso os arquivos são passíveis de leitura e alteração por parte do usuário.

Tanto o formato JSON quanto o *eXtensible Markup Language* (XML) são utilizados para o intercâmbio de dados entre aplicações. O formato XML se baseia em *tags* para demarcar informações e pode ser facilmente manipulado para criar dialetos próprios

para aplicações. O formato JSON não utiliza *tags* e sim pares atributo-valor para armazenar dados referentes a objetos que podem ser ainda utilizados em conjunto com vetores ou com outros objetos. Em uma breve comparação, o formato JSON é mais robusto nos sentidos de que por não utilizar *tags*, torna menor a quantidade de informações a serem transmitidas e processadas. Sua sintaxe é humanamente mais legível que a do XML e explicitamente é visível a utilização da orientação a objetos em sua essência. Por essas razões o formato JSON é utilizado neste trabalho tanto para o arquivo de *template Open Cloud Exportation File* (OCEF) quanto para os arquivos de credenciais das nuvens.

Antes de qualquer interação com as nuvens da *multi-cloud* é necessário autenticar-se nelas. Conforme Exemplo 1, os dados necessários para isso ficam presentes nos arquivos de credenciais (JSON) lidos por os processos de *discovery* e *build*. Os gerenciadores de nuvem disponibilizam URLs específicas para que ferramentas externas possam se comunicar com eles por meio de suas APIs. Usuários podem possuir mais de um *tenant* (projeto) na mesma nuvem, logo é necessário que seu nome esteja especificado no arquivo de credenciais. Cada *tenant* pode possuir mais de um domínio o que leva à necessidade de sua menção também no arquivo de credenciais. Nome de usuário e senha são atributos do arquivo que em conjunto com os outros dados são enviados à URL das APIs dos *endpoints*. Se os dados estiverem corretos ocorre a autenticação e autorização para interação com os seviços da nuvem.

Na arquitetura da ferramenta representada pela Figura 1 existem três entidades: nuvem de origem e de destino que formam a *multi-cloud*, e usuário dessa *multi-cloud*. O papel do usuário é preencher os arquivos de credenciais com informações fornecidas por seus provedores de nuvem e utilizar a ferramenta AZOS para realizar o processo de *discovery* na nuvem de origem e *build* na nuvem de destino. Durante esses dois processos a ferramenta AZOS interage com os gerenciadores de nuvem em tarefas locais e remotas. Ao final o cliente irá possuir em seu computador local os discos virtuais de suas instâncias e o arquivo de *template* OCEF que descreve sua nuvem. Esses itens serão utilizados como entrada para o processo de *build* na nuvem de destino.

O arquivo OCEF é o formato de arquivo de *template* utilizado pela ferramenta AZOS. Seu uso pode ser visualizado no Exemplo 2. O arquivo de *template* descreve informações acerca de uma máquina virtual. Seu primeiro item é referente ao seu VHD. Nele consta o seu tamanho em *gigabytes* e o caminho no computador do cliente que leva ao seu arquivo. O item seguinte se refere ao *firewall* que organiza suas regras em *security groups*. Logo estão o nome da instância, sua quantidade de memória principal em *megabytes*, seu endereço IP local e sua quantidade de processadores virtuais. Cada dado do arquivo está armazenado no formato atributo-valor.

```
{
"auth_url_sdk" : "http://192.168.10.10:5000/v3/",
"auth_url" : "http://192.168.10.10:5000",
"username" : "demo",
"user_domain_name":"Default",
"project_name":"demo",
"project_domain_name":"Default",
"password": "password"
}
```

Exemplo 1. Arquivo de credenciais

```
{
    "VM": [
        {
            "disk": {
                "/dev/vda": {
                    "file": "/tmp/instancia01_2016-12-13_16-36-11.img",
                    "size": 10
                }
            },
            "firewall": {
                "description": "Default security group",
                "name": "default",
                "rules": []
            },
            "memory": 1024,
            "name": "instancia01",
            "networks": {
                "private": [
                    "10.0.0.12/26"
                ]
            },
            "vcpu": 1
        }
    ]
}
```


Exemplo 2. Open Cloud Exportation File (OCEF)

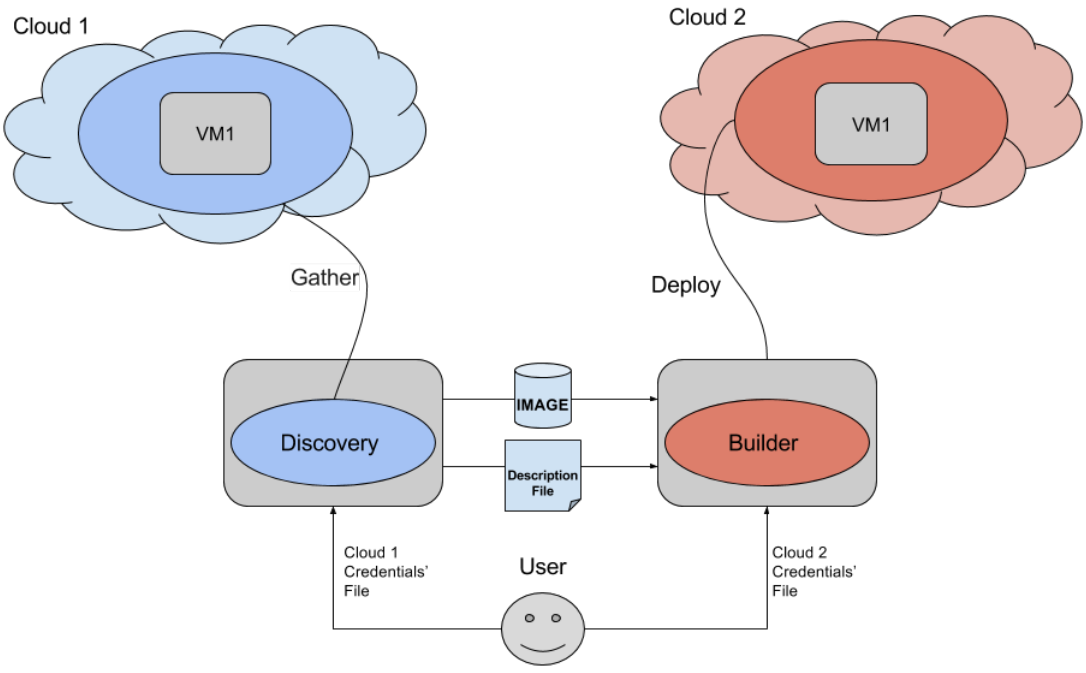


Figura 1. Arquitetura da aplicação

5. Estudo de caso

No presente estudo de caso foi desenvolvido um módulo para a ferramenta AZOS ser capaz de migrar instâncias de nuvens privadas OpenStack para nuvens públicas Microsoft Azure, e vice-e-versa. Ao passo que o OpenStack possui seus próprios formatos, nomenclaturas, *templates* para orquestração, topologias e serviços; a Azure também dispõem de seus próprios, o que ocasiona incompatibilidade entre as duas referidas nuvens. Em vista disso, a migração *offline* possibilita a portabilidade de serviços e instâncias entre nuvens (*multi-clouds*) estabelecendo um protocolo comum entre elas. Esse tipo de migração consiste em realizar processos de *discovery* e *build*, e implicitamente a eles trabalhar na homogenização dos recursos a serem migrados. Os processos de *discovery* e *build* acontecem em ordem, ocorrendo primeiro o *discovery* para gerar um *template* referente à nuvem de origem e após o *build* para importá-lo e orquestrá-lo, conforme mostrado na Figura 2.

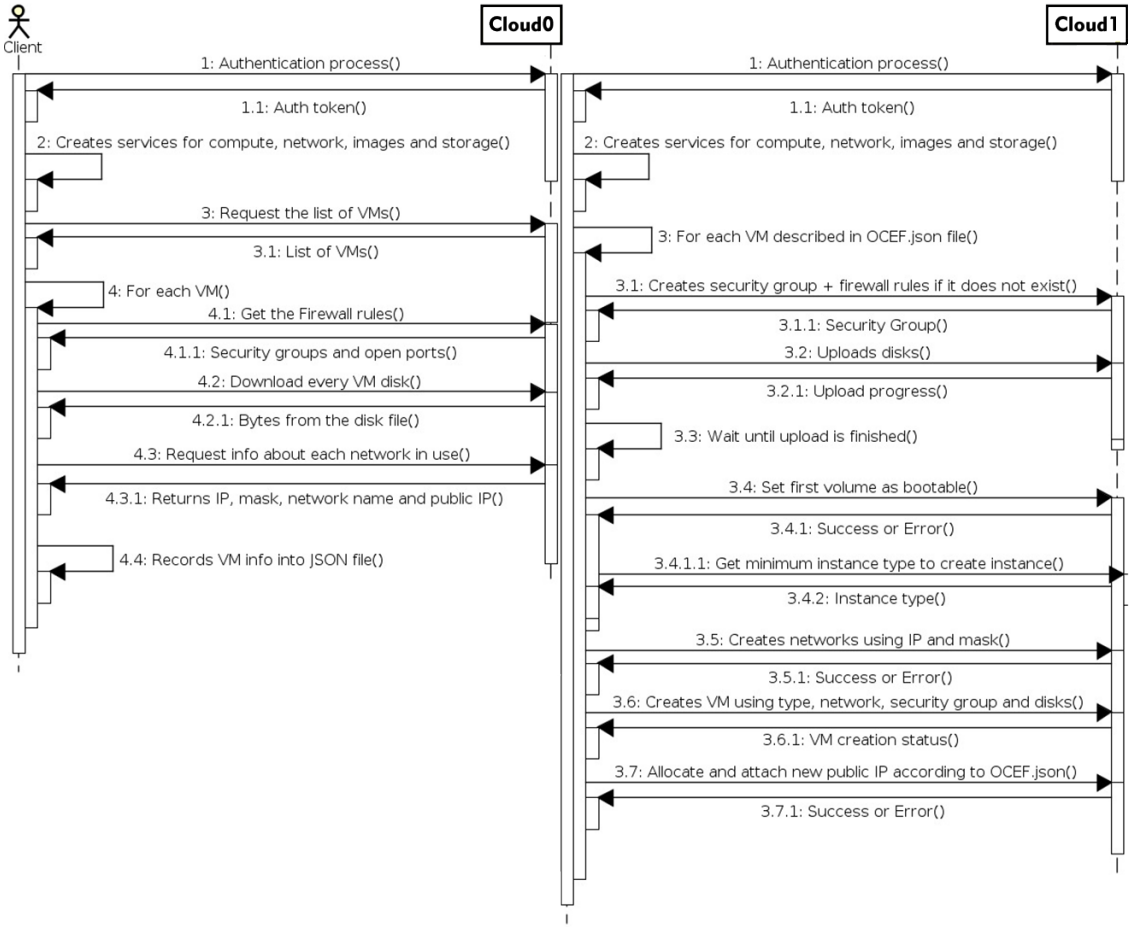


Figura 2. Diagrama de sequência dos processos de *discovery* e *build*

No processo de *discovery*, os dados do usuário são enviados para a API do serviço de identificação OpenStack Keystone que autoriza o acesso para determinado *tenant* retornando um *token* a ser utilizado em cada operação. Em sequência ocorre o instanciamento de objetos para interação com os serviços de *compute* (Nova), rede (Neutron) e imagem (Glance). Após é enviada uma requisição para gerar uma lista de instâncias contidas no *tenant* do cliente. A cada instância encontrada efetua-se o *download* de seus discos virtuais e são extraídas informações acerca do tamanho dos discos e memória principal, número de

processadores, IPs públicos, IPs locais, *security groups* e outras políticas de *firewall*. Em seguida os dados coletados são escritos no formato JSON OCEF para futura orquestração via módulo de *build*. Uma das particularidades desse estudo de caso é a necessidade de conversão do formato de disco do OpenStack. Nativamente a nuvem Microsoft Azure não aceita o disco no formato QCOW2, sendo necessáro convertê-lo manualmente para o formato VHD antes de iniciar o processo de *build*.

Da mesma forma que o processo de *discovery*, o *build* inicia pela autenticação na API do serviço de identificação da Azure que devolve um *token* caso a autenticação seja bem sucedida. Por seguinte os objetos referentes aos serviços da Azure são instanciados e se inicia a leitura do arquivo JSON OCEF gerado no processo de *discovery*. Para cada instância encontrada no arquivo *template* são criados, caso não existam, seus respectivos *security groups* e regras de *firewall*. Após é realizado o *upload* de seu disco virtual e ao final dessa etapa são extraídos da atual iteração informações acerca de quantidade de memória principal, endereços IP, número de processadores e tamanho do disco, para que seja selecionado na Azure um *instance type* com no mínimo as mesmas configurações. Por fim é criada a instância e se constar na iteração a presença de IPs públicos anexados a ela, é solicitado à API da Azure um número igual de IPs para que a instância fique acessível na Internet.

Durante o desenvolvimento do módulo Azure-OpenStack foram identificadas limitações no sentido da Azure que estão listadas a seguir: (i) usuários não administrativos não possuem privilégios para criar *resource groups*, isso faz com que usuários com privilégios de administrador sejam requeridos no processo de *build*; (ii) instâncias do Azure possuem no mínimo 30GB de tamanho de disco quando criadas pelo console Web, o que leva a um maior tempo de *download* do VHD no momento da migração para o OpenStack; (iii) no momento da migração para a nuvem Azure, o SDK utilizado no desenvolvimento realiza o carregamento das instâncias em memória principal no cliente para compactação dos dados e por conseguinte diminui o tempo de *upload*. É recomendado que o cliente possua no minimo 16GB de RAM disponível para esse processo, visto que essa é uma peculiaridade imposta por o SDK e não por a ferramenta AZOS; (iv) a nuvem Azure possui alguns IPs privados reservados para seu uso interno, podendo haver colisão de IPs caso instâncias orquestradas possuam esses mesmos endereços. Por essa razão o processo de *build* na Azure suporta apenas alocação dinâmica de IPs.

No sentido do OpenStack, são relatadas as seguintes limitações: (i) algumas instâncias utilizam seu sistema operacional no formato *Amazon Machine Image* (AMI), que consiste em arquivos separados para *kernel* e *Random Access Memory* (RAM) do *guest*. Nesses casos é necessário que haja na nuvem de destino os arquivos mencionados e eles devem estar nas mesmas versões para que a instância possa dar boot a partir deles; (ii) A ferramenta suporta apenas um volume por instância; (iii) O OpenStack oferece uma opção para *download* do disco virtual da instância tanto via interface de linha de comando quanto via Horizon. Porém, foi encontrado um *bug* que impossibilita esse *download*. Para contornar essa limitação, a ferramenta AZOS no processo de *discovery* no OpenStack arquiva o VHD da instância como imagem no Glance e após é possível efetuar o seu *download*. Esse processo em seu inverso é válido para o processo de *build*, onde primeiramente a imagem é carregada para o Glance e após é gerado o VHD da instância a partir dela.

6. Avaliação

O estudo de caso realizado neste trabalho envolveu o desenvolvimento de módulos para que a ferramenta AZOS pudesse realizar migrações entre nuvens OpenStack e Microsoft Azure. Com o propósito de aferir o funcionamento desses módulos, foi realizada a avaliação de suas funcionalidades por meio da medição de tempo de execução de cada tarefa (etapa) dos processos de *discovery* e *build*. Para isso foram realizadas medições granulares acerca desses processos, sendo executados os processos de *discovery* e *build* tanto na nuvem OpenStack quanto na Azure. Dessa maneira foi possível realizar comparações entre os tempos de mesmas tarefas utilizando implementações diferentes, ainda que as duas nuvens em questão possuam APIs e SDKs distintos para realizar as migrações que o cliente necessita.

No que diz respeito ao cenário, as medições foram realizadas em uma nuvem privada OpenStack e em uma nuvem pública Azure. Tentou-se ao máximo reproduzir um cenário real para que as medidas coletadas tivessem maior significado no que diz respeito à ambientes de produção. O ambiente de nuvem privada foi provisionado utilizando o Devstack [Foundation 2017] em uma instalação *all-in-one* onde todos os serviços do OpenStack ficam no mesmo servidor. Já o ambiente de nuvem pública encontrava-se pré-configurado na Azure bastando apenas alguns ajustes para deixá-lo em produção com uma instância. Em ambas nuvens há uma instância com a configuração de um processador, 1GB de memória principal, disco rígido virtual de 30GB e um endereço IP local. O cliente das duas nuvens está na mesma rede local que a nuvem privada OpenStack enquanto que a nuvem pública Azure está acessível a partir da Internet em um *data center* do provedor Microsoft.

No estudo de caso foram realizadas migrações de instâncias no sentido de OpenStack para Azure e vice-versa. Para ser possível comparar as mesmas etapas de migração para as duas nuvens, foi estabelecida a métrica tempo. É possível medir o tempo das etapas em razão de haver uma padronização de tarefas nos métodos da ferramenta, onde cada método representa uma ou mais tarefas a serem desempenhadas ou no processo de *discovery* ou no processo de *build*. Com isso as etapas (tarefas) medidas foram em relação à rede, regras de *firewall*, *flavor*, autenticação e migração (*download* e *upload*) de discos virtuais. As nomenclaturas e modos de operação das duas nuvens são distintos, por isso foi necessário estabelecer etapas bem definidas de maneira que mesmo diferentes implementações para a mesma tarefa pudessem ser comparadas tanto nos dois processos de *discovery* quanto nos dois de *build* realizados no estudo de caso.

As medições de tempo das etapas equivalentes foram feitas utilizando contadores de tempo dentro do código. Os processos de *discovery* e *build* são divididos em etapas, antes de cada uma delas é inserido um temporizador inicial e após um temporizador final. Assim é possível realizar a diferença entre tempo final e inicial para que seja possível obter o tempo total de execução de determinado trecho de código referente à etapa analisada. Com o propósito de diminuir o erro nas medições, foram realizadas três execuções para cada etapa em diferentes turnos, já que a vazão e disponibilidade dos *links* de rede poderiam ser afetadas. Essa metodologia foi adotada principalmente em razão dos tempos de *download* e *upload* dos discos virtuais serem maiores do que os das outras tarefas, haja vista que ambos possuem tamanho na unidade de gigabyte e por isso dependendo da rede e por conseguinte do horário pode haver uma drástica variação de tempo.

A Figura 3 contém o gráfico com os tempos das etapas dos processos de discovery nas nuvens OpenStack e Azure. Na etapa de leitura das regras de *firewall* na nuvem OpenStack o tempo é aproximadamente duas vezes maior que na Azure, isso ocorre em razão de serem necessários três laços de repetição aninhados na OpenStack ao passo que na Azure são necessários somente dois para leitura dos *security groups*. Quando lidas as informações referentes à rede das instâncias, a implementação do *discovery* para a nuvem Azure consegue obter os dados apartir de um único objeto enquanto que por limitações do SDK utilizado na OpenStack é necessário realizar iterações para obter os respectivos dados. O referido SDK ainda possui um método nativo para obter os dados relativos ao *flavor* associado à instância, o que motiva essa etapa a possuir menor tempo do que a mesma etapa no *discovery* da Azure. A etapa de autenticação em ambas implementações do *discovery* (Azure e OpenStack) é semelhante, com isso acredita-se que o maior tempo necessário para autenticação na nuvem Azure ocorre por alguma particularidade em sua API.

Através da Figura 4 é possível visualizar o gráfico que contém os tempos das etapas de *build* realizadas nas duas nuvens. O tempo para criar a instância é maior na Azure em razão de ser necessário estabelecer uma URL para que o *blob storage* referente ao VHD vindouro da OpenStack seja anexado a ela. As regras de *firewall* por sua vez requisitam maior tempo para serem estabelecidas na instância da Azure pois é necessário criar um *security groups* e definir suas regras, ao passo que no provisionamento do OpenStack um *security group* padrão já é criado para todas as instâncias. Em razão da limitação de a Azure reservar alguns endereços IPs para uso interno, o processo de *build* solicita um endereço de IP dinâmico para o servidor DHCP da Azure, o que leva o seu tempo de estabelecimento da rede a ser maior. Na etapa referente a *flavor* da instância é necessário selecionar o que atenda no mínimo as suas características de *hardware* virtual, caso não seja possível então é necessário criá-lo e isso faz com que o tempo da etapa *flavor* na OpenStack seja maior. Os tempos de autenticação são iguais ao do processo de *discovery* já que a implementação e utilização são as mesmas.

Apesar de os processos de *discovery* e *build* possuirem respectivamente as tarefas de *download* e *upload* dos discos virtuais das instâncias, por razões de disparidade entre as grandezas obtidas nas medições de tempo, optou-se em colocá-los no gráfico representado pela Figura 5 de forma exclusiva. O tempo da etapa de migração (*download*) no processo de *discovery* na Azure é aproximadamente 5,6 vezes maior do que na OpenStack em razão de o cliente das duas nuvens estar na mesma rede local que a nuvem privada. Já na etapa de *upload* do disco virtual do *build* da Azure o tempo é aproximadamente 2,3 vezes maior do que na do OpenStack, a pouca diferença de tempo ocorre em razão de o SDK utilizado no desenvolvimento da ferramenta realizar um mapeamento do disco virtual da instância migrada em memória principal no computador do cliente, realizando dessa maneira uma compactação dos dados antes deles serem enviados pela rede.

Na avaliação do estudo de caso os grandes gargá-los encontrados são relativos à quantidade de banda disponível nos *links* de rede, tamanho dos discos virtuais das instâncias e peculiaridades para interação com as APIs das duas nuvens. O tempo total de migração de uma instância da nuvem OpenStack para a nuvem Azure foi de aproximadamente 71,783 minutos, já a migração da nuvem Azure para a OpenStack levou aproximadamente 147,30 minutos, sendo então duas vezes maior.

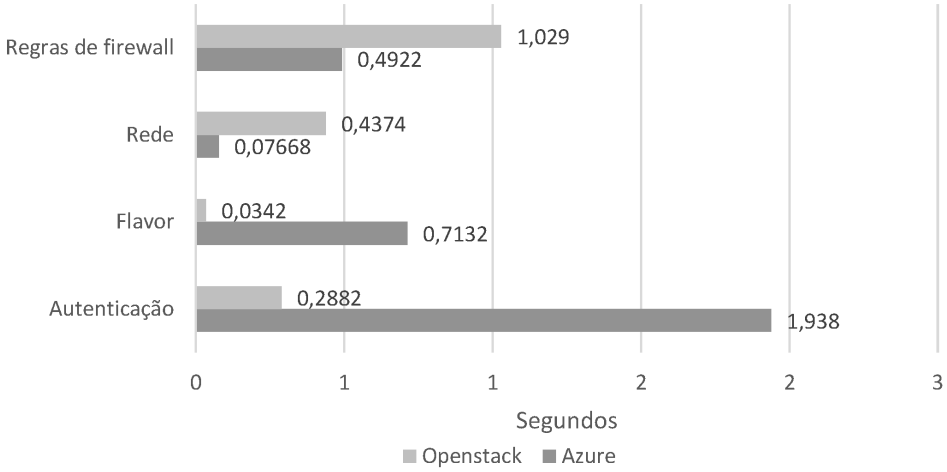


Figura 3. Médias dos tempos dos processos de discovery

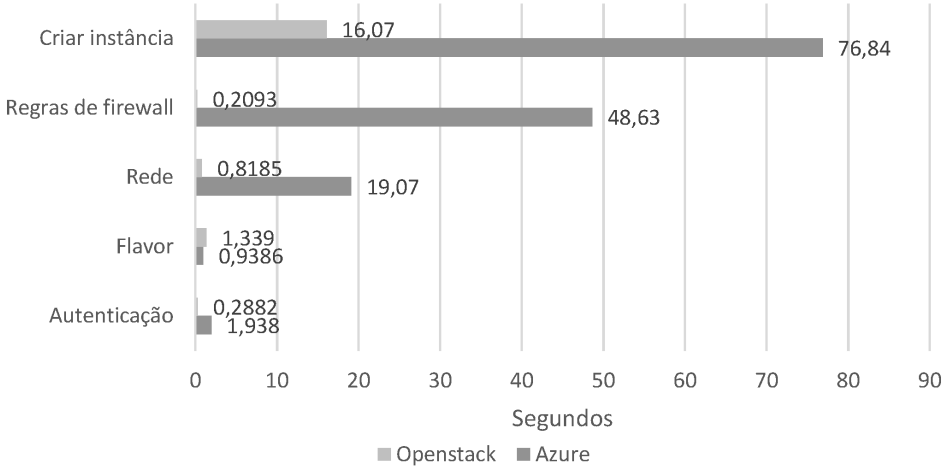


Figura 4. Médias dos tempos dos processos de build

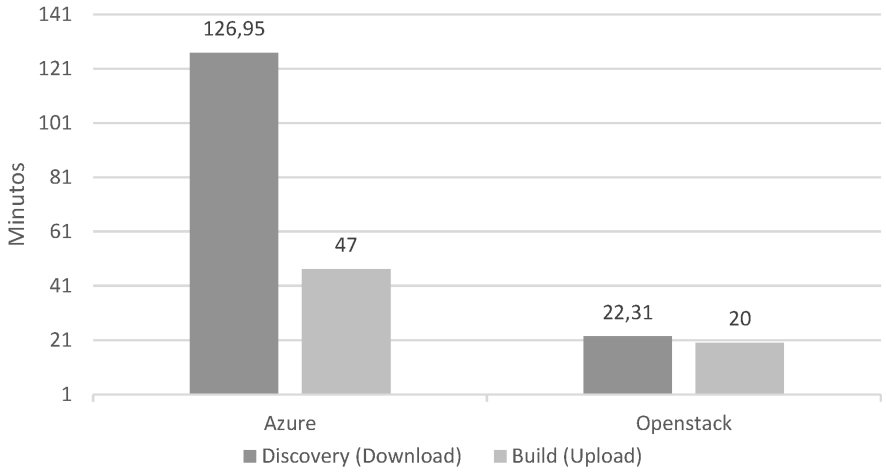


Figura 5. Médias dos tempos de migração dos discos rígidos virtuais

7. Conclusão e trabalhos futuros

O objetivo deste trabalho foi apresentar a ferramenta AZOS para efetuar migrações de instâncias em *multi-clouds*. Em seu modelo arquitetural há duas nuvens e um cliente que deseja portabilidade entre elas. Na nuvem de origem é realizado um processo de *discovery* que consiste na leitura da topologia da nuvem do cliente para gerar um arquivo de *template* contendo informações referentes ao ambiente e as instâncias que nele estão. Na nuvem de destino é efetuado um processo de *build* para orquestrar a nuvem representada no arquivo *template* gerado pelo processo de *discovery*, nesse processo ainda são efetuados os *uploads* dos discos virtuais resultantes do processo de *discovery*. A ferramenta em sua atual versão suporta migrações entre nuvens OpenStack e Microsoft Azure, ela é compatível com diferentes sistemas operacionais *guests*, contanto que eles sejam virtualizáveis e suportem as arquiteturas dos *hypervisores* utilizados.

A abordagem utilizada neste trabalho para realizar a migração de instâncias no modelo IaaS foi a migração do tipo *offline*. Optou-se por esse tipo em razão da *live migration* levar à limitações de compatibilidade em *multi-clouds* devido a heterogeneidade de tecnologias presentes nesse modelo. Ainda, em algumas abordagens são utilizados *web services* para intermediar às migrações e com isso a latência, *overhead* e tempo total podem aumentar em relação à migração feita a partir do computador do cliente. Nesse caso o cliente das duas nuvens orquestra toda a migração a partir de sua estação de trabalho. Ele efetua o processo de *discovery* para gerar o *template* referente a sua nuvem de origem e logo efetua o processo de *build* para orquestrar sua nuvem na plataforma de destino. Dessa forma as conversões entre formatos e descritores de ambiente podem ser efetuadas sem prejuízos a integridade das instâncias.

Foi realizado um estudo de caso onde houve uma migração na *multi-cloud* formada pelas nuvens OpenStack e Microsoft Azure, com o propósito de verificar e validar a ferramenta proposta. A migração de Azure para OpenStack levou 147,30 minutos em razão dos dados migrados terem sido trafegados pela Internet até um dos *data center* da Microsoft e não haver nenhum tipo de compactação de dados na transmissão. Na segunda migração, de OpenStack para Azure, o tempo total de migração foi de aproximadamente 71,783 minutos. A segunda migração levou a metade do tempo da primeira por o SDK utilizado para interagir com a Azure carregar toda a instância a ser migrada em memória principal no cliente e então realizar uma compactação de dados. Contudo, o tempo de migração sempre é proporcional à largura de banda dos *links*, tamanho dos VHDs migrados e a disponibilidade de rotas percorridos durante a migração. Eventuais falhas na transmissão dos dados podem comprometer a migração em um todo.

Em trabalhos futuros pretende-se estender a arquitetura da ferramenta AZOS para ser possível abranger outros modelos de serviço e provedores de nuvem, para inclusive aumentar a abrangência dos testes realizados. A compatibilidade de recursos entre os provedores será um dos maiores desafios. O atual cenário da Internet propicia cada vez mais o uso da computação em nuvem, exemplos disso são serviços de *storage*, *hosting*, *Internet bankings*, *eCommerces*, VDIs (*Virtual Desktop Infrastructures*), dentre outros vários. A latência, controle de custos e sensação de *hardware* inifito muitas vezes levam às empresas a migrar suas infraestruturas para a nuvem. O lado contrário disso é a mobilidade e incompatibilidade entre provedores, o que gera a necessidade de ferramentas que possibilitem a portabilidade e compatibilidade entre eles.

Referências

Awasthi, A. and Gupta, R. (2016). Multiple hypervisor based open stack cloud and vm migration. In *2016 6th International Conference - Cloud System and Big Data Engineering (Confluence)*, pages 130–134.

Bellard, F. (2005). Qemu, a fast and portable dynamic translator. In *USENIX Annual Technical Conference, FREENIX Track*, pages 41–46.

Binz, T., Breitenbücher, U., Haupt, F., Kopp, O., Leymann, F., Nowak, A., and Wagner, S. (2013). Opentosca–a runtime for tosca-based cloud applications. In *International Conference on Service-Oriented Computing*, pages 692–695. Springer.

Buyya, R., Vecchiola, C., and Selvi, S. T. (2013). *Mastering cloud computing: foundations and applications programming*. Elsevier.

Chirammal, H. D., Mukhedkar, P., and Vettathu, A. (2016). *Mastering KVM Virtualization*. Packt Publishing Ltd.

Desai, M. R. and Patel, H. B. (2015). efficient virtual machine migration in cloud computing. In *Communication Systems and Network Technologies (CSNT), 2015 Fifth International Conference on*, pages 1015–1019. IEEE.

Foundation, O. (2017). OpenStack Open Source C. C. `http://openstack.org/`.

Jin, D. and Lin, S. (2012). *Advances in Computer Science and Information Engineering*. Springer.

Katzmarski, B., Herrholz, A., Paolino, M., Rigo, A., and Nebel, W. (2014). Considering vm migration between iaas clouds and mobile clients: Challenges and potentials. In *Cloud Networking (CloudNet), 2014 IEEE 3rd International Conference on*, pages 327–332. IEEE.

Kivity, A., Kamay, Y., Laor, D., Lublin, U., and Liguori, A. (2007). kvm: the linux virtual machine monitor. In *Proceedings of the Linux symposium*, volume 1, pages 225–230.

Mangal, G., Kasliwal, P., Deshpande, U., Kurhekar, M., and Chafle, G. (2015). Flexible cloud computing by integrating public-private clouds using openstack. In *2015 IEEE International Conference on Cloud Computing in Emerging Markets (CCEM)*, pages 146–152.

Microsoft (2017). Microsoft Azure: Cloud Computing Platform & Services. `https://azure.microsoft.com/`.

Petcu, D. (2013). Multi-cloud: expectations and current approaches. In *Proceedings of the 2013 international workshop on Multi-cloud applications and federated clouds*, pages 1–6. ACM.

Ren, F. and Zhai, J. (2014). *Communication and popularization of science and technology in China*. Springer.

Tanenbaum, A. S. and Bos, H. (2014). *Modern operating systems*. Prentice Hall Press.

Um Método de Seleção de Provedores de Computação em Nuvem Baseado em Indicadores de Desempenho

Lucas Borges de Moraes[1], **Adriano Fiorese**[1]

[1]Departamento de Ciência da Computação – (DCC)
Universidade do Estado de Santa Catarina – (UDESC)
Caixa Postal 631 – 89.219-710 – Joinville – SC – Brasil

lucasborges1292@gmail.com, adriano.fiorese@udesc.br

Abstract. *Coping with the succesfull cloud computing paradigm, the arising of a large number of new cloud computing service providers have been leveraged. This fact hindered the clients' ability to choose among those several cloud computing providers the most appropriate one to attend their requirements and computing needs. This work aims to specify a logical/mathematical method able to select the most appropriate cloud computing providers to the user, based on the analysis of the values of each performance indicator desired by the customer and associated with every cloud computing provider. The method is a three stages algorithm that evaluates, scores, sorts and selects different cloud providers based on the utility of their performance indicators. An example of the method usage is given in order to illustrate its operation.*

Resumo. *O paradigma de computação em nuvem tornou-se bem sucedido e alavancou o surgimento de um grande número de novas empresas provedoras de serviços de computação em nuvem. Esse fato dificultou a capacidade de escolha por parte do cliente de selecionar o provedor de nuvem mais apropriado para suprir as suas necessidades computacionais. Este trabalho visa especificar um método lógico/matemático capaz de selecionar os mais adequados provedores de nuvem para o usuário do método, baseado na análise e pontuação dos valores de seus indicadores de desempenho. O método é estruturado em três etapas e representa um algoritmo que avalia, pontua, ordena e seleciona diferentes provedores, com base na utilidade de seus indicadores. Um exemplo de utilização do método é fornecido de forma a ilustrar seu funcionamento.*

1. Introdução

A modernização da sociedade trouxe consigo a necessidade de dispor recursos computacionais sob demanda, de maneira eficiente, facilitada e com custo acessível. A computação em nuvem tornou-se um paradigma diferenciado de hospedagem e distribuição de serviços computacionais pelo mundo via Internet. Esse paradigma abstrai do usuário a complexa infraestrutura e arquitetura interna provedora do serviço e trouxe o benefício do melhor aproveitamento dos recursos computacionais [Hogan et al. 2013, Zhang et al. 2010].

O sucesso do paradigma da computação em nuvem motivou o surgimento de um grande número de novas empresas como provedoras de computação em nuvem. Essa quantidade crescente de novos provedores tornou a tarefa de escolha e seleção do mais

adequado provedor, para cada necessidade/requisito de seus consumidores, um processo complexo.

A escolha do melhor provedor de nuvem pode ser feita através da avaliação da qualidade geral de um provedor de nuvem. Essa avaliação pode ser feita através da medição sistemática da qualidade de cada um dos indicadores de desempenho ou PIs (*Performance Indicators*) individuais do provedor em questão, resultando em um determinado valor ou *ranking*. Assim, cada provedor possuirá um *ranking* associado calculado em função dos seus PIs. O provedor que apresentar a melhor pontuação através de seus PIs, estará nas primeiras colocações desse *ranking*, e será assim, teoricamente, o mais apropriado para aquele usuário.

Os PIs são ferramentas que permitem uma coleta sintetizada de informações referentes a um determinado aspecto de uma organização e são responsáveis por quantificar (atribuir valor) aos objetos de estudo a serem mensurados. A questão que surge é: Como utilizar tais PIs para mensurar a qualidade de cada provedor para cada usuário? A resposta a essa questão dá-se por meio do método de seleção a ser especificado neste trabalho. Esse método busca utilizar os diferentes tipos de dados (numéricos ou categóricos) coletados de cada provedor de nuvem e armazenados em diferentes PIs, para pontuar uma lista finita de distintos provedores de acordo com os requisitos demandados por cada possível consumidor dos recursos desses provedores. Cada consumidor é um usuário do método, apresentando x requisitos e desejando w diferentes provedores com base nos valores esperados para m PIs de interesse. O método desenvolvido é um algoritmo lógico/matemático capaz de pontuar, ordenar e selecionar os w provedores mais adequados para cada usuário específico. O método proposto é determinístico e agnóstico quanto a quais PIs utilizar, ou seja, o usuário pode solicitar (na expressão de entrada do método) qualquer PI e valor desejado, e havendo provedor compatível, o mesmo será pontuado, ordenado e selecionado.

Este trabalho está organizado da seguinte forma: a seção 2 apresenta trabalhos relacionados à seleção, pontuação e ranking de provedores de nuvem com base em indicadores. A seção 3, apresenta e discute o método proposto que pontua e ordena os provedores de nuvem com base nos PIs desejados pelo usuário. A seção 4 ilustra um exemplo, com dados hipotéticos, de aplicação do método proposto a fim de demonstrar seu funcionamento, apresentando os resultados obtidos. Finalmente, a seção 5 apresenta as considerações finais do trabalho.

2. Trabalhos Relacionados

Diferentemente dos trabalhos já desenvolvidos, esse trabalho apresenta uma solução determinística para o problema da seleção de provedores de nuvem, que leva em conta a análise de indicadores tanto qualitativos quanto quantitativos, considerando diferentes níveis de importância atribuídos pelo próprio usuário de acordo com suas necessidades computacionais. Essa análise também leva em consideração uma abordagem matemática que leva em consideração não só a priorização de certos indicadores, como também tolerâncias máximas aceitáveis aos valores dos indicadores desejados.

Por outro lado, em [Sundareswaran et al. 2012], é proposta uma nova arquitetura de corretagem na nuvem, onde os corretores são responsáveis pela seleção do serviço apropriado para cada usuário/cliente (consumidor). O corretor tem um contrato com os provedores, recolhendo suas propriedades (os indicadores de desempenho), e com os con-

sumidores, recolhendo suas exigências de serviço. Ele analisa e indexa os prestadores de serviço de acordo com a semelhança de suas propriedades. Ao receber uma solicitação de seleção de serviço o corretor procurará o índice para identificar uma lista ordenada dos provedores candidatos com base em quão bem eles correspondem às necessidades dos utilizadores.

Um *framework* chamado "SmiCloud" é apresentado em [Garg et al. 2013]. Ele é responsável por mensurar a qualidade de serviço (QoS) de provedores de nuvem e ranqueá-los com base nessa qualidade calculada. A qualidade está diretamente relacionada aos valores de cada métrica do *Service Measure Index*(SMI) [CSMIC 2014] classificados em funcionais e não funcionais. O trabalho utiliza o método *Analytical Hierarchical Process* (AHP) [Saaty 2004] para ponderação no cálculo da qualidade e ranqueamento dos provedores.

O *framework* desenvolvido em [Baranwal and Vidyarthi 2014] apresenta uma expectativa quanto as métricas de QoS (baseadas também no SMI) que o provedor deve ter e as repassa a um corretor que o auxilia na escolha do mais adequado. Ele utiliza um método de votação para o ranqueamento (idêntico a [Shirur and Swamy 2015]) em que são levadas em consideração as exigências do usuário. Cada métrica (que pode assumir diferentes tipos - numérico, booleano, um intervalo, um conjunto não ordenado) irá funcionar como um eleitor e os provedores de nuvem serão os candidatos para elas.

O trabalho desenvolvido em [Achar and Thilagam 2014] apresenta uma abordagem baseada em um corretor de serviços para selecionar o mais adequado provedor de nuvem com base na medição da qualidade de serviço prestada de cada provedor, priorizando aqueles mais adequados às necessidades de cada solicitação ao corretor.

Em [Shirur and Swamy 2015], um *framework* é especificado para quantificar, através de métricas de Qualidade de Serviço (QoS), a eficiência de diferentes provedores de computação em nuvem, ranqueando-os. O *framework* divide as métricas de QoS em duas categorias: as dependentes da aplicação (confiabilidade, disponibilidade, segurança, *data centers*, custo, sistemas operacionais suportados, plataformas suportadas, tempo de resposta, *throughput* e eficiência) e as dependentes do usuário (reputação, interface do cliente, teste gratuito, certificação, sustentabilidade, escalabilidade, elasticidade e experiência do usuário).

O trabalho desenvolvido em [Wagle et al. 2015] propõe um modelo de avaliação que verifica a qualidade e o status dos serviços em nuvem fornecidos por provedores de nuvem. O dados são obtidos por auditores de nuvem e são visualizados em um mapa de calor ordenado pelo desempenho de cada provedor, mostrando-os em ordem decrescente de qualidade de serviço global prestado.

3. O Método de Seleção Proposto

O método proposto visa auxiliar a tomada de decisão de um usuário que deseja selecionar o mais adequado provedor de computação em nuvem por meio de valores de PIs de interesse e a importância de cada PI para o usuário em questão. Para tal, o usuário ou um terceiro (ex: o servidor onde está hospedado o método de seleção) deve possuir inicialmente uma base de dados contendo uma lista com distintos provedores de nuvem, cada qual com seu respectivo conjunto de PIs e seus valores. A Figura 1 apresenta uma visão

geral do método de seleção de provedores a ser descrito nesta seção. A entrada de dados (*Inputs*) corresponde a uma lista $P = [P_1, P_2, P_3, ..., P_n]$ com n distintos provedores de nuvem candidatos a seleção, cada qual com, inicialmente, M diferentes PIs associados (cujos valores são conhecidos e estão armazenados no banco de dados do método) e uma expressão de entrada gerada pelo usuário, contendo os m PIs de interesse (subconjunto dos PIs cadastrados no banco de dados do método) e o nível de importância de cada um (L níveis possíveis, arbitrado pelo usuário).

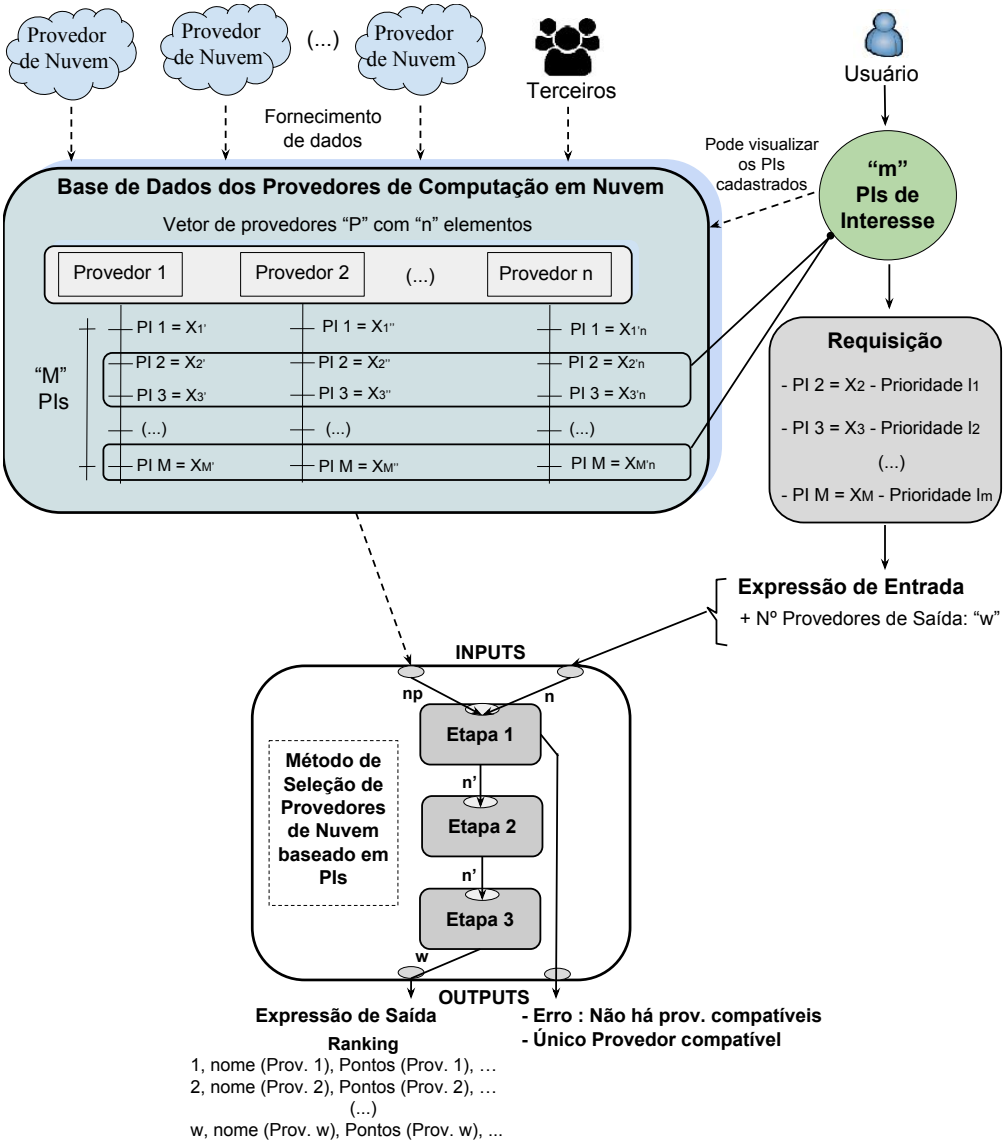


Figura 1. Método de seleção de provedores de nuvem baseado em PIs

Essa lista inicial P será diminuída, logo na primeira etapa do método, com base na expressão de entrada do usuário (que deve apresentar o identificador de cada PIs de interesse, assim como seus respectivos valores desejados e sua respectiva importância e eventual tolerância) e terá n' elementos, com $n \geq n'$. Se $n' = 0$, nenhum provedor disponível no banco de dados é compatível com o usuário, e assim interrompe-se o processo com uma mensagem de erro; se $n' = 1$ há um único provedor compatível, que é retor-

nado; se $n' > 1$ segue-se com as etapas seguintes. A saída de dados esperada (*Outputs*), exceto as condições especiais mencionadas, será uma lista com os w provedores mais bem pontuados pelo método.

O método não restringe quais PIs o usuário deve utilizar para selecionar seu provedor. Contudo, para a computação em nuvem, é possível utilizar um conjunto de PIs específicos, como aqueles presentes no *framework Service Measure Index* (SMI) [CSMIC 2014], e outros, presentes em trabalhos como: [Sundareswaran et al. 2012, Garg et al. 2013, Baranwal and Vidyarthi 2014, Achar and Thilagam 2014, Wagle et al. 2015, Shirur and Swamy 2015].

O método proposto é dividido em três etapas principais:

1. Eliminação de provedores incompatíveis ao usuário;
2. Pontuação dos PIs quantitativos e qualitativos para cada nível de importância; e
   (a) Pontuação dos PIs quantitativos.
   (b) Pontuação dos PIs qualitativos.
3. Cálculo da pontuação final geral e ranqueamento dos provedores.

Cada uma dessas etapas será discutida nas subseções a seguir.

3.1. Etapa 1 – Eliminação dos Provedores Incompatíveis

A Figura 2 ilustra o que ocorre nesta primeira etapa do método de seleção. A Etapa 1 remove da lista inicial de provedores candidatos (lista P), todo provedor incompatível ao usuário (conceito e ser definido nessa seção), gerando assim, uma nova lista P' com n' diferentes provedores. As pontuações de cada provedor ocorrem efetivamente na segunda e terceira etapa.

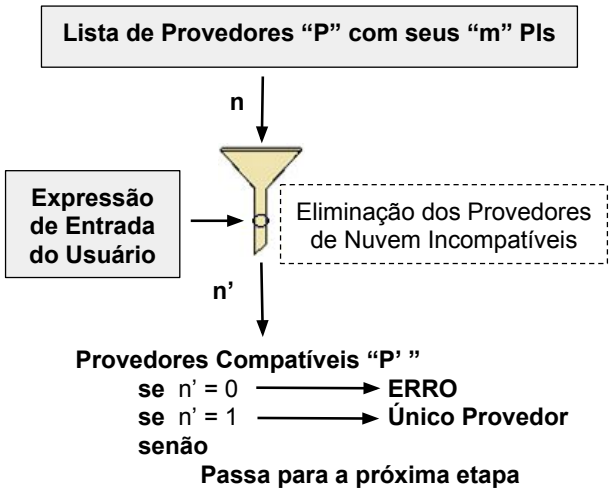


Figura 2. Etapa 1 – Eliminação de provedores incompatíveis

Dependendo do valor de n' o processo de pontuação e seleção prossegue ou finaliza. Se há mais de um provedor compatível para a seleção ($n' > 1$), inicia-se a próxima etapa, caso contrário, ocorrem as saídas especiais identificadas na Figura 2.

O conceito de compatibilidade ou incompatibilidade de um provedor de nuvem está associado com o atendimento ou não dos requisitos do usuário. Esses requisitos são

modelados como valores desejados dos PIs de interesse do usuário. Nesse sentido, para o completo entendimento do que significa esse atendimento, é necessário uma compreensão mais detalhada de como são considerados e tratados os PIs no método proposto.

Um PI pode ser classificado como essencial ou não essencial pelo usuário. Um PI essencial possui importância máxima, indicando que se ele não for atendido com aquele valor informado, o usuário não conseguirá atingir seus objetivos. Logo, será considerado incompatível todo provedor que não atender a todos os PIs essenciais do usuário.

Os PIs não essenciais possuem diferentes níveis de importância, que podem variar entre os níveis "Alto" e "Baixo". Quando um provedor de nuvem não atende um PI não essencial, isso não incapacita o usuário de atingir seus objetivos, mas pode prejudicá-lo (diretamente relacionado a importância estipulada pelo usuário). Por exemplo, não atender um PI de nível "Alto" significa que usuário ficará seriamente prejudicado em atingir seus objetivos; por outro lado, não atender um de nível "Baixo", implica em que o impacto negativo será mais tolerável (ex: PIs opcionais).

Os PIs podem ser classificados em duas classes [Jain 1991]: quantitativo (discreto ou contínuo) e qualitativo (ordenado ou não ordenado). Se o PI for quantitativo, há três classificações quanto ao comportamento [Jain 1991] da expectativa que seus valores apresentam para as necessidades dos usuários: HB ou *Higher is Better* (maior é melhor), indicando que o provedor de nuvem pode apresentar valores maiores que desejado pelo usuário; LB ou *Lower is Better* (menor é melhor), indicando que o provedor pode apresentar valores menores que o desejado pelo usuário e NB ou *Nominal is Best* (nominal é melhor) implicando que valores devem ser idênticos ao ideal. Assim, seja PI_{ij} o j-ésimo PI quantitativo de interesse de um usuário e pertencente ao i-ésimo provedor de P (P_i). Se PI_{ij} armazena o valor x e PI_{ij} atende ao valor y, especificado pelo usuário, então conclui-se que:

$$\begin{cases} x \geq (y - t_j) \text{ se } PI_{ij} \in HB \\ x \leq (y + t_j) \text{ se } PI_{ij} \in LB \\ x = (y \pm t_j) \text{ se } PI_{ij} \in NB \end{cases} \quad (1)$$

Em que t_j representa a tolerância (a x) por parte do usuário para o j-ésimo PI quantitativo de interesse, quando o valor desejado pelo mesmo é y. A tolerância padrão é zero, para cada PI, mas ela pode ser ajustada pelo usuário via expressão de entrada, para cada PI desejado.

Se PI_{ij} for qualitativo, ele pode ser ordenado ou não ordenado [Jain 1991]. Essa subclassificação advém da existência ou não de uma relação de ordem entre os possíveis valores assumíveis de um PI qualitativo. Assim, caso o PI seja qualitativo não ordenado (não existe relação de ordem), a regra é simples: se o valor de PI_{ij} (x) é aquele que o usuário deseja, então PI_{ij} atende y, caso contrário não atende. Caso o PI seja ordenado, cada valor (categoria ou classe) de PI_{ij} possui uma certa relação com os outros, graduando de um nível mais baixo até um nível mais alto. Se o usuário especificar um valor de nível mais baixo, um valor de nível mais alto pode também satisfazê-lo, depende do PI em questão.

Para resolver tal problema será criada uma nomenclatura que indicará se um PI qualitativo ordenado possui tolerâncias para categorias abaixo (*Lower*) e/ou acima

(*Higher*) da categoria desejada, semelhante a nomenclatura de [Jain 1991]: HT ou *Higher is Tolerable* (Categorias acima da desejada são toleráveis), LT ou *Lower is Tolerable* (Categorias abaixo da desejada são toleráveis) e HLT ou *Higher and Lower is Tolerable* (Categorias acima e abaixo da desejada são toleráveis). Todas essas tolerâncias podem ser especificadas pelo usuário do método via expressão de entrada. Caso nada seja informado, o padrão usado é NB (sem tolerância alguma). Assim, dado um PI qualitativo PI_{ij}, que armazena o valor x (*string*-categoria), e PI_{ij} é o j-ésimo PI qualitativo de interesse pertencente ao i-ésimo provedor de P (P_i). Se PI_{ij} atende ao valor y, especificado pelo usuário, conclui-se que:

$$\begin{cases} x = y \text{ se } PI_{ij} \text{ é não ordenado OU } PI_{ij} \in NB \\ x \geq y \text{ se } PI_{ij} \text{ é ordenado E } PI_{ij} \in HT \\ x \leq y \text{ se } PI_{ij} \text{ é ordenado E } PI_{ij} \in LT \\ PI_{ij} \text{ é ordenado E } PI_{ij} \in HLT \end{cases} \tag{2}$$

Logo, levando em consideração a premissa de atendimento e o conceito de incompatibilidade, ao fim dessa etapa inicial restará uma lista com somente provedores de nuvem compatíveis com as exigências de PIs essenciais atendidas. Na próxima etapa é necessário pontuar esses provedores de acordo com os demais PIs desejados pelo usuário.

3.2. Etapa 2 – Pontuação dos PIs para cada Nível de Importância

A segunda etapa visa pontuar os níveis de importância para cada provedor individualmente, de acordo com a utilidade de cada um dos seus PIs, sejam eles quantitativos ou qualitativos. A Equação 3 apresenta a pontuação por nível de importância, como a razão entre a pontuação dos PIs quantitativos e qualitativos que pertencem àquele nível l, e o total de PIs pertencentes ao nível l (quantidade m_l).

$$P_l(P_i) = \frac{\sum_{k=1}^{m} (P_{qt}(PI_k) + P_{ql}(PI_k))}{m_l}_{\ [PI_k \in l]} \tag{3}$$

A Etapa 2 finaliza quando são pontuados todos os L níveis de importância, para cada um dos n' provedores disponíveis. Independentemente do nível, a pontuação individual para um PI quantitativo e um qualitativo é feita de formas diferentes.

3.2.1. Pontuação dos PIs Quantitativos

A pontuação do j-ésimo PI quantitativo de um provedor i (PI_{ij}), será 0, caso não atenda ao valor numérico y, especificado pelo usuário. Caso o valor atenda, ele será pontuado proporcionalmente ao quão útil esse valor é em relação a todos os outros provedores disponíveis na lista de seleção. A função de avaliação de um PI quantitativo está representada na Equação 4. Ela retorna sempre um valor real normalizado entre 0 e 1.

$$P_{qt}(PI_j) = \begin{cases} 0, \text{se } valor(PI_{ij}) \text{ não atende } y \\ C_1 + C_2 * \frac{valor(PI_{ij}) - y}{X_{max} - y}, \text{se } PI_{ij} \in HB \\ C_1 + C_2 * \frac{y - valor(PI_{ij})}{y - X_{min}}, \text{se } PI_{ij} \in LB \\ C_1 + C_2 * \frac{t_j - |y - valor(PI_{ij})|}{t_j}, \text{se } PI_{ij} \in NB \end{cases} \tag{4}$$

O número X_{max} é o maior valor encontrado dentre todos os n' provedores na lista P' para aquele PI_{ij}; assim como X_{min} é o menor valor e t_j a distância máxima tolerada do ponto ótimo (y) para um PI do tipo NB (desde que o mesmo atenda y, ou seja, pertença ao intervalo de aceitação $[y - t_j; y + t_j]$). As constantes reais (parâmetros) C_1 e C_2 pertencem ao intervalo aberto normalizado entre $]0, 1[$, assim, $C_1 + C_2 = 1$, obrigatoriamente. O valor de t_j é configurado pelo usuário para cada PI de interesse.

A constante real C_1 pondera a pontuação dada a correspondência (*matching*) mínima desejada (incluindo as tolerâncias associadas ao PI, se existirem) entre o valor do provedor (x) e o valor desejado (y), baseando-se no tipo de PI (HB, LB ou NB) em análise. A constante real C_2 pondera a pontuação dada ao quanto esse valor de PI se sobressai ao mínimo desejado, ou seja, o valor x é, na prática, melhor que o valor y desejado.

3.2.2. Pontuação dos PIs Qualitativos

Para os PIs qualitativos ao invés de valores numéricos tem-se valores categóricos (*string*). Em relação aos PIs qualitativos não ordenados: se a categoria é a que o usuário especificou, então o PI recebe pontuação máxima 1; caso contrário recebe 0. Para os PIs qualitativos ordenados, a pontuação depende da tolerância suportada e informada pelo usuário (HT, LT ou HLT) em relação ao valor (categoria) ofertado pelo provedor. Categorias de níveis superiores e/ou inferiores à categoria desejável y podem ser toleráveis ao usuário, podendo assim pontuar.

A Figura 3.2.2 ilustra como a pontuação é influenciada pelo tipo de tolerância associada a um PI qualitativo ordenado (HT, LT e HLT), como se especificam os níveis das categorias (valores) e como se calcula a distância entre esses níveis, para nove distintas categorias hipotéticas. O nível da categoria y e das categorias toleradas estão sublinhadas na Figura 3.2.2. O nível de uma categoria é representado por um número inteiro positivo, que inicia em 1 e finaliza com o total de categorias disponíveis para aquele PI, ordenados de forma crescente (níveis inferiores - números menores; níveis superiores - números maiores). A distância entre as categorias A (valor do PI qualitativo do provedor) e y (valor desejado) é o módulo da diferença entre o nível de ambas, tendo papel relevante na pontuação de um PI qualitativo ordenado. A pontuação de uma categoria tolerada (pontuação não nula) é uma constante (nomeada C_3, valor real menor que 1) multiplicada pela distância normalizada entre os níveis das categorias A e y. Assim, a categoria desejada recebe pontuação máxima 1, as categorias vizinhas (superior e inferior) terão pontuação igual a C_3. As demais variam no intervalo aberto de $]0, C_3[$ conforme a Equação 5. Com K_1 e K_2 sendo, respectivamente, o número total de categorias acima e abaixo do nível da categoria y.

$$P_{ql}(PI_{ij}) = \begin{cases} 1, \text{se } valor(PI_{ij}) = y \\ C_3 * \frac{K_1 - dist(valor(PI_{ij}), y) + 1}{K_1}, \text{se } PI_{ij} \in HT \\ C_3 * \frac{K_2 - dist(valor(PI_{ij}), y) + 1}{K_2}, \text{se } PI_{ij} \in LT \\ C_3 * \frac{K_1 + K_2 - dist(valor(PI_{ij}), y) + 1}{(K_1 + K_2)}, \text{se } PI_{ij} \in HLT \\ 0, \text{Caso contrário} \end{cases} \quad (5)$$

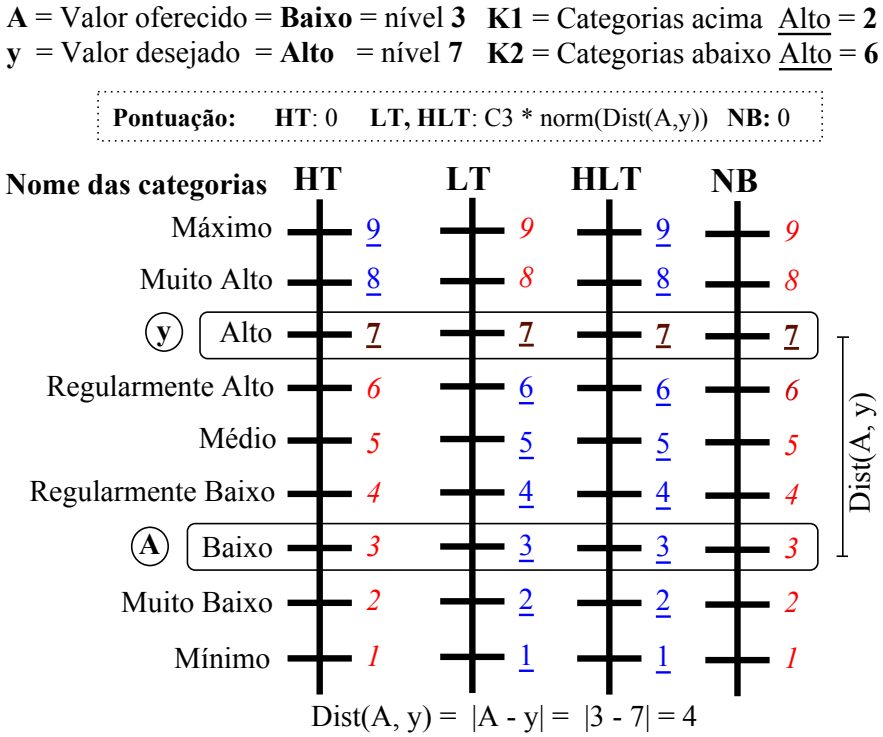


Figura 3. Relação entre pontuação e tipo de tolerância associada a um PI qualitativo ordenado hipotético com 9 categorias

A constante real C_3 representa a pontuação máxima que um categoria tolerável (diferente da desejável y, mas que se enquadra nas tolerâncias associadas para aquele PI) pode assumir. Logo, quanto menor o valor de C_3 mais agressiva é a penalidade (perda de pontuação) aplicada a todo e qualquer PI_{ij}, cuja categoria x diverge do ponto ótimo desejado y.

3.3. Etapa 3 – Cálculo da Pontuação Final de cada Provedor

Por meio das duas etapas anteriores, foram calculadas as pontuações para cada um dos L diferentes níveis de importância, para cada provedor da lista P'. Assim, a pontuação final do provedor de nuvem será a média aritmética ponderada das pontuações dos diferentes níveis de importância, cujos pesos são diretamente proporcionais ao nível de importância que aquela pontuação se refere. A Equação 6 apresenta a pontuação final de um provedor P_i.

$$P_{final}(P_i) = \sum_{l=1}^{L} (\alpha_l * P_l(P_i)) \tag{6}$$

É importante notar que obrigatoriamente $alpha_1 + \ldots + \alpha_L = 1$, normalizando a pontuação de cada provedor entre 0 e 1. Uma técnica para calcular cada α_l é utilizar uma matriz de julgamentos [Saaty 2004]. Nessa matriz, para esse trabalho, os elementos a serem relacionados são os níveis de importância para os PIs.

A Tabela 1 apresenta uma possível matriz de julgamentos para 4 níveis de importância ($L = 4$). Os valores atribuídos são baseados na escala de Saaty [Saaty 2004]. A

matriz de julgamento representa a análise pareada dos elementos que a compõe, e nesse caso, indica quão importante é o elemento da linha i em relação a coluna j. Assim, com essa metodologia obtém-se a diagonal igual a 1 e as inversões observáveis. A Tabela 1 representa a matriz de julgamento original idealizada para esse trabalho. Na última linha é apresentada a soma das colunas, antecipando o próximos passo que é a normalização dessa matriz através da divisão dos valores da coluna pela soma dos valores da coluna. Com a normalização, o cálculo dos pesos para cada nível dá-se pela média aritmética dos valores das linhas, da tabela normalizada. A Tabela 2 representa a Tabela 1 normalizada e já com os pesos calculados para cada nível de importância.

Tabela 1. Matriz de julgamento: Relações de importância para 4 diferentes níveis de importância

Nível de importância	Essencial	Alto	Médio	Baixo
Essencial	1	2	4	9
Alto	$\frac{1}{2}$	1	2	6
Médio	$\frac{1}{4}$	$\frac{1}{2}$	1	3
Baixo	$\frac{1}{9}$	$\frac{1}{6}$	$\frac{1}{3}$	1
Soma Coluna	$1,8611$	$3,6667$	$7,3333$	$19,00$

Tabela 2. Matriz de julgamento normalizada para 4 níveis de importância

Nível de importância	Essencial	Alto	Médio	Baixo	Pesos
Essencial	0,5373	0,5455	0,5455	0,4737	**0,5255**
Alto	0,2687	0,2727	0,2727	0,3158	**0,2825**
Médio	0,1343	0,1364	0,1364	0,1579	**0,1413**
Baixo	0,0597	0,0455	0,0455	0,0526	**0,0508**

Assim, após as verificações de consistência da matriz de julgamentos [Saaty 2004], temos os pesos para cada nível de importância ($l \in L$), de acordo com a última coluna da Tabela 2. A Equação 7 apresenta as incógnitas α_j da Equação 6, resolvidas por meio da Tabela 2.

$$P_{final}(P_i) = 0,5255 * P_{Essencial}(i) + 0,2825 * P_{Alto}(i) + 0,1413 * P_{Medio}(i) + 0,0508 * P_{Baixo}(i) \quad (7)$$

Após a pontuação de todos os provedores, a lista de provedores compatíveis é ordenada em ordem decrescente de pontuação. Com isso, os w primeiros provedores dessa lista são retornados ao usuário conforme solicitado.

4. Utilizando o Método Proposto

Após especificado o método, é necessário expor alguns exemplos quanto a aplicação prática do mesmo sobre um conjunto de dados reais ou hipotéticos, para mostrar o seu funcionamento e sanar dúvidas quanto aos procedimentos e resultados obtidos. Essa seção visa aplicar o método especificado sobre um possível conjunto de dados. A Tabela 3 é um exemplo de dados representativos de provedores de nuvem candidatos que podem ser utilizados no método de seleção descrito. Ela contém seis provedores fictícios disponíveis, cada qual com sete PIs.

Tabela 3. Exemplo de dados de PIs para seis provedores de nuvem

PI/Provedor	P_1	P_2	P_3	P_4	P_5	P_6
RAM (Gb)	4	2	4	8	16	4
Armz (Gb)	10,00	50,00	80,00	15,00	5,00	3,00
Custo (R$/h)	0,30	0,50	1,00	1,50	1,20	1,00
Disp (%)	99,90	95,00	88,00	99,00	90,00	90,00
NI	5	3	9	12	10	8
Seg	Médio	Médio	Alto	Baixo	Médio	Médio
SOS	L, W	L, M	W	L	L, M, W	W
Legenda						
RAM: Quantidade média de memória RAM disponibilizada em Gigabytes.						
Armz: Quantidade média de memória para armazenamento de dados em Gigabytes.						
Custo: Custo médio do serviço em reais por hora.						
Disp: Disponibilidade média do serviço por ano em percentagem.						
NI: Número total de tipos de instâncias (VMs) disponibilizadas. Inteiro > 1.						
Seg: Nível estimado de segurança e privacidade da informação. Possui 3 categorias: Alto, Médio e Baixo.						
SOs: Sistemas operacionais suportados. Possui 3 categorias: Windows (W), Linux (L) e Mac (M).						

O primeiro passo é identificar a natureza de cada PI e verificar quais atendem e quais não atendem os valores desejados pelo usuário. Assim, inicialmente é necessário que de alguma maneira (ex: um banco de dados mantido por especialistas ou pelo usuário do método) o método proposto reconheça que o PI "NI" seja quantitativo discreto com função de utilidade HB; "Custo", quantitativo contínuo LB; "RAM", quantitativo discreto HB; "Armz", quantitativo contínuo HB; "Disp", quantitativo contínuo HB; "Seg", qualitativo ordenado NB; e "SOS", qualitativo não ordenado. Se o usuário buscar um provedor de nuvem que atenda os requisitos conforme a Tabela 4 , temos a Tabela 5. Ela mostra os valores desejáveis (de acordo com as funções de utilidade associadas) e toleráveis (de acordo com os valores de tolerância associados às funções de utilidade fornecidas pelo usuário) para cada PI, bem como outros valores úteis necessários para a pontuação final de cada provedor.

Tabela 4. Exemplo de expressão de entrada do usuário

PI e valor desejado	**Tolerância**	**Nível de importância**
$RAM = 4$	–	Essencial
$Armz = 5$	0,50	Essencial
$Custo = 1,00$	0,10	Alto
$Disp = 90$	0,50	Alto
$NI = 8$	1	Médio
$Seg = Medio$	HT	Médio
$SOs = W$	–	Baixo

Por meio dessa análise, já é possível determinar se há provedores incompatíveis (Etapa 1). É incompatível qualquer provedor que não atenda todos os PIs essenciais ("RAM" e "Armz"). Um PI atende certo valor desejado, se o seu valor está no intervalo de

Tabela 5. Análise dos PIs apresentados no exemplo

PI	Valores desejáveis	Valores toleráveis	Valores Min / Max
RAM	$[4, +\infty)$	–	$X_{max} = 16$ GB
Armz	$[5,0; +\infty)$	$[4,5; 5,0)$	$X_{max} = 80,0$ GB
Custo	$[0; 1,00]$	$(1,00; 1,10]$	$X_{min} = 0,30$ R$/h
Disp	$[90,0; 100,0]$	$[89,5; 90,0)$	$X_{max} = 99,9\%$
NI	$[8, +\infty)$	7	$X_{max} = 12$
Seg	Médio	Alto	–
SOs	W	–	–

valores desejáveis ou, pelo menos, no intervalo de valores toleráveis, ambos identificados na Tabela 5. Com base nisso, foi construída a Tabela 6, onde (•) informa que o PI atende ao valor desejado pelo usuário e (○) não atende.

Tabela 6. Identificação se o PI de um provedor atende o valor desejado pelo usuário

PI / Provedor	P_1	P_2	P_3	P_4	P_5	P_6
RAM	•	○	•	•	•	•
Armz	•	•	•	•	•	○
Custo	•	•	•	○	○	•
Disp	•	•	○	•	•	•
NI	○	○	•	•	•	•
Seg	•	•	•	○	•	•
SOs	•	○	•	○	•	•

Por meio da Tabela 6 conclui-se que o Provedor 2 não atende ao PI essencial "RAM" de 4 GB ou mais e o Provedor 6 não atende ao PI essencial "Armz" de 4,5 GB ou mais. Assim, os provedores 2 e 6 devem ser removidos da lista de provedores aptos/compatíveis para a seleção. A seguir (Etapa 2), deve-se pontuar os PIs quantitativos (Equação 4) e qualitativos (Equação 5) para P_1, P_3, P_4 e P_5. A pontuação deve ser feita por nível de importância de acordo com a Equação 3. A Tabela 7 apresenta os resultados finais calculados para os quatro níveis de importância mostrados. As constantes utilizadas foram: $C_1 = 0.7$, $C_2 = 0.3$ e $C_3 = 0.7$.

Tabela 7. Pontuação dos provedores por nível de importância

Provedor	Níveis de Importância			
	Essencial	Alto	Médio	Baixo
P_1	$0,71$	$1,00$	$0,50$	1
P_3	$0,85$	$0,35$	$0,7375$	1
P_4	$0,77$	$0,485$	$0,50$	0
P_5	$0,85$	$0,35$	$0,925$	1

A seguir (Etapa 3), calcula-se a pontuação final e consequentemente o *ranking* para cada um dos 4 provedores concorrentes, de acordo com a Equação 7. Os valores dos coeficientes utilizados para a média ponderada são os valores apresentados na Tabela

2. Tais pesos são aplicados à pontuação de cada nível de importância calculados anteriormente (Tabela 7). A Tabela 8 apresenta a pontuação final dos Provedores 1, 3, 4 e 5. Assim, de acordo com a Tabela 8, é possível ordenar os quatro provedores candidatos em ordem decrescente de pontuação:

Tabela 8. Cálculo da pontuação final com 4 níveis de importância

Fórmula geral: $P(i) = 0,5255 * P_{Essencial}(i) + 0,2825 * P_{Alto}(i) + 0,1413 * P_{Medio}(i) + 0,0508 * P_{Baixo}(i)$	
Prov.	**Pontuação final**
P_1	$0,5255 * (0,71) + 0,2825 * (1,00) + 0,1413 * (0,50) + 0,0508 * (1,00) = 0,7771$
P_3	$0,5255 * (0,85) + 0,2825 * (0,35) + 0,1413 * (0,7375) + 0,0508 * (1,00) = 0,7006$
P_4	$0,5255 * (0,77) + 0,2825 * (0,485) + 0,1413 * (0,50) + 0,0508 * (0) = 0,6123$
P_5	$0,5255 * (0,85) + 0,2825 * (0,35) + 0,1413 * (0,925) + 0,0508 * (1,00) = 0,7271$

1. P_1 com $0,7771$ pontos;
2. P_5 com $0,7271$ pontos;
3. P_3 com $0,7006$ pontos; e
4. P_4 com $0,6123$ pontos.

Logo, o Provedor 1 seria o mais apropriado para o usuário desse exemplo. O método pode retornar ao usuário uma lista contendo os w provedores melhor ordenados. Considerando $w = 3$, o retorno seria: [$\{1, P_1, 0.7771\}$,$\{2, P_5, 0.7271\}$,$\{3, P_3, 0.7006\}$].

5. Conclusão

Esse trabalho especificou um método auxiliar à tomada de decisão que pontua e ordena, visando a seleção, o(s) mais apropriado(s) provedor(es) de nuvem, dentre um conjunto inicial de provedores, com base na requisição do usuário. A requisição deve apresentar os PIs de interesse, as preferências de valores para os mesmos e suas tolerâncias e a importância de um sobre o outro. O método é dividido em três etapas e separa os PIs em dois tipos: essenciais e não essenciais. Os não essenciais possuem diferentes níveis de importância. A pontuação final leva em consideração essa importância, resultando em um número real entre 0 e 1. Quanto mais perto de 1 mais adequado e preferível é o provedor em relação aos seus concorrentes para aquele usuário em questão. Um exemplo de aplicação do método foi apresentado, demonstrando sua utilização e a praticidade de adoção.

O principal benefício do método proposto é a sua alta generalidade e dimensionalidade, ou seja, capacidade de trabalhar com todo e qualquer tipo de PI disponível independentemente se o mesmo for quantitativo ou qualitativo. Ele apresenta como contribuição levar em conta tolerâncias para os valores quantitativos ou qualitativos desejados, baseadas na função de utilidade de cada PI, para a pontuação de cada nível de importância. Ainda, disponibiliza a possibilidade de cálculo de pesos para os níveis de importância baseado em uma técnica de avaliação pareada de critérios, que é a matriz de julgamentos, bastante utilizada em problemas de auxílio à tomada de decisão. O método também é simples e intuitivo, pois não exige um domínio ou modelagem matemática sofisticada para entendê-lo ou para utilizá-lo. O método é agnóstico, não restringindo quais PIs o usuário deve utilizar para selecionar seu provedor. Uma interface especializada pode ser

disponibilizada para o usuário, auxiliando-o a construir a sua expressão de entrada. A interface informa quais os PIs cadastrados que podem ser utilizados (explicando o que ele representa, o seu tipo, comportamento e domínio), quantos provedores estão disponíveis para a escolha, quais níveis de importância disponíveis para uso e que valor será atribuído entre eles.

Trabalhos futuros incluem a aplicação do método em cenários reais, bem como a criação de um corretor (*broker*) que incorpore o método desenvolvido e auxilie um conjunto diverso de usuários. Ainda, o refinamento e incorporação de automatização na definição dos valores das constantes de ponderação para o cálculo da pontuação de PIs quantitativos e qualitativos é uma tarefa planejada. Aplicar o método a instâncias consideravelmente grandes, assim como modelar e especificar uma maneira prática de obter os diversos PIs dos provedores de computação em nuvem e mantê-los atualizados é uma tarefa futura a ser desenvolvida.

Referências

Achar, R. and Thilagam, P. (2014). A broker based approach for cloud provider selection. In *2014 International Conference on Advances in Computing, Communications and Informatics (ICACCI)*, pages 1252–1257.

Baranwal, G. and Vidyarthi, D. P. (2014). A framework for selection of best cloud service provider using ranked voting method. In *Advance Computing Conference (IACC), 2014 IEEE International*, pages 831–837.

CSMIC (2014). Service measurement index framework. Technical report, Carnegie Mellon University, Silicon Valley, Moffett Field, California. Accessed in November 2016.

Garg, S. K., Versteeg, S., and Buyya, R. (2013). A framework for ranking of cloud computing services. *Future Generation Computer Systems*, 29(4):1012–1023.

Hogan, M. D., Liu, F., Sokol, A. W., and Jin, T. (2013). *Nist Cloud Computing Standards Roadmap*. NIST Special Publication 500 Series. accessed in September 2016.

Jain, R. (1991). *The Art of Computer Systems Performance Analysis: Techniques for Experimental Design, Measurement, Simulation, and Modeling*. John Wiley & Sons, Littleton, Massachusetts.

Saaty, T. L. (2004). Decision making – the analytic hierarchy and network processes (ahp/anp). *Journal of Systems Science and Systems Engineering*, 13:1–35.

Shirur, S. and Swamy, A. (2015). A cloud service measure index framework to evaluate efficient candidate with ranked technology. *International Journal of Science and Research*, 4(3):1957–1961.

Sundareswaran, S., Squicciarin, A., and Lin, D. (2012). A brokerage-based approach for cloud service selection. In *2012 IEEE Fifth International Conference on Cloud Computing*, pages 558–565.

Wagle, S., Guzek, M., Bouvry, P., and Bisdorff, R. (2015). An evaluation model for selecting cloud services from commercially available cloud providers. In *7th International Conference on Cloud Computing Technology and Science*, pages 107–114.

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.

XV Workshop de Computação em Clouds e Aplicações

SBRC 2017

Sessão Técnica 2

PAS-CA: Uma Arquitetura Auto-escalável para Ambientes Web de Alta Demanda

Marcelo Cerqueira de Abranches[1,2], **Priscila Solis**[1], **Eduardo Alchieri**[1]

[1]Departamento de Ciência da Computação – Universidade de Brasília (UnB)
[2]Diretoria de Sistemas e Informação – Controladoria-Geral da União (CGU)

Abstract. *This paper proposes a autoscalable architecture based on containers and a self-elasticity algorithm that uses the response time as threshold for requests in a Web system. The proposal promotes the optimization of the processing of requests through efficient allocation of containers. An evaluation was carried out with a workload characterized by a time series resulting from the requests of a Web system in production and of massive use. The results show that the proposal meets the performance requirements by allocating more efficiently the number of containers than other solutions already on the market.*

Resumo. *Este artigo propõe uma arquitetura auto-escalável baseada em containers e um algoritmo de auto-elasticidade que utiliza como limiar o tempo de resposta das requisições em um sistema Web de alta demanda. A proposta promove a otimização do processamento das requisições por meio da alocação eficiente de containers. Uma avaliação foi realizada com uma carga de trabalho caracterizada a partir de uma série temporal resultante das requisições de um sistema Web em produção e de uso massivo. Os resultados mostram que a proposta cumpre os requisitos de desempenho ao alocar com maior eficiência o número de containers quando comparado com outras soluções existentes no mercado.*

1. Introdução

A computação em nuvem é essencialmente um agrupado de computadores conectados em rede em locais geográficos iguais ou diferentes, operando em conjunto para atender clientes com diferentes necessidades e carga de trabalho sob demanda, com o uso de técnicas de virtualização [Zhang et al. 2010].

A tecnologia da nuvem se direciona a uma maior distribuição através de multi-nuvens e a inclusão de vários dispositivos, como no caso de IoT (*Internet of Things*) e a integração de rede no contexto de nuvem de borda e *fog computing*. As soluções de virtualização leve, como a utilização de *containers*, são benéficas para esta configuração com dispositivos menores. Além de oferecer benefícios sobre as máquinas virtuais tradicionais na nuvem em termos de tamanho e flexibilidade, os *containers* são especificamente relevantes para as aplicações de Plataforma como Serviço (PaaS) [Felter et al. 2015].

Recentemente, o uso dos *containers* de código aberto denominados Docker [Docker 2016] se consolidou como um padrão *de facto* para que as aplicações possam estender-se de uma plataforma para outra, funcionando como microserviços em servidores Linux e ambientes PaaS. Atualmente, muitos dos principais fornecedores de infraestrutura de nuvem utilizam o Docker, por exemplo Google Inc., IBM Corp., Red Hat Inc.,

Microsoft, Rackspace, VMware Inc. [Schwartz 2014]. Recentemente, a Google lançou o GKE (*Google Container Engine*) que utiliza Kubernetes, o qual automatiza o processo de provisionamento, execução e parada de máquinas virtuais e o processo de implantação de aplicações em um número variável de *containers* com base na demanda [Avran 2014].

Este trabalho tem como objetivo propor e avaliar o desempenho de uma arquitetura autoescalável e um algoritmo para auto-elasticidade baseado na alocação eficiente do número de *containers* para atender requisições em um sistema *web* de alta demanda na nuvem. O artigo está estruturado da seguinte forma: a Seção 2 apresenta a revisão bibliográfica e os trabalhos relacionados, a Seção 3 descreve a proposta da arquitetura que posteriormente é analisada experimentalmente na Seção 4. Finalmente, a Seção 5 apresenta as conclusões e trabalhos futuros.

2. Revisão Bibliográfica e Trabalhos Relacionados

2.1. Virtualização, Gerenciadores de Nuvem e Automação de Configuração

A virtualização utiliza *software* para emular as funcionalidades de *hardware*. Isto permite que vários sistemas operacionais e aplicações distintas sejam executados em um mesmo servidor [VMware 2016]. Uma máquina virtual é uma porção de software isolada, com um sistema operacional e aplicações associadas. Uma camada de software chamada *hypervisor*, que é responsável pela execução das máquinas virtuais e pelo gerenciamento do uso dos recursos físicos, implementa independência entre as máquinas virtuais.

Para provisão de funcionalidades destes ambientes, podem ser utilizados gerenciadores de nuvem [Sefraoui et al. 2012] os quais normalmente se comunicam com os *hypervisors*, ativos de rede e equipamentos de armazenamento por meio de APIs (*Application Program Interfaces*). As ferramentas de automação de configuração permitem que se defina e aplique a configuração desejada para um conjunto de servidores [Walberg 2008].

2.2. Containers

Containers são uma tecnologia para a criação de instâncias de processamento isoladas que permitem a virtualização no sistema operacional. Dois *containers* em execução no mesmo sistema tem sua própria abstração de camada de rede, memória e processos [Docker 2016]. Esse isolamento é feito via *Kernel namespaces*. Um *namespace* é uma abstração que permite que os processos dentro do *namespace* tenham sua própria instância isolada do recurso global. Mudanças em um *namespace* são percebidas apenas por processos que são membros daquele *namespace* [Linux 2016]. O *Linux* implementa *namespaces* para sistema de arquivos, processos, rede e usuários [Felter et al. 2015].

A limitação e contabilização de recursos dos *containers* é feita por meio dos *cgroups*. Os *cgroups* permitem a alocação de recursos por meio de grupos de processos definidos pelo usuário em um sistema [RedHat 2016]. Os *containers* têm uma maior portabilidade que as máquinas virtuais, ao serem configurados de forma genérica para qualquer sistema operacional baseado em Linux.

2.3. Carga de Trabalho

O comportamento dos acessos web apresenta comportamento monofractal [Crovella and Bestavros 1997].

Os processos monofractais ou autossimilares apresentam dependência de longa duração. Um processo com dependência de longa duração tem função de autocorrelação $r(k) \sim k^{-\beta}$ com $k \to \infty$, onde $0 < \beta < 1$. O parâmetro de Hurst indica o grau de autossimilaridade de uma série, dado por: $H = 1 - \frac{\beta}{2}$. Uma série autossimilar com longa dependência tem $\frac{1}{2} < H < 1$. Com $H \to 1$, tanto o grau de autossimilaridade quanto a dependência de longa duração aumentam [Crovella and Bestavros 1997]. Existem vários métodos para a estimativa do parâmetro de Hurst (*H*). Um deles é o método de Kettani-Gubner [Kettani et al. 2002] que permite o cálculo do parâmetro *H* utilizando séries de tamanho menor do que as necessárias com outros métodos. Nesse método, o coeficiente de autocorrelação é descrito pela Eq.1.

$$R(k) = 1/2(|k+1|^{2H}) - 2|k|^{2H} + |k-1|^{2H} \tag{1}$$

Para *k*=1 tem-se $R(1) = 2^{2H-1}$, isolando *H*, têm-se $\hat{H} = \frac{1}{2}[1 + \log(1 + \hat{R_n}(1))]$. Sob a suposição de que o processo seja ergódigo, é possível calcular o parâmetro *H*, conforme a Eq.2.

$$\hat{H_n} = \frac{1}{2}[1 + log(1 + \hat{R_n}(1))] \tag{2}$$

No contexto deste trabalho, as séries autossimilares foram utilizadas para caracterização e geração do perfil da carga usado na avaliação do ambiente.

2.4. Controladores PID

Controladores PID (Proporcional-Integral-Derivativo) são um dos algoritmos de controle mais utilizados na indústria. O valor da variável de controle é lido do sistema *S*. Este valor é comparado com o valor desejado, denominado de *setpoint* e é gerado o termo de erro *e(t)*. O termo de erro é combinado nos componentes proporcional, integral e derivativo e é gerado um sinal de controle que atua no sistema *S* para controlar o valor da variável [Instruments 2015]. O componente proporcional depende da diferença entre o valor desejado (*setpoint*) e o valor atual da variável. Esta diferença é referida como erro. O ganho proporcional (*Kp*) determina a taxa de resposta de saída para o erro e o componente derivativo indica a taxa de variação instantânea deste. O componente integral soma o termo de erro ao longo do tempo.

O ajuste dos parâmetros de ganho K_p, K_i e K_d permite que o controlador PID produza os ajustes necessários. Existem diversos métodos de ajuste destes parâmetros, como o método manual, o método Ziegler-Nichols [Instruments 2015] e o ajuste utilizando o algoritmo *Coordinate Ascent*. O método manual e o *Coordinate Ascent* foram os métodos utilizados neste trabalho. No método manual, os ganhos de cada um dos componentes são ajustados com tentativa e erro. Os termos K_i e K_d são ajustados para zero, e o termo K_p é aumentado até que a saída do ciclo comece a oscilar. A partir daí aumenta-se lentamente o termo K_i para reduzir o erro estacionário. Neste ponto inicia-se o incremento do termo K_d, de modo a diminuir as oscilações na saída do ciclo [Instruments 2015].

O método *Coordinate Ascent* (CA) [Thrun et al. 2006] estabelece um laço que maximiza ou minimiza uma função de pontuação. Esta função define quão próxima do objetivo uma variável permaneceu dentro de uma iteração do algoritmo. Dada uma estimativa inicial de um conjunto de parâmetros que afeta o valor da função de pontuação, o

algoritmo modifica cada um destes parâmetros com base em um valor fixo e a partir desta modificação, verifica se esta alteração apresentou uma melhora na função de pontuação. Havendo melhora, este parâmetro é registrado, assim como o novo valor da função de pontuação e aumenta-se o passo de forma gradual. Caso contrário, o passo é diminuído gradualmente até que o intervalo de busca do parâmetro seja menor do que um valor mínimo pré-fixado. O pseudocódigo do Algoritmo 1 descreve o CA.

No contexto deste trabalho, um controlador PID é utilizado em um ambiente de malha fechada que permite manter o tempo médio de resposta de uma aplicação constante, independente da carga aplicada, por meio da variação do provisionamento de recursos (*containers*) destinado a atender as requisições.

Algoritmo 1: COORDINATE ASCENT

```
1  início
2      Enquanto a soma dos passos para cada parâmetro > limiar mínimo
3        Para cada parâmetro
4          aumente o parâmetro com o valor do passo e calcule o valor da
           pontuação atual
5        Se pontuação atual > pontuação anterior
6          mantenha o valor de parâmetro, aumente o passo e registre o valor da
           pontuação
7        Se pontuação atual < pontuação anterior
8          diminua o parâmetro com o valor do passo e calcule o valor da
           pontuação atual
9          Se pontuação atual > pontuação anterior
10           mantenha o valor de parâmetro, aumente o passo e registre o valor da
             pontuação
11         Se pontuação atual < pontuação anterior
12           diminua o passo
13 fim
14 retorna Parâmetros otimizados
```

2.5. Trabalhos Relacionados

O trabalho [Lorido-Botrán et al. 2012] compara os diversos métodos de auto-escalabilidade e elasticidade em um ambiente de nuvem. Estes métodos são separados nas categorias reativos e preditivos. Este estudo mostra a variedade de técnicas que estão sendo propostas em diferentes áreas de conhecimento.

O algoritmo PRESS [Gong et al. 2010] propõe a predição de carga de CPU. Um dos métodos apresentado utiliza o processamento de sinais usando-se transformadas rápidas de Fourier para extração de frequências dominantes para gerar séries temporais e diversas janelas de tempo.

O Haven [Poddar et al. 2015] apresenta uma proposta de dimensionamento elástico para ambientes de nuvem que se baseia no monitoramento de cargas de CPU e memória para cada máquina virtual. A partir de limiares previamente estabelecidos para consumo de CPU e memória, o algoritmo instancia novas máquinas virtuais e as insere em um *pool* de balanceamento de carga. O balanceador de carga é implementado com

SDN (*Software Defined Network*) e tem inteligência para realizar o encaminhamento da requisição para membros com melhores condições.

A proposta do presente trabalho se diferencia da abordagem do PRESS pois não realiza predição de carga e sim um dimensionamento de recursos, baseado na observação do tempo de resposta de uma aplicação atrás de um balanceador de carga. Outra diferença é que o PRESS realiza escalabilidade vertical, ou seja, aumenta recursos de memória e CPU para se ajustar a carga de trabalho, enquanto que na proposta deste trabalho se aborda a escalabilidade horizontal, ou seja, novas instâncias capazes de atender a carga de trabalho são alocadas atrás de um balanceador de carga para se ajustar às variações na demanda. A escalabilidade horizontal tem a vantagem de que os recursos alocados para atender a carga de trabalho não são limitados aos recursos físicos de uma máquina. Além disso, esta abordagem facilita a alta disponibilidade, uma vez que as demandas são atendidas por um conjunto de instâncias em paralelo.

Em relação à abordagem do Haven, a proposta do presente trabalho apresenta um método de dimensionamento do sistema a partir da observação do tempo de resposta das requisições, o que permite uma visibilidade global do comportamento do sistema. Nos casos em que não haja excessos de consumo de processamento e memória, o ambiente pode se beneficiar do aumento do paralelismo no atendimento das requisições.

Os trabalhos anteriormente citados analisam ambientes que utilizam tecnologia de máquinas virtuais em que sistemas de escalabilidade devem levar em conta o tempo de provisionamento e configuração de máquina virtual, tempo que pode alcançar minutos, enquanto o provisionamento de *containers* pode alcançar segundos [Martin 2015]. Sistemas que utilizam máquinas virtuais trabalhando de forma reativa num laço de controle como o PAS podem incorrer em violações constantes de nível de serviço, motivo pelo qual técnicas de predição de carga e pré-provisionamento são comuns nestes ambientes ([Lorido-Botrán et al. 2012], [Poddar et al. 2015] e [Gong et al. 2010]).

Sistemas projetados para prover elasticidade em ambientes de máquinas virtuais, podem não ser adequados para escalar ambientes de *containers*. Os parágrafos seguintes analisam sistemas projetados para atuar com *containers* e posicionam a PAS-CA frente a esses sistemas.

O trabalho [Google 2016] propõe uma ferramenta nativa de auto-escalabilidade chamada de *Horizontal Pod Autoscaler* (HPA), a qual escala o ambiente a partir de limiares médios de consumo de CPU dos *containers* para atender às necessidades das aplicações. O HPA faz o uso do *cadvisor* (que deve estar funcional em todos os nós workers do Kubernetes) e do *heapster* para monitoramento do uso de CPU dos *containers*. O PAS-CA se diferencia do HPA no monitoramento pois não necessita do *cadvisor* e do *heapster*. O PAS trabalha de forma independente de qualquer componente de software de monitoramento ao monitorar os tempos de resposta em um ponto único, i.e., no balanceador de carga.

O trabalho [Kan 2016] usa um método reativo e um pró ativo para escalar *containers*. A métrica utilizada é requisições por segundo e o algoritmo utiliza a informação do número de requisições por segundo que um *container* consegue atender. Esta informação é utilizada na previsão do número de *containers* necessários para atender a demanda. Os autores afirmam que este valor pode ser configurado manualmente, entretanto, o número

de requisições por segundo que um *container* atende depende das condições de CPU e memória e isso pode variar em um ambiente real. O sistema adiciona ou subtrai quantidade fixa de *containers* de acordo com a demanda enquanto que o PAS-CA adiciona ou remove instâncias de acordo com o controlador PID que informa a quantidade de *containers*, que deve ser adicionada ou subtraída a cada iteração do algoritmo de modo a manter o tempo de resposta sob controle. Além disso o PAS-CA não precisa que seja determinado o número de requisições por segundo que um *container* pode atender, o que o torna mais robusto dado que esta métrica pode variar de aplicação para aplicação e com níveis de consumo de recursos dos *containers*. A avaliação do ambiente realizada em [Kan 2016], mostra apenas o número de *containers* variando conforme a demanda e não compara o sistema proposto com outras propostas e com outras métricas de desempenho, por exemplo, número de requisições não atendidas e variações no tempo de resposta da aplicação. No presente artigo o PAS-CA é comparado com a solução dominante de mercado [Google 2016] e avalia diversas métricas de desempenho (número de requisições não atendidas, variações no tempo de resposta da aplicação, quantidade média de *containers* alocados durante os experimentos, entre outras).

O trabalho [Müller et al. 2016] utiliza a métrica de operações por segundo como limiar para escalabilidade ao estabelecer de forma empírica o número de operações por segundo que um sistema consegue atender, o que também apresenta a dificuldade de se estabelecer um valor fixo da métrica em um ambiente dinâmico.

Os trabalhos [Amazon 2016a] e [Amazon 2016b] demonstram como construir e configurar aplicações auto escaláveis dentro do ambiente de nuvem da Amazon. Porém estes trabalhos utilizam ferramentas exclusivas da Amazon (por exemplo: *CloudWatch*). O PAS CA pode ser utilizado em qualquer provedor de nuvem pois utiliza ferramentas de código aberto em sua construção, além de ser independente das ferramentas de monitoramento de terceiros para prover auto escalabilidade. O trabalho [Jesheen 2016] descreve como prover um ambiente escalável utilizando *containers docker*, porém a escalabilidade é realizada de forma manual. O PAS-CA executa a escalabilidade de forma automática.

O trabalho [Solis 2016] propõe o uso de ferramentas de Big Data e teoria de controle para provisão de um ambiente auto escalável, onde o limiar para a escalabilidade é o tempo de resposta médio de uma aplicação. Neste trabalho somente utiliza-se a configuração dos parâmetros do PID de forma manual enquanto que o presente artigo implementa o PAS-CA que complementa a técnica ao empregar a técnica de otimização de parâmetros *Coordinate Ascent*.

3. Solução Proposta

O PAS-CA (PID *based Autoscaler*-CA) é uma arquitetura inovadora em nuvem para provisão de auto-elasticidade. É utilizado um ambiente baseado em *containers* e um método de alocação que reage aos aumentos no tempo de resposta do sistema, de modo a manter esse tempo abaixo de um limiar pré-estabelecido. É utilizado um sistema de malha fechada com um controlador PID que reage às variações no tempo de resposta do sistema em conjunto com um algoritmo de otimização de parâmetros do controlador PID.

O PAS-CA é apresentada na Figura 1 e funciona da seguinte forma: (1) É estabelecido um limiar (*setpoint*) de tempo médio de resposta desejado para as requisições. O monitor de requisições recebe o tempo de resposta das requisições que chegam no balan-

ceador; (2) O monitor de requisições envia o tempo médio de resposta para o dimensionador PID que calcula o número de *containers* necessários para atingir ou permanecer no *setpoint*. (3) O PID executa o Algoritmo 2 e informa o número desejado de *containers* ao Kubernetes. (4) Kubernetes cria ou remove novos *containers*, além de garantir que o ambiente permanecerá com o número desejado até a próxima rodada do algoritmo (depois de 10 segundos). Este valor de 10 segundos foi definido pois verificou-se que é suficiente para que o Kubernetes inicie novos *containers* e estes passem a responder as requisições no *cluster* de balanceamento.

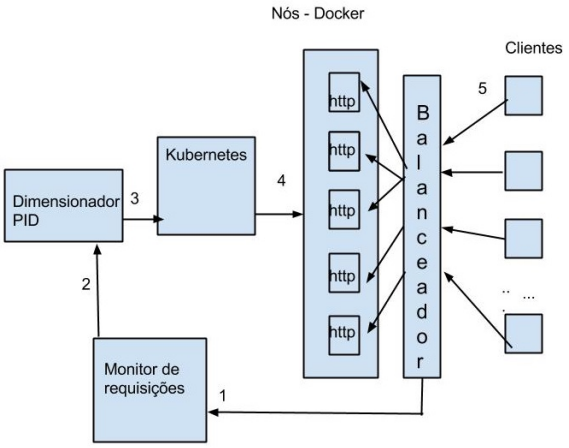


Figura 1. Arquitetura da Solução

Algoritmo 2: PAS

Entrada: Tempo médio de resposta do cluster, Número atual de *containers*

início

Leia o limiar de tempo médio de resposta desejado para as requisições: $t\_ms\_desejado$

Leia o número atual de *containers*: $n\_containers\_atual$

Leia tempo de resposta médio do cluster em ms: t_ms_atual

Calcule o erro: $e(t)$ = $t\_ms\_desejado$ - t_ms_atual

Calcule a saída do controlador PID:

$$u(t) = K_p e(t) + K_i \int_0^t e(t)\delta t + K_d \frac{d}{dt} e(t) \quad (3)$$

$n\_desejado\_containers$=$n\_containers\_atual$ + $u(t)$

fim

retorna *n_desejado_containers*

Para viabilizar a operação do algoritmo que pretende trabalhar com dados dinâmicos recebidos em tempo real, utiliza-se o fluxo de processamento mostrado na Figura 2 que descreve a integração e uso das ferramentas usadas na implementação da arquitetura da Figura 1. O *HAproxy* é um balanceador [Tarreau 2015] utilizado por *Reddit, Stack Overflow/Server Fault*, Instagram e Red Hat *OpenShift*. Seus *logs* são enviados ao *Flume* que agrega os dados em um canal de memória e os envia ao *Spark Streaming*. Para a disponibilização dos dados utilizados pelo dimensionador, o *Flume* realiza o envio em tempo

real dos *logs* de acesso do balanceador de carga. Esta informação do tempo de resposta é armazenada em formato de série temporal no servidor *Redis*. O *Redis* é utilizado para armazenar a série temporal. Com estes dados disponíveis no Redis, o PAS, nas versões manual e automática (CA) é executado e determina o número desejado de *containers*. O Kubernetes recebe a informação do número desejado de *containers* e utiliza a ferramenta *kubectl* para ajustar a alocação.

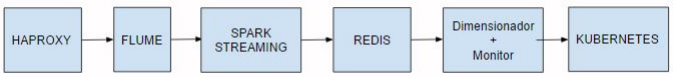


Figura 2. Fluxo de processamento entre o conjunto de ferramentas utilizadas

Para otimizar os parâmetros PID, o algoritmo *Coordinate Ascent* minimiza a soma do erro quadrático médio da série dos tempos médios de resposta em relação ao valor estabelecido no *setpoint*. São estabelecidos valores iniciais para os parâmetros *Kp, Ki* e *Kd* e um tamanho de passo inicial para cada um dos parâmetros. Também é estabelecido um valor inicial para o erro quadrático médio que é a função de pontuação, que neste caso é o quadrado da soma das diferenças dos tempos de resposta em relação ao *setpoint* dentro da janela de tempo. A janela foi definida em 2000 segundos pois verificou-se experimentalmente que esse tempo seria suficiente para ajustar cada um dos parâmetros do PID na função de pontuação.

4. Avaliação da Proposta e Resultados Experimentais

4.1. Ambiente

Foi configurado um *cluster* Kubernetes v1.4.1 no sistema operacional *CoreOS* (899.6.0 (2016-02-02)), virtualizado em *VMWare ESXi* 5.5.0. O *cluster* foi constituído com os seguintes componentes: 1 nó *master* (4 vCPUs, 6 GB de RAM), 1 nó *etcd* (4 vCPUs, 6 GB de RAM) e 3 nós *workers* (4 vCPUs, 6 GB de RAM). Foram instaladas máquinas virtuais (*VMWare ESXi* 5.5.0) no sistema operacional Ubuntu 14.04.3 LTS, com as seguintes configurações e ferramentas: 1 nó *Haproxy* 1.5.4 (4 vCPUs, 4G GB de RAM), 1 nó *Spark* 1.5.2 mais *Redis* 2.8.4 (2 vCPU, 10 GB de RAM) e 1 nó *Flume 1.7.0* mais o dimensionador (2 vCPU, 4 GB de RAM).

Os cenários de avaliação utilizam o ambiente Rubis [Rubis 2009], que é modelado como um clone do ebay (`www.ebay.com`). O RUBiS implementa as funcionalidades básicas de registro de produtos, venda, lances de leilão, navegação em produtos por região e categorias. A versão instalada do RUBiS foi a 1.4.3 obtida em [Squazt 2012]. O *Haproxy* foi configurado para balancear as requisições entre os nós Docker/Kubernetes usando os endereços IP dos nós e as portas publicadas pelo serviço do tipo *NodePort* do Kubernetes. Cada *container* teve seus recursos de processamento e memória limitados a 300 milicores e 300 MB de memória RAM.

4.2. Carga de Trabalho

Foi coletado um conjunto de acessos, entre os dias 25/05/2015 e 25/06/2015, do portal da transparência (`www.transparencia.gov.br`), na escala de 1 segundo. A série foi

caracterizada com o método Kettani-Gubner na qual foi confirmada a autossimilaridade com H=0,87, nas escalas de 1 segundo, 100 segundos e 600 segundos.

Esta série foi utilizada para gerar a cada segundo requisições simultâneas direcionadas ao endereço IP do balanceador de carga, o qual distribui as requisições entre os nós do *cluster* Kubernetes. A intensidade da carga foi multiplicada por vários índices preservando a autossimilaridade. Apesar de terem sido realizados testes com várias cargas, neste trabalho por restrição de espaço mostraremos os resultados para uma carga em que a série temporal é multiplicada por 3 (carga_3) e para uma carga variável que funciona da seguinte forma: o sistema é submetido a uma carga_1 (série multiplicada por 1) por 1000 segundos, em seguida a uma carga_2 (série multiplicada por 2) por 1000 segundos, depois a carga_3 por mais 1000 segundos, seguidas novamente pelas carga_2 e carga_1, por mais 1000 segundos cada. Esta carga variável também foi chamada de carga_1_2_3_2_1.

Quando configurado de forma manual, o controlador PID utilizou os seguintes parâmetros: Kp=0.016, Ki=0.000012 e Kd=0.096, que foram definidos após vários testes com a carga de trabalho. Para a otimização dos parâmetros do PID com o algoritmo *Coordinate Ascent*, os valores para cada um dos limiares podem ser apreciados na Tabela 1. Para a descoberta dos parâmetros para os limiares de 80, 100 e 120 ms do PAS, o *Coordinate Ascent* foi executado da seguinte forma: (1) os parâmetros Kp, Ki e Kd foram configurados em 0.01; (2) o passo de atualização foi configurado para 0.001; (3) o limiar de convergência da soma dos passos foi configurado para 0.000001; (4) o sistema foi exposto à carga_3 e esperou-se a convergência do algoritmo para cada um dos casos.

4.3. Avaliação de Resultados

Esta seção apresenta a comparação dos resultados do PAS (com ajuste manual) e do PAS-CA (com ajuste automático) com o HPA, ferramenta comercial desenvolvida pela Google. Para cada teste, o experimento foi repetido com 10 amostras diferentes. O intervalo de confiança foi definido em 95%, com 9 graus de liberdade. Em todos os testes foi utilizado o ambiente Rubis. Os limites de alocação de recursos para estes testes foram configurados como 300 *millicores* de processamento e 300 MB de memória RAM. Estas configurações foram estabelecidas para permitir que o Rubis sempre fosse iniciado em boas condições de consumo de CPU e processamento. O número mínimo de *containers* em todos os testes foi 3 para permitir que sempre houvesse pelo menos um *container* executando em cada nó.

As configurações do HPA, PAS e PAS-CA assim como as nomenclaturas utilizadas, podem ser consultadas na Tabela 1. Para o HPA foram configurados limites de processamento médio máximo desejado em 20% e 50%. Essas configurações de limiares de 20% e 50% foram definidas pois o HPA escalando com 20% de processamento permite bom desempenho de tempo de resposta, mantendo os *containers* em um nível seguro de processamento. o HPA escalando com 50% de processamento permite uma alocação menor de *containers*, piorando os tempos de resposta das aplicações, porém ainda com um nível seguro de processamento, sem que hajam travamentos dos *containers*. Limiares acima de 50% causam altos níveis de processamento ao submeter o ambiente às cargas utilizadas neste trabalho, o que origina travamento dos *containers*, *reset* de conexões e *timeouts*.

Para o PAS e PAS-CA é configurado o limite de tempo de resposta e os parâmetros

do PID (*Kp*, *Ki* e *Kd*). Para as configurações dos algoritmos PAS 80, PAS 100 e PAS 120, são usados os parâmetros encontrados de forma manual. Já para as configurações PAS 80 CA, PAS 100 CA e PAS 120 CA são usados os parâmetros calculados pelo *Coordinate Ascent*. Os limiares configurados para o PAS foram escolhidos para trazerem tempos de resposta compatíveis aos das configurações utilizadas para o HPA.

Algoritmo	CPU Livre	Limite tempo de Resposta	Kp	Ki	Kd
HPA 20	20%	-	-	-	-
HPA 50	50%	-	-	-	-
PAS 80	-	80 ms	0.01	0.000012	0.096
PAS 80 CA	-	80 ms	0.00121	0.0000505	0.00112
PAS 100	-	100 ms	0.01	0.000012	0.096
PAS 100 CA	-	100 ms	0.00131	0.0000515	0.00101
PAS 120	-	120 ms	0.01	0.000012	0.096
PAS 120 CA	-	120 ms	0.00135	0.000041	0.00115

Tabela 1. Configuração dos Algoritmos

As métricas analisadas para os testes foram: tempo médio de espera na camada de aplicação do cliente, tempo médio de espera da requisição no balanceador de carga, percentagem de requisições não atendidas, número médio de *containers* alocados durante os testes e eficiência média na alocação dos *containers*. O número médio de *containers* foi calculado registrando o número de *containers* alocados a cada segundo, e no final dividindo a soma do número total por esse tempo. A eficiência média na alocação de *containers* foi definida pela Eq. 4, em que *Nc* é o número médio de *containers* durante os experimentos, e *T* é o tempo médio de resposta em milissegundos das requisições na camada de aplicação. O cálculo da eficiência utilizando esta equação foi idealizado, pois quanto maior o tempo de resposta médio, ou quanto maior a alocação média de *containers*, menor será a eficiência obtida.

$$E = 1/(Nc * T) \tag{4}$$

As numeração das colunas das tabelas 2 e 3, apresentam a seguinte descrição: (1) intervalo de confiança para o tempo médio de resposta em milisegundos na camada de aplicação, (2) intervalo de confiança para a alocação média de *containers*, (3) intervalo de confiança para a percentagem de requisições não atendidas; (4) intervalo de confiança para a eficiência do algoritmo; (5) intervalo de confiança para o tempo médio de resposta em milisegundos medido no balanceador.

Os resultados dos experimentos realizados com a carga_3 por um tempo de 1200 segundos são mostrados na Tabela 2. Observa-se que o tempo de espera dos clientes, do PAS 80 CA foi próximo ao HPA 20, porém o PAS 80 CA apresentou um intervalo de confiança menor, o que demonstra a previsibilidade de comportamento em relação ao tempo de resposta para a carga_3. Verifica-se também que o HPA 50 alocou menos *containers*, porém apresentou maior tempo de resposta.

Nestes testes, a criação e destruição de *containers* executada pelos algoritmos de auto elasticidade não representou problemas críticos em relação ao não atendimento de requisições, visto que os intervalos de confiança para porcentagem de requisições não

atendidas apresentam valores baixos. Observa-se que as versões do PAS CA tenderam a perder menos requisições do que as versões otimizadas manualmente.

A tabela 2 também mostra que as configurações das versões do PAS conseguiram manter o tempo médio de resposta no balanceador próximo aos limiares desejados. É possível observar também que os algoritmos HPA tenderam a ter intervalos de confiança maiores do que os do PAS, o que demonstra uma maior previsibilidade no atendimento de requisitos de tempo de resposta do PAS-CA. O resultado mostra que as versões PAS-CA obtiveram melhores tempos de resposta que as versões otimizadas com parâmetros manuais, além de eficiência melhor ou igual. Também para um tempo de resposta equivalente ao obtido pelo HPA 20, o PAS 80 CA obteve melhor eficiência.

	1	2	3	4	5
HPA 20	164.797, 204.326	5.560, 5.684	0.00001, 0.000001	0.0009, 0.0009	61.696, 87.191
HPA 50	199.988, 271.039	3.0, 3.0	0.0, 0.0	0.001, 0.001	134.060, 166.074
PAS 80	168.619, 225.594	5.009, 5.111	0.00007, 0.00007	0.00100, 0.00100	84.202, 95.339
PAS 80 CA	181.819, 185.631	5.270, 5.283	0.00004, 0.00004	0.00103, 0.00103	84.544, 85.648
PAS 100	193.845, 221.755	4.254, 4.277	0.00009, 0.00009	0.0011, 0.0011	103.742, 110.421
PAS 100 CA	180.234, 207.569	4.370, 4.388	0.00001, 0.00001	0.0011, 0.0011	98.089, 99.315
PAS 120	207.609, 215.419	3.492, 3.502	0.00009, 0.00009	0.0013, 0.0013	118.899, 120.642
PAS 120 CA	203.998, 208.970	3.556, 3.561	0.000009, 0.000009	0.0013, 0.0013	113.701, 117.323

Tabela 2. Resultados dos testes para carga_3 com 95% de confiança

Para verificar a diferença entre o PAS e o HPA com a carga variável (carga_1_2_3_2_1), foram realizados testes para as configurações PAS 80 CA e HPA 20. Os resultados obtidos estão sumarizados na Tabela 3. Nesta tabela, verifica-se que o tempo de resposta médio na camada de aplicação e a alocação média de *containers* durante os testes, foram melhores para o PAS 80 CA. Verifica-se também que os dois algoritmos obtiveram tempos médios de resposta próximos, dentro do intervalo de confiança, entretanto, o intervalo de confiança obtido para os tempos de resposta apresenta-se menor para o PAS 80 CA, o que indica uma maior previsibilidade de desempenho em relação ao HPA 20. O PAS 80 CA também alocou um número menor de *containers*.

	1	2	3	4	5
HPA 20	108.786, 115.244	4.137, 4.143	0.000003, 0.000003	0.0021, 0.0021	73.477, 75.618
PAS 80 CA	108.523, 110.753	4.052, 4.078	0.000003, 0.000003	0.0022, 0.0022	70.861, 71.536

Tabela 3. Resultados dos testes para carga variável com 95% de confiança

Conforme os dados da tabela 3, o PAS 80 CA apresentou melhor eficiência que o HPA 20. Este comportamento é coerente com os dados obtidos nos testes anteriores, já que o PAS 80 CA teve tempos de resposta próximos aos do HPA 20, porém alocou menos *containers*. A percentagem de requisições não atendidas, foi baixa para tanto para o PAS 80 CA quanto para o HPA 20, o que demonstra a robustez dos dois algoritmos para cargas com mudanças abruptas de intensidade. A versão PAS 80 CA obteve tempos médios de resposta no balanceador menores que o HPA 20, o qual estava configurado para manter este valor próximo a 80 ms, porém verifica-se que este valor ficou próximo de 70 ms. Isto pode ser explicado pelo fato de que nestes testes com carga variável, o sistema passa 2000 segundos sendo submetido a carga_1, e foi observado nos diversos testes que nessa carga o tempo de resposta esteve abaixo de 80 ms, mesmo sem que o sistema seja escalado.

4.4. Testes de média zero

Para comparação dos resultados foram realizados testes de média zero para verificar estatisticamente os resultados dos diferentes sistemas dentro de um intervalo de confiança [Jain 1990]. Os testes realizados nesta seção foram configurados para apresentarem intervalo de confiança de 95%, usando-se a distribuição *t student* com 9 graus de liberdade. Os dois sistemas a serem comparados são o HPA 20 e o PAS 80 CA pois são as configurações com maiores requisitos de desempenho dentre os cenários avaliados. Foram comparados os tempos de resposta, alocação de *containers* e eficiência para a carga_3 e para a carga variável.

O resultado do teste para o tempo de resposta é mostrado na Tabela 4 (as linhas representam o "sistema 1" e as colunas o "sistema 2"). Para a carga_3, o intervalo mostra valores negativos e positivos quase simétricos, o que indica a equivalência dos dois sistemas. Para a carga variável o intervalo mostra tempos de resposta equivalentes, porém o HPA 20 mostra uma tendência para tempos de resposta maiores. A Tabela 5 apresenta a alocação de *containers* em que para ambas as cargas, o HPA 20 fez uma alocação maior de *containers*. A Tabela 6 compara a eficiência entre os algoritmos. Para ambas as cargas o PAS 80 CA foi mais eficiente. Todos os resultados nesta seção são coerentes com os resultados apresentados na seção anterior, o que valida a precisão do método de avaliação utilizado neste trabalho.

	carga_3		**carga variável**	
Sistema 1/Sistema 2	**HPA 20**	**PAS CA 80**	**HPA 20**	**PAS CA 80**
HPA 20	(0.0, 0.0)	(-20.77, 22.44)	(0.0, 0.0)	(-0.5108585, 5.26436)
PAS 80 CA	(-22.44, 20.77)	(0.0, 0.0)	(-5.26436, 0.510859)	(0.0, 0.0)

Tabela 4. Testes média zero para tempo de resposta

	carga_3		**carga variável**	
Sistema 1/Sistema 2	**HPA 20**	**PAS CA 80**	**HPA 20**	**PAS CA 80**
HPA 20	(0.0, 0.0)	(0.2767679, 0.414394)	(0.0, 0.0)	(0.0589784, 0.097679)
PAS 80 CA	(-0.4143941, -0.276768)	(0.0, 0.0)	(-0.0976786, -0.058978)	(0.0, 0.0)

Tabela 5. Testes média zero para número médio de *containers*

	carga_3		**carga variável**	
Sistema 1/Sistema 2	**HPA 20**	**PAS CA 80**	**HPA 20**	**PAS CA 80**
HPA 20	(0.0, 0.0)	(-6.53e-05, -6.5e-05)	(0.0, 0.0)	(-8.95e-05, -8.9e-05)
PAS 80 CA	(6.53e-05, 6.5e-05)	(0.0, 0.0)	(8.95e-05, 8.9e-05)	(0.0, 0.0)

Tabela 6. Testes média zero para eficiência

5. Conclusões e Trabalhos Futuros

Este trabalho apresentou a PAS-CA, uma arquitetura de auto escalabilidade para computação em nuvem que propõe o uso de *containers* e a provisão da auto elasticidade com base em uma métrica que impacta diretamente a percepção do usuário, o tempo de resposta da aplicação. Foram integradas diversas ferramentas de *Big Data*, orquestradores de *containers* e técnicas de controle. Os resultados experimentais mostram que nos cenários avaliados a proposta otimiza a alocação do número de *containers* para manter o tempo

de resposta das aplicações dentro de um limiar estabelecido. Foi feita uma comparação da proposta com uma ferramenta da Google, o HPA, e os resultados mostram que a PAS-CA tem potencial para ser uma alternativa de escalabilidade em sistemas de nuvem. Nos cenários avaliados, a PAS-CA obteve melhor eficiência na alocação de *containers* que o HPA. Trabalhos futuros pretendem integrar outras métricas de interesse para provisão de auto-elasticidade, como limiar de CPU e memória e a avaliação da proposta em um sistema em produção.

Referências

Amazon (2016a). Aws, tutorial: Scaling container instances with cloudwatch alarms. `http://docs.aws.amazon.com/AmazonECS/latest/developerguide/cloudwatch_alarm_autoscaling.html`. [Acessado em 29/03/2017].

Amazon (2016b). Deploying elastic beanstalk applications from docker containers. `http://docs.aws.amazon.com/elasticbeanstalk/latest/dg/create_deploy_docker.html`. [Acessado em 29/03/2017].

Avran, A. (2014). Google announces cloud container engine using kubernetes. `https://www.infoq.com/news/2014/11/google-cloud-container-engine`. [Acessado em 16/11/2016].

Crovella, M. E. and Bestavros, A. (1997). Self-similarity in world wide web traffic: evidence and possible causes. *Networking, IEEE/ACM Transactions on*, 5(6):835–846.

Docker (2016). The definitive guide to docker containers. `https://www.Docker.com/sites/default/files/WP-\%20Definitive\%20Guide\%20To\%20Containers.pdf`. [Acessado em 03/04/2016].

Felter, W., Ferreira, A., Rajamony, R., and Rubio, J. (2015). An updated performance comparison of virtual machines and linux containers. In *Performance Analysis of Systems and Software (ISPASS), 2015 IEEE International Symposium On*, pages 171–172.

Gong, Z., Gu, X., and Wilkes, J. (2010). Press: Predictive elastic resource scaling for cloud systems. In *Network and Service Management (CNSM), 2010 International Conference on*, pages 9–16. IEEE.

Google (2016). Horizontal pod autoscaler. `https://github.com/kubernetes/kubernetes/blob/release-1.2/docs/design/horizontal-pod-autoscaler.md`. [Acessado em 03/04/2016].

Instruments, N. (2015). Explicando a teoria pid. `http://www.ni.com/white-paper/3782/pt/`. [Acessado em 20/08/2015].

Jain, R. (1990). *The art of computer systems performance analysis: techniques for experimental design, measurement, simulation, and modeling*. John Wiley & Sons.

Jesheen, H. (2016). Auto scaling with docker. `https://botleg.com/stories/auto-scaling-with-docker/`. [Acessado em 29/03/2017].

Kan, C. (2016). Docloud: An elastic cloud platform for web applications based on docker. In *Advanced Communication Technology (ICACT), 2016 18th International Conference on*, pages 478–483. IEEE.

Kettani, H., Gubner, J., et al. (2002). A novel approach to the estimation of the hurst parameter in self-similar traffic. In *Local Computer Networks, 2002. Proceedings. LCN 2002. 27th Annual IEEE Conference on*, pages 160–165. IEEE.

Linux (2016). Man-page namespaces(7). http://man7.org/linux/man-pages/man7/namespaces.7.html. [Acessado em 16/11/2016].

Lorido-Botrán, T., Miguel-Alonso, J., and Lozano, J. A. (2012). Auto-scaling techniques for elastic applications in cloud environments. *Department of Computer Architecture and Technology, University of Basque Country, Tech. Rep. EHU-KAT-IK-09*, 12:2012.

Martin, N. (2015). A brief history of docker containers' overnight success. http://searchservervirtualization.techtarget.com/feature/A-brief-history-of-Docker-Containers-overnight-success. [Acessado em 03/04/2016].

Müller, C., Truong, H.-L., Fernandez, P., Copil, G., Ruiz-Cortés, A., and Dustdar, S. (2016). An elasticity-aware governance platform for cloud service delivery. In *Services Computing (SCC), 2016 IEEE International Conference on*, pages 74–81. IEEE.

Poddar, R., Vishnoi, A., and Mann, V. (2015). Haven: Holistic load balancing and auto scaling in the cloud. In *2015 7th International Conference on Communication Systems and Networks (COMSNETS)*, pages 1–8. IEEE.

RedHat (2016). Resource management guide. https://access.redhat.com/documentation/en-US/Red_Hat_Enterprise_Linux/6/html/Resource_Management_Guide/ch01.html. [Acessado em 16/11/2016].

Rubis (2009). Rubis rice university bidding system. http://rubis.ow2.org/. [Acessado em 03/04/2016].

Schwartz, J. (2014). Are containers the beginning of the end of virtual machines? https://virtualizationreview.com/articles/2014/10/29/containers-virtual-machines-and-docker.aspx. [Acessado em 23/11/2016].

Sefraoui, O., Aissaoui, M., and Eleuldj, M. (2012). Openstack: toward an open-source solution for cloud computing. *International Journal of Computer Applications*, 55(3).

Solis, M. A. P. (2016). An algorithm based on response time and traffic demands to scale containers on a cloud computing system. *he 15th IEEE International Symposium on Network Computing and Applications (NCA 2016)*, 1:343–350.

Squazt (2012). Implementation of the rice university bidding system (rubis). https://github.com/sguazt/RUBiS. [Acessado em 10/11/2015].

Tarreau, W. (2015). Haproxy configuration manual. http://www.haproxy.org/download/1.5/doc/configuration.txt. [Acessado em 03/04/2016].

Thrun, S., Montemerlo, M., Dahlkamp, H., Stavens, D., Aron, A., Diebel, J., Fong, P., Gale, J., Halpenny, M., Hoffmann, G., et al. (2006). Stanley: The robot that won the darpa grand challenge. *Journal of field Robotics*, 23(9):661–692.

VMware (2016). What is virtualization. http://www.vmware.com/solutions/virtualization.html. [Acessado em 16/11/2016].

Walberg, S. (2008). Automate system administration tasks with puppet. *Linux Journal*, 2008(176):5.

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.

Consolidação de MVs em nuvem IaaS orientada por operações elásticas e focada em consumo energético

Denivy B. Rück, Charles C. Miers, Maurício A. Pillon, Guilherme P. Koslovski

[1]Programa de Pós-Graduação em Computação Aplicada (PPGCA)
Universidade do Estado de Santa Catarina (UDESC) - Joinville – SC – Brasil
denivy.ruck@labp2d.udesc.br, {charles.miers, mauricio.pillon, guilherme.koslovski}@udesc.br

Resumo. *O provisionamento de Infraestruturas Virtuais (IVs) elásticas possibilita o gerenciamento dinâmico de recursos computacionais em nuvens. Com a elasticidade, recursos virtuais de computação, armazenamento e comunicação são ajustados aos requisitos da aplicação hospedada (e.g., carga submetida, número de usuários). Para atender as solicitações elásticas dos usuários, provedores especificam políticas de gerenciamento e aplicam mecanismos de realocação. A preocupação com sustentabilidade e com custos variáveis torna o consumo energético de data centers um tema recorrente no gerenciamento de provedores. Sobretudo, reduzir o consumo energético é interessante tanto para usuários quanto para provedores. O algoritmo proposto considera o compartilhamento proporcional do uso de CPU de servidores hospedeiros para calcular o consumo individual, desativando equipamentos ociosos e reorganizando os recursos virtuais. A análise experimental indica uma redução do consumo energético de até* 50%*, sem impactar na taxa de aceitação de novas requisições.*

Abstract. *The elastic provisioning of virtual infrastructures (IVs) enable a dynamic management of cloud resources. Based on elasticity, virtual computing, storage, and communication the resources are adjusted to fit on hosted application requirements. In order to accomplish elasticity requests, providers rely on reallocation mechanisms and policies. The concern, regarding sustentability and operational costs, turns the data center power consumption a recurring theme in provider policies. Moreover, power-aware provisioning is beneficial for both tenants and providers. Thus, we propose an algorithm considering the proportional sharing of CPU usage of data center servers in order to calculate individual costs, disable idle equipments, and reallocate virtualized resources. The experimental analysis indicates a reduction of energy consumption near to* 50% *without impacting the acceptance ratio of new requests.*

Introdução

Os benefícios administrativos e econômicos oferecidos, tanto para provedores quanto para clientes, garantiram o êxito das nuvens IaaS (*Infrastructure as a Service*). Com este modelo, provedores podem oferecer a capacidade computacional ociosa dos seus *data centers* na forma de Infraestruturas Virtuais (IVs), diminuindo custos administrativos. Uma IV é composta por recursos virtuais de processamento e comunicação. Clientes de IaaS contratam IVs para diversas finalidades, sem a preocupação com qualquer tipo de manutenção nos recursos, totalmente a cargo do provedor [Mell and Grance 2011].

Um importante aspecto em nuvens computacionais é o provisionamento elástico de IVs. A variação na carga de processamento das aplicações hospedadas requer mecanismos de elasticidade que consigam ampliar, reduzir e reoganizar, em número e configuração, os recursos já alocados, respeitando o *Service Level Agreement* (SLA) estabelecido [Righi et al. 2015]. Ao invés de provisionar recursos de maneira estática, gerando desperdícios por provisionar sempre para o pior caso, o provisionamento elástico garante a quantidade de recursos necessária para a execução da carga de trabalho do momento.

Dentre as tarefas administrativas efetuadas para provisionar uma IV, destaca-se a utilização de algoritmos de alocação e realocação de recursos. O primeiro analisa a capacidade atual do *data center* para encontrar os componentes que melhor hospedem a IV requisitada [Oliveira and Koslovski 2015]. O segundo, adapta os recursos elásticos considerando todas as IVs já alocadas no *data center*, buscando a sua reorganização, quando necessária, para atender as requisições de elasticidade. Ambos os problemas são de categoria NP-Difícil [Zhu and Ammar 2006]. Tanto a alocação quanto a realocação de recursos são guiadas por políticas que representam o objetivo do provedor IaaS. Atualmente, é crescente a preocupação com o consumo energético dos servidores, visto que implica diretamente em custos administrativos bem como em aspectos de sustentabilidade. Estudos indicam que *data centers* nos Estados Unidos foram responsáveis por 3% do total de energia consumida em 2011 [Duan et al. 2016], com previsão de crescimento. Este consumo é principalmente associado ao resfriamento dos servidores, que pode chegar a 53% do seu total [Zhang et al. 2010].

Diante do exposto, neste trabalho, segue-se o princípio que provedores podem provisionar IVs elásticas simultaneamente reduzindo o consumo energético dos seus *data centers*. Entretanto, a imprevisibilidade de chegada e a definição não determinística das requisições elásticas são fatores críticos para provedores IaaS que buscam a diminuição do consumo energético. Uma simples requisição pode desbalancear um *data center*, criando gargalos de comunicação ou aumentando desnecessariamente o número de servidores ativos. As abordagens comumente aplicadas para consolidação de servidores, muitas vezes desrespeitam o SLA estabelecido com os clientes [Buyya et al. 2010, Duan et al. 2016]. Ainda, um estudo recente [Hinz et al. 2016] indicou que o consumo energético de CPUs pode ser utilizado para efetuar um rateamento mais justo do custo de provisionamento de servidores em nuvens IaaS. O custo individualizado de processamento (CPUs virtuais e operações de rede) das máquinas virtuais (MVs) é combinado com um custo proporcional das atividades de gerenciamento (operações dos hipervisores de MVs) para composição de um custo final, atribuindo somente a fração justa identificada.

Em suma, o algoritmo de realocação de IVs elásticas proposto neste trabalho considera o compartilhamento proporcional do uso de CPU de servidores hospedeiros para calcular o consumo individual, podendo desativar equipamentos ociosos e reorganizar os recursos virtualizados. O mecanismo tem como premissa o modelo de compartilhamento de custo proporcional [Hinz et al. 2016] pelo qual conclui-se que o aumento do gasto energético, na perspectiva do provedor, está diretamente ligado não somente ao número de servidores ativos para atender as IVs provisionadas, mas também a carga de processamento das MVs. A proposta é inovadora ao considerar o compartilhamento proporcional do uso de CPU das MVs, realizando uma consolidação de servidores que, além de atender ao SLA estabelecido, contabiliza a fração justa atribuída a cada cliente. Os resultados

experimentais apontam uma redução no consumo de energia sem diminuir a taxa de aceitação de novas requisições.

O restante deste trabalho está organizado da seguinte forma: a Seção 2 discute o provisionamento de IVs elásticas. A Seção 3 apresenta os trabalhos relacionados, enquanto a Seção 4 descreve a solução proposta. A análise experimental é discutida na Seção 5. Por fim, a Seção 6 apresenta as considerações finais.

Provisionamento de IVs Elásticas

O cenário no qual o mecanismo proposto é aplicado constitui-se de um *data center* contendo diversas requisições de IVs já alocadas. O provedor que hospeda este *data center* recebe, ocasionalmente, requisições por parte de seus clientes, tanto para alocação quanto para elasticidade de recursos.

Mecanismos para Provisionamento de Elasticidade

Tanto inquilinos quanto provedores usufruem dos benefícios da elasticidade. Para o inquilino, tais técnicas são usadas para controlar o desempenho da aplicação hospedada, mesmo na ocorrência de variações da carga de trabalho. No lado do provedor, a elasticidade pode ser utilizada para consolidar MVs em servidores, aumentando a taxa de aceitação de novas requisições de IVs.

As requisições de elasticidade provenientes de inquilinos podem ser de quatro tipos [Righi 2013]: *(i)* liberação, representando o desprovisionamento de um recurso virtual; *(ii)* redimensionamento, em que as capacidades virtuais são aumentadas ou diminuídas; *(iii)* replicação, em que um novo recurso virtual é criado a partir de um recurso base, comumente utilizado em contextos de balanceamento de carga de aplicações [Righi et al. 2015] e; *(iv)* migração, no qual todo o conteúdo de uma MV é transferido para outro hospedeiro. Este último mecanismo é favorável em tarefas administrativas, aplicado para reorganizar os recursos virtuais existintes. Ainda, requisições elásticas de redimensionamento e replicação podem ser convertidas em requisições de migração em cenários em que a capacidade residual do hospedeiro não for suficiente para o seu provisionamento. O redimensionamento é a abordagem mais indicada por ser a menos custosa em relação ao tempo de adaptação [Sharma et al. 2011], requisitando o aumento ou diminuição da configuração (*e.g.*, CPU, RAM, largura de banda).

Requisição de Provisionamento Elástico

Ao requisitar uma IV, os clientes estabelecem um SLA com o provedor. Através dos SLIs (*Service Level Indicators*), o SLA estabelece características que devem ser respeitadas, tais como confiabilidade, custos, desempenho, segurança, localização geográfica, entre outras [Sauvé et al. 2005]. De forma complementar, o SLO (*Service Level Objective*) é responsável por representar estas restrições, permitindo uma forma de quantificá-las em termos do SLI. O SLO mede o desempenho do provedor de serviço de acordo com o SLA, trabalhando com métricas como vazão, disponibilidade, tempo de resposta e qualidade [Righi 2013]. Ou seja, caso ocorram picos de execução, algumas características do SLO podem ser violadas, tais como a disponibilidade do serviço ou o tempo de resposta.

Para que a qualidade do serviço hospedado na IV continue satisfatória, é necessária a aplicação de mecanismos de elasticidade, respeitando outras restrições que possam

estar presentes no SLA. No presente trabalho, além das requisições tradicionais de configuração dos recursos (vCPU, memória, armazenamento e largura de banda), o SLA inclui a definição do tempo máximo permitido para a migração de um recurso virtual. A definição foi incluída pois é de conhecimento comum que a migração de MVs induz a uma sobrecarga na aplicação hospedada [Akoush et al. 2010, Strunk 2012].

O inquilino/usuário tem liberdade para solicitar a reconfiguração da IV apoiado em mecanismos de monitoração e controle [Simões and Kamienski 2014, Pfitscher et al. 2013, Darolt et al. 2016]. Nos provedores de nuvens IaaS atuais, tais como *Amazon EC2*, *Microsoft Azure* e *Google Computing Engine*, a reconfiguração é realizada reativamente, respondendo ao SLI. O algoritmo proposto neste trabalho é agnóstico às métricas e ao mecanismo de monitoração e controle selecionado pelo inquilino.

Trabalhos Relacionados

A literatura especializada relacionada com o escopo do presente trabalho compreende técnicas para alocação e realocação de IVs, mecanismos para provisionamento de elasticidade e políticas guiadas pela eficiência energética.

Alocação e Realocação de IVs

Os algoritmos para alocação de IVs devem encontrar uma solução de mapeamento em tempo computacional aceitável, atendendo aos requisitos definidos no SLA e aos objetivos dos provedores. Oliveira *et al.*(2015) propuseram um algoritmo heurístico baseado em árvores para diminuir o tempo necessário para alocação, denominado VITreeM. Por sua vez, as soluções D-ViNE e R-ViNE [Chowdhury et al. 2009] buscaram a alocação com foco na garantia da rota mais curta entre os recursos virtuais. O mecanismo proposto na Seção 4 é agnóstico ao algoritmo de alocação utilizado. Ou seja, dentre as diversas soluções existentes [Fischer et al. 2013, Gong et al. 2014], o provedor pode selecionar o mecanismo que melhor representar os seus objetivos. Baseado na comparação realizada entre algumas abordagens [Oliveira and Koslovski 2016], VITreeM foi selecionado para compor o cenário experimental (Seção 5).

Quanto a realocação de IVs, um algoritmo destinado ao balanceamento de carga do *data center*, priorizando requisições que resultem em maior lucro, foi investigado em [Zhu and Ammar 2006], enquanto [Duan et al. 2016] propôs um algoritmo com mecanismo de predição baseado em fractais e utilizando *Ant Colony Optimization* como estratégia de realocação de modo a reduzir o custo energético. No presente trabalho (Seção 4), a redução do consumo energético do *data center* é viabilizada através da consolidação de MVs guiadas pelo princípio da elasticidade, prioritário na decisão. No SLA, além das definições tradicionais, o inquilino define o tempo máximo aceitável para uma migração (Seção 4.3), que representa o limiar de consolidação que o provedor pode aplicar.

Mecanismos para Provisionamento Elástico

Tendo como principal motivação a diminuição do custo da IV alocada por parte do inquilino, Sharma *et al.* (2011) propuseram um método de elasticidade baseado em programação linear . Quando uma requisição de elasticidade é submetida, o provedor seleciona as instâncias de MVs com menor custo para provisionamento. Por fim, o mecanismo aplica migrações, para consolidação do *data center*, e replicações para balanceamento de carga nas aplicações. Por sua vez, Darolt e seus colegas exploraram a elasticidade dos re-

cursos que compõem uma IV para otimizar o desempenho de aplicações *n-layers* [Darolt et al. 2016]. Foi observada a diminuição do tempo médio de processamento e do custo de provisionamento para o inquilino ao hospedar a aplicação em uma IV elástica com abordagem de reconfiguração reativa. A presente proposta é independente dos mecanismos para controle de limiares.

O provisionamento elástico de recursos com foco em processamento de alto desempenho foi proposto em [Righi et al. 2015]. O mecanismo provisiona MVs sempre que o desempenho da aplicação está aquém do esperado. Entretanto, inova ao antecipar a realização das solicitações de elasticidade, diminuindo o tempo de espera pela reconfiguração do serviço. O mecanismo proposto (Seção 4) pode ser incorporado a solução para reorganizar o *data center*, tendo potencial para reduzir o consumo de energia.

Eficiência Energética

Alguns fatores relacionados com o impacto do provisionamento elástico em recursos de nuvens já foram investigados [Assuncao et al. 2016]. As simulações elucidaram que abordagens tradicionais para desativação de servidores devem ser substituídas por estratégias de reserva antecipada. Seguindo essa linha, a Seção 5 aborda a aplicação do mecanismo proposto em cenários com diferentes configurações de elasticidade e requisitos de migração. Quanto ao cálculo do custo de provisionamento guiado pelo consumo de energia ocasionado por processadores, Hinz *et al.* (2016) apresentaram um modelo para distribuição proporcional. Assim, inquilinos são tarifados de acordo com o seu consumo real, enquanto provedores são responsáveis pelos custos adjacentes. O modelo inovou o cálculo tradicionalmente aplicado por provedores públicos, e consequentemente foi incorporado ao algoritmo descrito na Seção 4. Por fim, é importante ressaltar que o diferencial deste artigo é a inserção do atendimento de requisições de elasticidade junto com o procedimento de realocação de IVs com foco na redução do consumo de energia, não encontrado em nenhum dos outros trabalhos.

EAVIRA

O algoritmo proposto, denominado *Energy-Aware Virtual Infrastructure Realocation Algorithm* (EAVIRA), objetiva o atendimento de requisições de reconfiguração de IVs, ao mesmo tempo que busca diminuir o consumo energético do *data center*. Entretanto, este objetivo necessita ser alcançado respeitando o SLA contratado pelo cliente que, neste contexto, adiciona o tempo máximo aceitável para a migração de um recurso como uma restrição. EAVIRA consiste nas etapas de: *(i)* definição da infraestrutura base; *(ii)* processamento das requisições de elasticidade; e *(iii)* realocação das IVs no *data center*. As etapas são discutidas nas próximas subseções.

Definição da Infraestrutura Base

O ponto inicial do algoritmo é a obtenção de uma infraestrutura base (IB) para as migrações. Visto que a realocação é um problema NP-Difícil, essa infraestrutura representa apenas uma fração do *data center*, diminuindo o número de servidores candidatos para cálculo das possibilidades de realocação. Assim, somente os recursos participantes de uma IB são candidatos a receberem a migração de recursos virtuais. Para fins de comparação durante a análise experimental (Seção 5), propõem-se dois tipos de infraestruturas bases, denominadas IB-W e IB-SLA. A primeira é definida como a infraestrutura física

que hospeda a IV de maior peso no *data center*. A sua escolha é realizada pelo cálculo do custo de provisionamento. Considerando que uma IV é composta por um conjunto de MVs (R) interconectadas por um conjunto de enlaces virtuais (E), o custo de provisionamento $C(IV)$ é definido por $C(IV) = \sum_{i \in R} c(i) + \sum_{l \in E} c(l) \times len(p)$, sendo $c(.)$ a representação da capacidade virtual solicitada de um recurso. Um enlace virtual l é hospedado sobre um caminho físico p no *data center*. Em suma, o custo é dado pelo somatório das capacidades provisionadas para cada recurso virtual da IV. A hipótese para esta abordagem é que a infraestrutura de maior impacto no *data center* corresponde a de maior dificuldade de realocação. Assim, a IB é composta pelos servidores e enlaces compromissados com o provisionamento da IV de maior custo (IB-W).

A segunda alternativa para composição de uma IB é realizada pelas informações do SLA (IB-SLA). Como o EAVIRA busca a consolidação de MVs para desativar servidores ociosos, MVs com restrição de migração (representado por $t_{mig} = 0$) inviabilizam o desligamento dos servidores que as hospedam. Restringir o número de servidores candidatos a desativação pode representar um ponto crítico na reorganização de IVs e, consequentemente, um menor ganho na redução do consumo energético do *data center*. Assim, a IB-SLA é composta por todos os servidores físicos que hospedem ao menos uma MV com esta restrição de migração.

Para reconhecer alterações ocorridas no *data center* (*e.g.*, chegada de novas requisições, reconfigurações), a definição da IB ocorre a cada execução do algoritmo. Entretanto, as opções de IB são aplicadas separadamente, ou seja, uma escolha do provedor.

Processamento das Requisições Elásticas

Após a definição da IB, EAVIRA processa as requisições de desalocações, redimensionamentos, replicações e migrações, nesta ordem. A ordem da aplicação dos mecanismos foi definida com base no seu custo de realização [Galante and de Bona 2012]. A desalocação oportuniza novas possibilidades para realocação e consolidação. Dentre os três mecanismos de elasticidade, o redimensionamento é o que tem menor custo computacional, seguido da replicação e da migração [Sharma et al. 2011].

Desalocação de IVs. Consiste na liberação da capacidade de servidores e enlaces previamente reservada para hospedar as IVs.

Redimensionamento de Recursos Virtuais. EAVIRA considera o aumento ou diminuição da configuração de uma MV (CPU, RAM, etc) como uma operação de redimensionamento. Entretanto, caso a capacidade disponível no servidor que hospeda a MV a ser redimensionada não seja suficiente para atender a nova configuração, a operação é transformada em uma solicitação de migração, movendo a MV para um novo hospedeiro.

Replicação de Recursos Virtuais A replicação é um método comum quando a finalidade é o balanceamento de carga de aplicações compostas por um despachante de requisições, que contém a visão de todas as réplicas disponíveis, e que decide qual réplica atenderá uma nova requisição do usuário [Righi 2013] [Righi et al. 2015] [Darolt et al. 2016]. Em suma, é um processo que consiste na cópia da imagem de uma MV para o servidor de destino. Após a cópia, o sistema operacional hospedado e as aplicações são iniciados, ingressando no conjunto de recursos disponíveis. O tempo total para replicação ($t(i)$) de uma MV i pode ser calculado por $t(i) = \frac{D(i)}{b} + \beta$, sendo $D(i)$ o tamanho da imagem do sistema, b a largura de banda disponível para mover a MV até o hospedeiro destino e β

uma constante representando o tempo necessário para inicialização do sistema operacional no equipamento de destino [Sharma et al. 2011].

EAVIRA opta por replicar a MV no mesmo servidor hospedeiro, visto que a imagem do sistema já está disponível localmente. Quando o servidor não possui capacidade ociosa, o mecanismo seleciona os hospedeiros da IB como candidatos a receber o novo recurso replicado. Caso não tenha êxito, a última possibilidade de atendimento da solicitação é a replicação em um servidor disponível no *data center* que ainda não pertença a IB. Em caso de sucesso, este servidor será incorporado a IB, caso contrário, a solicitação não será atendida. O hospedeiro escolhido deve deixar as menores lacunas de recursos ociosos (abordagem *best-fit*) alinhada com a consolidação de recursos [Ruck et al. 2014].

Realocação das IVs no *Data center*

Após as etapas de desalocações, redimensionamentos e replicações requisitadas pelos inquilinos, inicia-se a fase de sucessíveis migrações de IVs no *data center* buscando a consolidação dos servidores.

Representação da Eficiência Energética. EAVIRA busca diminuir o número de servidores ativos, bem como realizar a quantificação proporcional do custo guiado pelo consumo das MVs. Conforme representado pela Figura 1, o consumo energético de um servidor pode ser proporcionalmente compartilhado entre as MVs. O consumo total é composto por *(i)* uma fração mínima, necessária para manter o equipamento ativo; *(ii)* uma fração atribuída as operações de gerenciamento das MVs, efetuadas pelo hipervisores; *(iii)* a representação das operações de rede efetuadas pelas MVs; e *(iv)* o consumo efetivamente oriundo do processamento da carga de trabalho nas MVs. Como os itens *(iii)* e *(iv)* dependem da carga de trabalho aplicada, o provedor busca diminuir sua responsabilidade sobre as frações *(i)* e *(ii)*. Quando não existem MVs hospedadas em um servidor, as frações *(i)* e *(ii)* são atribuídas ao provedor.

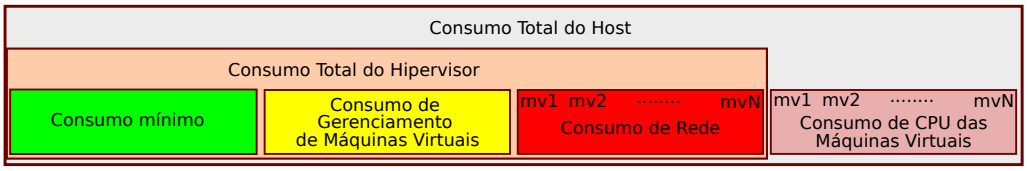


Figura 1. Compartilhamento proporcional do consumo energético de um servidor hospedeiro [Hinz et al. 2016].

Ao atender uma solicitação de redimensionamento de MV localmente, o provedor IaaS percebe a consolidação de seus servidores. O mesmo raciocínio é aplicado as solicitações de replicação: quando a operação é atendida localmente, as frações *(i)* e *(ii)* são diluídas entre os inquilinos. Entretanto, o escalonamento de uma migração para atender uma requisição de elasticidade representa um ponto crítico pois potencialmente um novo servidor será ativado. Embora EAVIRA busque a realocação com foco na diminuição do consumo energético, as operações de elasticidade são priorizadas para cumprir o SLA.

Estimando o Tempo Total de uma Migração de MV. Dentre os desafios para implementação da migração de MVs, a correta predição do tempo total de migração desponta como uma tarefa complexa. Existem vários parâmetros e métricas aplicáveis para essa predição, tais como a vazão do canal disponível, número de páginas modificadas e sobrecarga pré e pós migração [Akoush et al. 2010]. Neste cenário, [Strunk 2012] estabelece

um algoritmo para se obter uma estimativa do tempo de migração baseado no mecanismo de *live-migration* [Clark et al. 2005], descrito no Algoritmo 1.

Algoritmo 1: Estimativa do tempo de migração [Strunk 2012].

input : $m_{th}, v_{th}, p_{th}, v_{mem}, b, d, l$
output: Tempo de migração

```
1  let v_0 = v_mem, v_mig = 0, t_mig = 0, t_down = 0;
2  for i = 0; i < m_th do
3  |    t_i = v_i / b;
4  |    v_{i+1} = t_i * d * l;
5  |    t_mig = t_mig + t_i;
6  |    v_mig = v_mig + v_i;
7  |    if v_mig > v_th ∨ (v_{i+1}/l) < p_th then
8  |    |    break;
9  t_down = v_{i+1} / b;
10 t_mig = t_mig + t_down;
11 return t_mig;
```

O número máximo de iterações realizadas pelo algoritmo é dado por m_{th}, enquanto v_{th} indica a quantidade máxima de memória para ser migrada a cada iteração. Por sua vez, p_{th} representa a quantidade máxima aceitável de páginas sujas para uma iteração, v_{mem} a memória provisionada para a MV, d o número de páginas sujas no momento da execução do algoritmo e l o tamanho de cada página. Quanto a configuração da rede do *data center*, b representa a largura de banda disponível para migração. Iterativamente, o algoritmo quantifica a porção de memória que deve ser copiada nas iterações futuras (linhas 3 e 4). Se os limiares de páginas sujas ou memória máxima forem ultrapassados (linha 7), o algoritmo passa para a fase de *stop-and-copy* (linha 9), em que a MV encerra sua execução e o restante da memória é copiada para o servidor de destino.

Algoritmo e Exemplo de Realocação. Para exemplificar as migrações executadas por EAVIRA, a Figura 2 retrata 5 migrações realizadas com sucesso, desativando os servidores ociosos 7, 9 e 10.

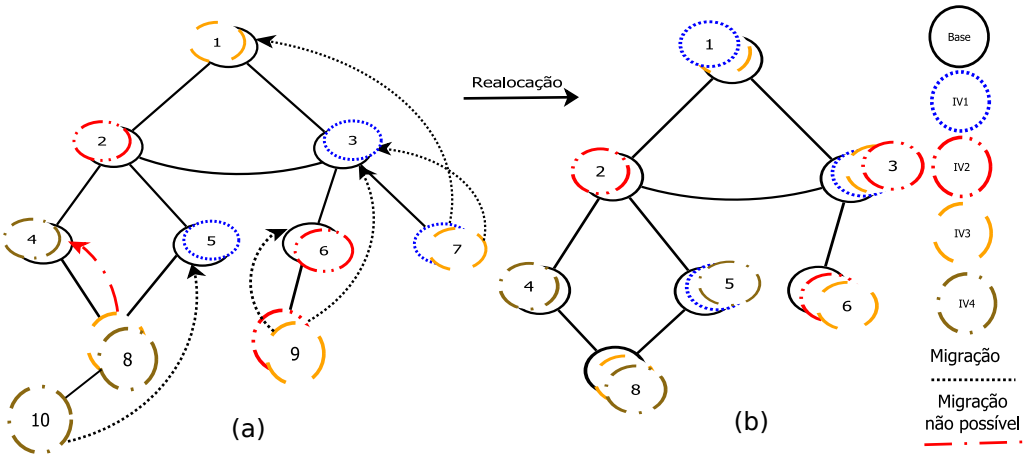


Figura 2. Cenário de realocação de IVs.

As arestas tracejadas indicam a ocorrência de migrações. Após o sucesso de uma migração, os enlaces virtuais são realocados para o caminho mais curto entre os recursos conectados à MV migrada, diminuindo o custo de provisionamento (discutido na Seção 4.1). No caso de impossibilidade na migração, representado pela tentativa com a MV pertencente à IV3 no hospedeiro 8, a IB será expandida para comportar este nó (grafo Figura 2(b)). Na próxima iteração o hospedeiro 8 poderá ser alvo de outras migrações. Além disso, as migrações das MVs de um nó são realizadas em conjunto, ou seja, basta

que uma migração seja impossibilitada para que a IB seja expandida. O procedimento será repetido para todas as MVs ainda não analisadas (busca em largura). O Algoritmo 2 representa o mecanismo implementado na forma de pseudo-código.

Algoritmo 2: Etapa de realocação

input : $BI(V, E)$
output: Número de migrações realizadas

```
1  resources = get_resources(BI);
2  nodes = {∀neighb; neighb ∈ adj(node) ∧ node ∈ resources};
3  visited = {};
4  total_migrations = 0;
5  for ∀node; node ∈ nodes do
6      nodes = nodes − {node};
7      visited = visited ∪ {node};
8      resources = get_resources(BI);
9      nodes = nodes ∪ {∀neighb; neighb ∈ adj(node) ∧ neighb ∉ resources ∧ neighb ∉
         nodes ∧ neighb ∉ visited};
10     rollback_list = {};
11     migrations = 0;
12     for ∀vnode ∈ virtual_resources(node) do
13         resources = get_resources(BI, vnode.type);
14         if ¬migrate(vnode, resources) then
15             failed = True;
16             break;
17         else
18             migrations = migrations + 1;
19             rollback_list = rollback_list ∪ {vnode};
20     if failed then
21         rollback(rollback_list);
22         rollback_list = {};
23     else
24         total_migrations = total_migrations + migrations;
25 return total_migrations;
```

As linhas 1, 2 e 3 inicializam as listas iniciais da busca em largura a partir das folhas da IB. Os novos nós são inseridos iterativamente conforme os forem visitados (linha 9). Para cada nó pertencente a esta lista (linha 5), busca-se a migração de todos os seus recursos virtuais (linha 12-19). Em caso de falha, todo o processo é desfeito para o servidor em análise (linhas 20-22). Ao final do algoritmo (linha 25), é retornado o número de migrações realizadas nesta iteração. É importante ressaltar que em tempo de análise de possibilidades, nenhuma migração é efetivada, apenas analisada.

Ao término da execução, a fração ativa de recursos no *data center* será menor ou igual à fração anterior, visto que, no pior caso, a infraestrutura física não sofrerá alterações, e, no melhor caso, o número de servidores ativos será reduzido por conta das consolidações. Havendo consolidações, mantém-se a mesma quantidade de recursos virtuais alocados, porém, tem-se economia de energia com o desligamento de recursos físicos.

Em resumo, para que a migração seja possível, é necessário que: *(i)* a restrição de tempo máximo para a migração do SLA contratado seja respeitada; *(ii)* exista um hospedeiro de destino com recursos disponíveis; e *(iii)* todos os recursos virtuais conectados à MV migrada tenham seus enlaces reconectados para o hospedeiro destino. Caso contrário, a migração não é realizada. EAVIRA propõe o atendimento de requisições elásticas reduzindo o consumo energético do *data center*.

Análise Experimental

A análise experimental segue o plano de testes resumido na Tabela 1. EAVIRA e um simulador de eventos discretos foram implementados em Python[1]. Para compor o cenário experimental, o algoritmo VITreeM [Oliveira and Koslovski 2015], discutido na Seção 3, foi selecionado para alocação *online* de requisições. O estudo com outros mecanismos é indicado como trabalho futuro. Os testes apresentam os resultados com VITreeM (sem elasticidade) e com EAVIRA (elasticidade e foco na redução do consumo energético).

Tabela 1. Configurações utilizadas nos experimentos.

Algoritmos	VITreeM – EAVIRA
Infraestrutura de base	IB-W – IB-SLA
Seleção dos Candidatos	best-fit – worst-fit
Requisições de IVs	fat-tree – n-layers – VPC – mixed
Recursos virtuais restritos (%)	10% – 40% – 80%

As métricas escolhidas para identificar o impacto da política de redução de energia são: *(i)* número de operações, contabilizando as operações em IVs (alocação e desalocação) e de recursos virtuais (desalocação, reconfiguração, replicação e migração); *(ii)* taxa de aceitação (TA), representando o percentual de IVs alocadas com sucesso; e *(iii)* consumo de energia, calculado a partir do modelo de custos de [Hinz et al. 2016]. Segundo as características dos algoritmos VITreeM e EAVIRA, foram identificados os quatro principais elementos (Tabela 1) que impactam no consumo de energia de um *data center*.

O *data center* modelado é constituído de servidores idênticos, totalizando 432, interconectados por 90 switches, seguindo padrões de projeto da Cisco [Cisco 2007]. Optou-se por um *data center* de grande porte para evidenciar melhor os aspectos de consolidação da proposta, visto que quanto menor o *data center* ficam mais restritas as operações de consolidação. Cada servidor é composto por dois processadores com 12 núcleos cada, 256GB de memória, 1TB de armazenamento e uma interface de rede de 1Gbps. A relação de carga de processamento versus consumo de energia destes servidores foram obtidos em [Hinz et al. 2016]. O provedor de IaaS simulado não dispõe de locação por pacotes de recursos, deixando livre ao seu inquilino a composição de suas IVs. Portanto, definiu-se limiares individuais para cada recurso, sem vínculo entre eles. Dentre estes limiares, todas as combinações são possíveis. Assim, as IVs foram compostas por vCPUs (de 1 a 8), RAM (de 1 a 32GB), capacidade de armazenamento (de 8 a 128), enlaces de redes (de 1 a 100Mbps) e tempo de SLA para migração (de 1 a 600s).

Por definição, o EAVIRA inicia seu provisionamento identificando a IB (Seção 4.1) que servirá como alvo para as requisições elásticas. A escolha da IB afeta diretamente o número total de nós físicos utilizados após o atendimento de todas as requisições do período. Na Tabela 1, a característica infraestrutura de base pode ser IB-SLA ou IB-W, indicando o critério de escolha. O terceiro componente da Tabela 1 trata da forma de seleção dos candidatos, *best-fit* ou *worst-fit*. Dentre um conjunto de solicitações, a configuração *best-fit* seleciona o candidato que deixará o menor residual de recurso disponível, após seu provisionamento, e, o *worst-fit*, escolhe aquele que deixará o maior residual. O tipo de requisição de IVs especifica a topologia escolhida. Foram realizados testes com quatro categorias: *fat-tree*, *Virtual Private Cloud (VPC)*, *n-layers* e *mixed*. Requisições do tipo *fat-tree* são topologias que organizam os *switches* em camadas, sendo os servidores virtuais posicionados nas folhas da árvores. É uma topologia de propósito geral que

[1]Disponível em: https://bitbucket.org/denivyruck/eavira

utiliza de maneira eficiente os recursos de comunicação oferecidos [Al-Fares et al. 2008]. Recentemente, a Amazon[2] passou a oferecer a topologia VPC, constituída de uma rede virtual controlada completamente pelo próprio inquilino. O terceiro tipo, as topologias *n-layers*, foram incluídas por representarem aplicações que operam em diferentes camadas, tais como aplicações *web* [Darolt et al. 2016]. Por fim, *mixed* é constituído de um conjunto de IVs cujo as topologias podem ser *fat-tree*, VPC e/ou *n-layers*.

O último componente dos experimentos descreve o percentual de recursos virtuais restritos dentre o conjunto de solicitações de realocação de IVs requisitadas. Quanto maior o percentual de restrição, maior o número de MVs que não admitem migração ($t_{mig} = 0$, Seção 4) e menor o grau de maleabilidade na alocação de recursos. Foram feitos testes com percentuais de restrição em 10%, 40% e 80%, porém, neste trabalho, escolheu-se por discutir somente os resultados do pior caso (80%). Os gráficos com os resultados com 10% e 40% encontram-se disponíveis no repositório do EAVIRA.

Análise dos Resultados

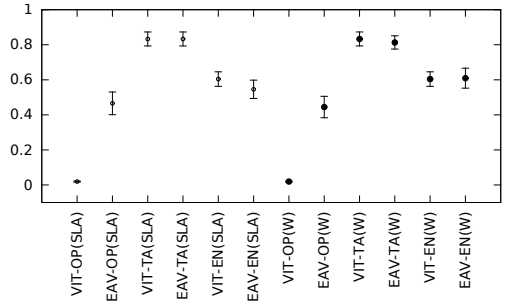
O conjunto de resultados obtidos é oriundo do cruzamento das características descritas na Tabela 1. Utilizou-se, simultaneamente, um total de 76 servidores da plataforma *Grid'5000*[3]. Os gráficos apresentados na Figura 3 correspondem aos resultados com o percentual de recursos virtuais restritos em 80%. A amostra tem tamanho 100, sendo que os pontos são as médias dos valores e as barras o desvio padrão com um intervalo de confiança de 95%. Cada gráfico possui os valores normalizados por métrica, iniciando por número de operações com VITreeM, representado por VIT-OP, seguido pelo número de operações do EAVIRA (EAV-OP). A mesma representação e ordem foi utilizada para as demais métricas, taxa de aceitação (TA) e consumo de energia (EN), respectivamente identificadas por VIT-TA, EAV-TA, VIT-EN e EAV-EN.

Quanto a análise das métricas, para OP, espera-se quantificar o número de operações elásticas efetuadas pelo EAVIRA, tomando como base o número mínimo necessário de operações efetuadas pelo VITreeM. Para TA, quanto maior o valor, melhor, tendo em vista que este corresponde a um melhor aproveitamento dos recursos físicos. Para EN, deseja-se a redução do consumo, portanto valores menores são buscados. As primeiras seis barras de cada gráfico correspondem aos resultados com a infraestrutura de base IB-SLA e as demais com IB-W. Finalmente, os gráficos da esquerda descrevem os resultados com o algoritmo de seleção de candidatos *best-fit* e os da direita com *worst-fit*.

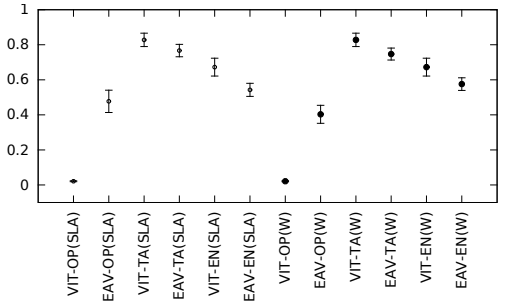
Na Figura 3 pode-se observar que VIT-OP, por representar a linha de base para comparações, possui, para todos os dezesseis casos analisados, média igual a zero. Ao comparar os valores de VIT-OP e EAV-OP, pode-se concluir que o número de operações elásticas efetuadas pelo EAVIRA varia entre 40% e 60%, independente do tipo de requisição, da escolha da infraestrutura de base ou da forma de seleção dos candidatos. Os resultados permitem ainda concluir que a escolha entre IB-SLA ou IB-W, para o pior caso de 80% de recursos restritos, gera pouco impacto nos resultados. A diferença dos valores obtidos com IB-SLA (primeiras 6 colunas) comparados com o valor correspondente de IB-W (últimas 6 colunas) são inferiores a 10%.

[2]https://aws.amazon.com/vpc/

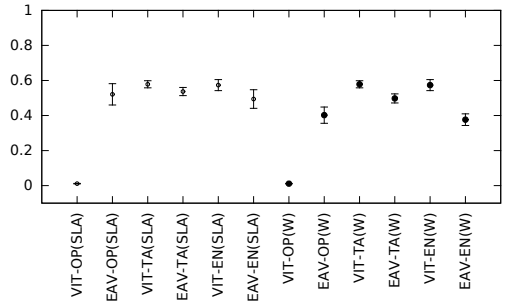
[3]http://www.grid5000.fr



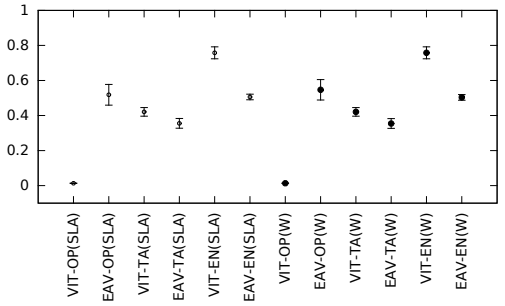
(a) Requisições de IV *fat-tree* (*best-fit*)



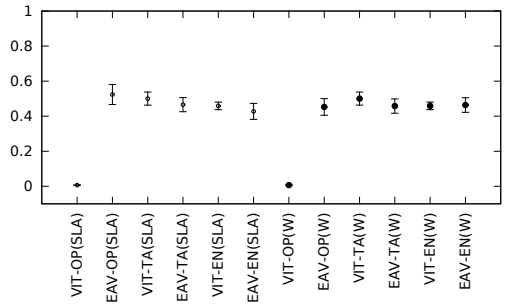
(b) Requisições de IV *fat-tree* (*worst-fit*)



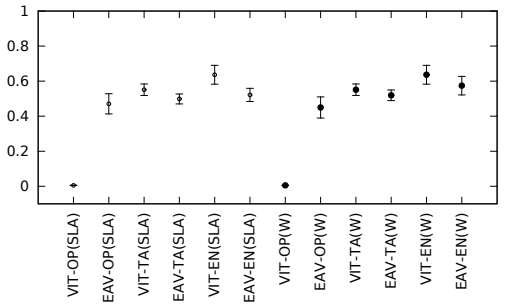
(c) Requisições de IV *mixed* (*best-fit*)



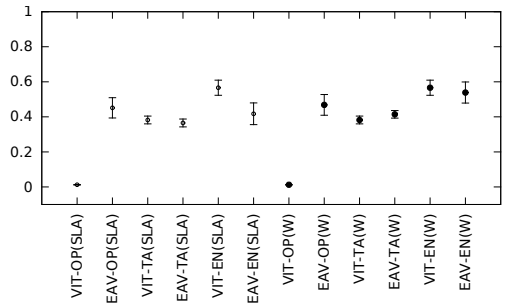
(d) Requisições de IV textitmixed (*worst-fit*)



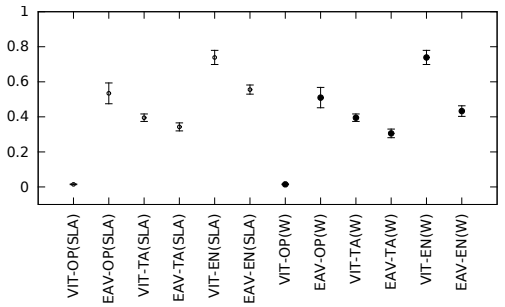
(e) Requisições de IV *n-layers* (*best-fit*)



(f) Requisições de IV *n-layers* (*worst-fit*)



(g) Requisições de IV VPC (*best-fit*)



(h) Requisições de IV VPC (*worst-fit*)

Figura 3. Resultados normalizados das métricas número de operações (OP), taxa de aceitação (TA) e consumo de energia (EN) obtidas nas infraestruturas de bases, IB-SLA (SLA) e IB-W (W). Nestes testes, 80**% das IVs não permitem migração, caso que reduz a possibilidade de realocação.**

Quanto a taxa de aceitação (TA), o VITreeM e o EAVIRA são equivalentes em todos os casos. Ou seja, o atendimento das requisições elásticas não gera prejuízos ao provedor. No entanto, o impacto do tipo de requisição na TA é evidente, enquanto *fat-tree* atinge taxas próximas a 80%, melhor resultado, VPC obtém aproximadamente 40%. Por fim, para a métrica de consumo de energia (EN), cujo objetivo é a redução dos valores consumidos, o EAVIRA é igual ou inferior ao VITreeM. Os consumos são equivalentes em 7 dos 16 casos, sendo 5 deles nos resultados com *best-fit* (gráficos a esquerda). O EAVIRA tem consumo inferior ao VITreeM com *best-fit* para requisições *mixed*/IB-W e VPC/IB-SLA. A redução de consumo de energia com *mixed*/IB-W(Figura 3(c)) foi próximo a 20%. Com *worst-fit* (gráficos a direita), o EAVIRA surpreende obtendo resultados positivos em 7 dos 8 casos analisados, ocasionando uma única equivalência, com requisições *mixed*/IB-W. Pode-se ainda destacar o gráfico da Figura 3(h) com uma diferença de 50%. A redução oportunizada por EAVIRA é o resultado das consolidações de servidores, combinada com o cálculo proporcional de custo energético.

Os resultados indicam que o algoritmo EAVIRA flexibiliza o provisionamento de recursos com operações elásticas, beneficiando provedores e inquilinos, e ainda permite a redução dos custos com energia elétrica. A redução do consumo de energia, sem prejuízo a outras métricas, é de interesse do provedores, de inquilinos e da própria sociedade (sustentabilidade). No que se refere ao objeto principal de estudo, o consumo de energia, o EAVIRA é no mínimo equivalente ao VITreeM e, no melhor caso, 50% mais eficiente.

Considerações & Trabalhos futuros

Em nuvens computacionais, a elasticidade é uma característica essencial, tanto para provedores quanto para inquilinos. Enquanto provedores apoiam-se em operações de realocação de IVs para gerenciar a estrutura física, inquilinos as usam para adequar a disponibilidade de recursos virtuais as suas necessidades. A alocação e realocação de recursos são problemas NP-Difícil [Zhu and Ammar 2006]. Considerando os custos administrativos e o impacto ambiental, a redução do consumo energético de *data centers* é uma preocupação de provedores, de inquilinos e da sociedade. Neste contexto, este trabalho propõe um algoritmo de realocação de IVs elásticas, o EAVIRA, com foco na redução do consumo energético de *data centers*. EAVIRA baseia-se em consolidação de MVs e no modelo de compartilhamento de custo proporcional de [Hinz et al. 2016]. Os resultados obtidos foram analisados com base em três métricas: número de operações, taxa de aceitação e consumo de energia; e indicam que o EAVIRA: *(i)* efetua um maior número de operações, o que é natural, afinal admite operações elásticas; *(ii)* é equivalente ao VITreeM em taxa de aceitação, ou seja, não reduz o número de inquilinos; e *(iii)* é equivalente ou superior ao VITreeM na economia de energia, obtendo ganhos de até 50%.

O EAVIRA é agnóstico ao mecanismo de alocação de recursos para hospedar IVs. Assim, um estudo com outras soluções é um tema para trabalhos futuros. A função objetivo de realocação de EAVIRA é guiada pelo consumo de energia. Uma investigação futura compreende abordagens multi-objetivos, acrescentando outras perspectivas.

Agradecimentos
Os experimentos foram conduzidos na plataforma Grid'5000.

Referências

Akoush, S., Sohan, R., Rice, A., Moore, A. W., and Hopper, A. (2010). Predicting the Performance of Virtual Machine Migration. In *MASCOTS, 2010 IEEE Int. Symp. on*, pages 37–46.

Al-Fares, M., Loukissas, A., and Vahdat, A. (2008). A Scalable, Commodity Data Center Network Architecture. *SIGCOMM Comput. Commun. Rev.*, 38(4):63–74.

Assuncao, M. D., Lefevre, L., and Rossigneux, F. (2016). On the impact of advance reservations for energy-aware provisioning of bare-metal cloud resources. In *Int. Cont. on Network and Service Management*.

Buyya, R., Beloglazov, A., and Abawajy, J. H. (2010). Energy-efficient management of data center resources for cloud computing: A vision, architectural elements, and open challenges. *CoRR*, abs/1006.0308.

Chowdhury, N., Rahman, M. R., and Boutaba, R. (2009). Virtual network embedding with coordinated node and link mapping. In *INFOCOM 2009, IEEE*, pages 783–791. IEEE.

Cisco (2007). Cisco Data Center Infrastructure 2.5 Design Guide. Technical report, Cisco Systems, Inc.

Clark, C., Fraser, K., Hand, S., Hansen, J. G., Jul, E., Limpach, C., Pratt, I., and Warfield, A. (2005). Live migration of virtual machines. In *Proc. of the Symp. on Networked Systems Design & Implementation*.

Darolt, D., Souza, F., and Koslovski, G. (2016). Explorando a elasticidade de nuvens iaas para reconfigurar dinamicamente aplicações n-camadas. *Revista Brasileira de Computação Aplicada*, 8(2):2–15.

Duan, H., Chen, C., Min, G., and Wu, Y. (2016). Energy-aware scheduling of virtual machines in heterogeneous cloud computing systems. *Future Generation Computer Systems*.

Fischer, A., Botero, J. F., Beck, M. T., de Meer, H., and Hesselbach, X. (2013). Virtual network embedding: A survey. *IEEE Communications Surveys Tutorials*, 15(4):1888–1906.

Galante, G. and de Bona, L. (2012). A survey on cloud computing elasticity. In *Utility and Cloud Computing (UCC), 2012 IEEE Fifth International Conference on*, pages 263–270.

Gong, L., Wen, Y., Zhu, Z., and Lee, T. (2014). Toward profit-seeking virtual network embedding algorithm via global resource capacity. In *INFOCOM, 2014 Proceedings IEEE*, pages 1–9. IEEE.

Hinz, M., Miers, C. C., Pillon, M. A., and Koslovski, G. P. (2016). Um modelo de custo para nuvens iaas baseado no consumo de energia de máquinas virtuais. In *Simpósio Brasileiro de Sistemas de Informação*.

Mell, P. and Grance, T. (2011). The NIST definition of cloud computing. NIST SP 800-145, National Institute of Standards and Technology.

Oliveira, R. and Koslovski, G. (2015). VITreeM - um algoritmo baseado em árvores para alocação de infraestruturas virtuais. In *XXXIII Simpósio Brasileiro de Redes de Computadores e Sistemas Distribuídos*.

Oliveira, R. and Koslovski, G. (2016). A tree-based algorithm for virtual infrastructure allocation with joint virtual machine and network requirements. *International Journal of Network Management*. nem.1958.

Pfitscher, R., Pillon, M., and Obelheiro, R. (2013). Diagnóstico do Provisionamento de Recursos para Máquinas Virtuais em Nuvens IaaS. In *Simp. Brasileiro de Redes de Computadores e Sistemas Distribuídos*.

Righi, R. (2013). Elasticidade em cloud computing: conceito, estado da arte e novos desafios. *Revista Brasileira de Computação Aplicada*, 5(2):2–17.

Righi, R., Rodrigues, V., da Costa, A., Galante, G., Bona, L., and Ferreto, T. (2015). Autoelastic: Automatic resource elasticity for high performance applications in the cloud. *Cloud Computing, IEEE*, PP(99):1–1.

Ruck, D. B., Oliveira, R., and Koslovski, G. P. (2014). Comparação de algoritmos para alocação de Infraestruturas Virtuais. *Revista Brasileira de Computação Aplicada*, 6(2):98–112.

Sauvé, J., Marques, F., Moura, A., Sampaio, M., Jornada, J., and Radziuk, E. (2005). Sla design from a business perspective. In *Int. Workshop on Distributed Systems: Operations and Management*.

Sharma, U., Shenoy, P., Sahu, S., and Shaikh, A. (2011). A cost-aware elasticity provisioning system for the cloud. In *31st Int. Conf. on Distributed Computing Systems*, pages 559–570.

Simões, R. and Kamienski, C. (2014). Gerenciamento de elasticidade em nuvens privadas e híbridas. In *Anais do XII Workshop de Computação em Clouds e Aplicações, WCGA*, pages 67–78.

Strunk, A. (2012). Costs of virtual machine live migration: A survey. In *Services (SERVICES), 2012 IEEE Eighth World Congress on*, pages 323–329.

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.

Zhu, Y. and Ammar, M. H. (2006). Algorithms for assigning substrate network resources to virtual network components. In *INFOCOM*, pages 1–12.

Análise Integrada de Desempenho e Consumo de Energia em Sistemas de Armazenamento de Dados Distribuídos

Juccelino Barros[1], **Gustavo Callou**[1], **Glauco Gonçalves** [1]

[1]Departamento de Estatística e Informática
Universidade Federal Rural de Pernambuco (UFRPE) – Recife – PE – Brasil

{juccelino.abarros,gustavo.callou,glauco.goncalves}@ufrpe.br

Abstract. *Performing large amount of data storage is a challenge, since the demand to manage the massive volume of data is growing. However, traditional data storage systems, such as relational databases, were not designed to meet this demand. Therefore, new data storage systems have been emerging, for instance, non-relational databases NoSQL. The main factor for choosing a data storage system is the performance in typical operations. However, energy consumption also represents a major factor for both environment and cost. This paper performs an integrated analysis of performance and energy consumption of a distributed data storage system, represented by Cassandra, in the data insertion process. A case study was conducted taking into account measurement techniques and the YCSB benchmark to generate load in several scenarios with different numbers of nodes in the Cassandra cluster. The achieved results were significant, observing in several situations that the improvement of the computational resources lead, as a matter of priority, the increase of the energy consumption, without having a representative increase in the performance.*

Resumo. *Realizar o armazenamento de grande quantidade de dados é um desafio, visto que a demanda para gerenciar o volume massivo de dados vem crescendo. Contudo, os sistemas tradicionais de armazenamento de dados, como os bancos de dados relacionais, não foram projetados para atender à essa demanda. Consequentemente surgiram novos sistemas de armazenamento de dados, entre eles os bancos de dados não relacionais NoSQL. O principal fator para a escolha de um sistema de armazenamento de dados é o desempenho em operações típicas. No entanto, o consumo de energia também é considerado um fator importante principalmente para o meio ambiente e para os investimentos. Este artigo realiza uma análise integrada de desempenho e consumo de energia, através da medição, de um sistema de armazenamento de dados distribuídos, representado pelo Cassandra no processo de inserção de dados. Um estudo de caso foi realizado fazendo uso de técnicas de medição e do benchmark YCSB para gerar carga em diversos cenários com quantidade diferente de nós no cluster do Cassandra. Os resultados obtidos foram significativos, observando-se em diversas situações que o aumento dos recursos computacionais acarretam, prioritariamente, o aumento do consumo de energia, sem ter um incremento representativo no desempenho.*

1. Introdução

A demanda para gerenciar grandes quantidades de dados em *datacenters* vem crescendo ao longo dos anos. Além disso, os tradicionais sistemas de armazenamento de dados,

como os bancos de dados relacionais, não foram projetados para atender esta demanda [Lóscio et al. 2011] e, consequentemente, novos sistemas estão surgindo.

Paralelo ao aumento desta demanda está o rápido crescimento no consumo de energia nesses ambientes [Koomey 2011][Venkatraman 2012]. Segundo [NRDC 2015], o consumo de energia dos *datacenters* nos Estados Unidos foi de 91 bilhões *KiloWatts* (KW) em 2013, custando USD 9 bilhões. Estima-se que este consumo atingirá 140 bilhões de KW em 2020, custando em torno de USD 13,7 bilhões. Alguns dos principais motivos para este crescimento foram o aumento do número de *datacenters*, a demanda de dados e o barateamento do custo de aquisição de novos equipamentos, como servidores e demais componentes.

Entre os novos sistemas de armazenamento de dados estão os bancos de dados não relacionais, conhecidos como NoSQL (*Not Only SQL*), os quais são facilmente escaláveis entre várias máquinas e que têm o foco no desempenho em operações típicas como inserção e consulta. Os bancos de dados NoSQL apresentam características diferentes dos bancos de dados relacionais, pois focam na disponibilidade dos dados ao invés da consistência e não apresentam os padrões rígidos encontrados no modelo relacional [McCreary and Kelly 2014]. Por serem facilmente escaláveis, bancos de dados NoSQL permitem maior tolerância a falhas. O principal fator para a escolha de um sistema de armazenamento de dados é o desempenho em operações típicas. Este fator é bastante investigado na academia [Cooper et al. 2010][Li and Manoharan 2013][Abubakar et al. 2014]. Contudo, o consumo de energia também é considerado um fator importante principalmente para o meio ambiente e para os investimentos. Porém, quais são as melhores formas de utilização desses sistemas levando em consideração o desempenho e o consumo de energia?

Este artigo realiza uma análise integrada de desempenho e consumo de energia, através da medição de um sistema de armazenamento de dados distribuídos, representado pelo Cassandra, no processo de inserção de dados. Para isso, foi utilizada a ferramenta de *benchmark Yahoo! Cloud Serving Benchmark* (YCSB) [Cooper et al. 2010] para gerar carga em diversos cenários variando a quantidade de operações de inserção e a quantidade de nós no *cluster* do Cassandra. Sendo assim, o objetivo desse trabalho é fornecer insumos para ajudar os projetistas de ambientes de armazenamento de grande quantidade de dados a escolher uma configuração mais apropriada do seu ambiente de armazenamento, levando em consideração esses dois requisitos conflitantes, energia e desempenho.

O artigo está dividido conforme descrito a seguir. A Seção 2 apresenta as características de um sistema de armazenamento de dados distribuídos, abordando alguns aspectos do Cassandra e uma descrição da ferramenta geradora de carga YCSB. A Seção 3 mostra os trabalhos relacionados. Em seguida, a Seção 4 apresenta a metodologia utilizada nessa pesquisa. A Seção 5 apresenta um estudo de caso que realiza uma análise de desempenho e consumo de energia nos cenários propostos. Por último, a conclusão do artigo e futuros direcionamentos desse trabalho são apresentados.

2. Fundamentação Teórica e Ferramentas

Esta seção apresenta as principais características de um sistema de armazenamento de dados distribuídos, abordando alguns aspectos dos bancos de dados não relacionais e do Cassandra. Também é oferecida uma descrição do *Yahoo! Cloud Serving Benchmark*.

2.1. Características dos Sistemas de Armazenamento de Dados Distribuídos

Com o surgimento de novos meios de acesso à Internet, o número de dados trafegados está em constante crescimento. Com o objetivo de atender a grande demanda de dados, os sistemas de armazenamento estão cada vez mais robustos. Estes sistemas de armazenamento apresentam um conjunto de características que atendem a grande demanda de dados. Entre as principais características estão: escalabilidade, elasticidade e alta disponibilidade.

A escalabilidade pode ser dividida em 2 formas: vertical e horizontal. A vertical consiste em aumentar o número de recursos computacionais (memória, processador e espaço em disco) no servidor com o sistema de armazenamento. Já a horizontal consiste em aumentar o número de servidores, com cópias ou partes do sistema de armazenamento [Lóscio et al. 2011]. A elasticidade compreende em adicionar mais servidores com o sistema de armazenamento de dados em execução e sem afetar os demais componentes negativamente, podendo até distribuir os dados entre as instâncias do sistema de armazenamento seguindo critérios previamente definidos. A alta disponibilidade consiste em fazer com que o sistema de armazenamento de dados seja tolerante a falhas, visto que em um cenário com vários servidores é comum acontecer uma falha. Logo estes sistemas possuem técnicas e estratégias para prover alta disponibilidade mesmo quando os servidores apresentam problemas [De Diana and Gerosa 2010].

Um dos grandes desafios é aplicar essas 3 características nos sistemas de armazenamento tradicionais [Cooper et al. 2010], como os bancos de dados relacionais, pois eles não foram projetados com tais finalidades. Consequentemente, novos bancos de dados foram surgindo com essas características. Porém, para atender as 3 principais características, as consultas complexas e transações sofisticadas foram sacrificadas em alguns desses novos sistemas de armazenamento de dados. Entre os sistemas de armazenamento de dados distribuídos que têm essas características estão os bancos de dados NoSQL, que são pertencentes a um novo paradigma chamado de Não Relacional.

2.2. Banco de Dados Não Relacional - NoSQL

Os bancos de dados NoSQL (*Not Only SQL*) não utilizam a linguagem SQL para realizar suas transações. Eles apresentam um conjunto de conceitos que permitem o processamento de dados de forma rápida e eficiente com o foco em desempenho [McCreary and Kelly 2014]. Representam uma alternativa para modelar dados sem se preocupar com os padrões rígidos propostos pelo modelo relacional. O NoSQL tem uma estrutura distribuída e tolerante a falhas que se baseia na redundância de dados em vários servidores.

Existem vários modelos diferentes desses bancos de dados. Os principais modelos são chave-valor, orientado a coluna, orientado a documentos e orientado a grafos [Carniel et al. 2012]. Cada modelo possui vantagens e desvantagens. Para manipulação de dados estatísticos, frequentemente escritos mas raramente lidos, pode ser usado um banco de dados do tipo chave e valor ou orientado a documentos. Para uma aplicação com alta disponibilidade, onde a minimização da inatividade é fundamental, é recomendado utilizar um banco de dados orientado a coluna. Para alto desempenho de consultas mais complexas, recomenda-se o modelo orientado a grafos [Lóscio et al. 2011].

Cassandra. É um banco de dados orientado a coluna, criado para armazenar grandes quantidades de dados espalhados por vários servidores e, mesmo assim, oferece alta viabilidade de acesso a dados consistentes [Lakshman and Malik 2010]. Este banco de dados foi baseado na arquitetura do *Dynamo* da Amazon[1] e o no modelo de dados do *Bigtable* da Google[2]. Possui uma linguagem própria chamada CQL (*Cassandra Query Language*), muito semelhante ao SQL.

Uma das principais características do Cassandra é a sua escalabilidade horizontal, onde vários servidores possuem a mesma instância do banco de dados. A forma de distribuição é do tipo anel, não existe a abordagem do tipo "*master-slave*" como também não existe a distribuição de tarefas exclusivas para um nó específico, ou seja, todos os nós armazenam dados e podem executar as instruções requisitadas pela aplicação cliente. A grande vantagem de utilizar esta arquitetura é que o Cassandra não apresenta um ponto comum de falha, caracterizando a alta disponibilidade dos dados. Outra vantagem desta forma de distribuição é a possibilidade de adicionar mais nós no *cluster* em tempo de execução, caracterizando a elasticidade.

O nó responsável pela distribuição das requisições é chamado de *Coordinator Node*. Normalmente, o nó que apresenta este papel é o que a aplicação cliente se conecta. É importante lembrar que qualquer nó pode ser um *Coordinator Node*. As requisições da aplicação cliente também poderão ser executadas no próprio *Coordinator node*, visto que este nó também armazena dados.

A escrita no Cassandra passa por algumas etapas. Quando uma escrita ocorre, o Cassandra armazena os dados na memória RAM, em uma estrutura chamada *memtable*, enquanto a instrução de escrita é gravada em outra estrutura no disco chamada *commitlog*. O *commitlog* é importante para que, em caso de falha no *hardware*, não percam os registros que serão inseridos. Quando a *memtable* fica cheia acontece o processo de *flush*, em que os dados são descarregados para outra estrutura em disco chamada de *SSTable*. Após o *flush*, o *commitlog* é apagado. Para aumentar a tolerância a falhas, é necessário diminuir a memória da *memtable*, pois assim os dados são escritos mais rapidamente na *SSTable*, mas isto diminui o desempenho da consulta. A Figura 1 ilustra a estrutura da escrita e leitura do banco.

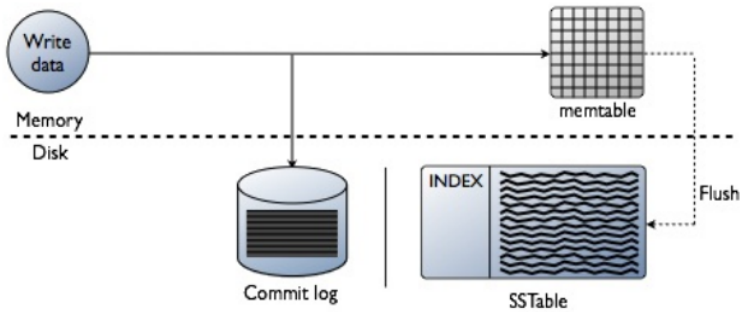


Figura 1. Estrutura da escrita do Cassandra [Enterprise 2017].

[1]http://aws.amazon.com/pt/documentation/dynamodb/ (Acessado em 27/04/2017)
[2]https://cloud.google.com/bigtable/ (Acessado em 27/04/2017)

2.3. Yahoo! Cloud Serving Benchmark (YCSB)

A ferramenta YCSB foi desenvolvida pela *Yahoo!* em conjunto com vários membros dos mais variados sistemas de armazenamento de dados (entre eles o Cassandra). Os objetivos dessa ferramenta são: criar uma ferramenta padrão para auxiliar a escolha do sistema de armazenamento de dados, facilitar a comparação de desempenho provendo as principais métricas e automatizar o ambiente de medição. É uma ferramenta de código aberto[3], desenvolvida em Java e com suporte para vários bancos de dados. Contudo, a principal característica do YCSB é a sua extensibilidade. Por meio desta ferramenta é possível criar cargas de trabalho através de arquivos de configuração e experimentar diferentes bancos de dados escrevendo novas classes aproveitando os métodos e as interfaces já desenvolvidos [Cooper et al. 2010].

O YCSB possui duas fases de execução. A primeira fase é denominada de *load*, onde os dados são inseridos. Nesta fase é definida a quantidade de dados inseridos no banco de dados. A outra fase é a *transaction*, que é executada através do parâmetro *run*, onde os dados são lidos e alterados. Normalmente, nesta fase são realizadas consultas e alterações numa quantidade menor que a inserida na fase anterior. Em ambas as fases é possível determinar a quantidade de usuários simultâneos, a carga de trabalho, o banco de dados e também definir uma vazão fixa. Após a execução de cada fase, o YCSB computa diversas métricas, entre elas: o tempo de execução, a vazão média, a latência das requisições e a quantidade de operações bem sucedidas e que falharam.

3. Trabalhos Relacionados

A avaliação de desempenho de bancos de dados é uma área de pesquisa ativa e que tem se intensificado recentemente devido ao aparecimento de modelos de bancos de dados alternativos ao relacional. Contudo, poucos trabalhos têm focado em analisar o consumo de energia desses sistemas.

Em [Abubakar et al. 2014] foi feita uma análise de desempenho em operações de inserção, leitura e atualização comparando 4 sistemas de armazenamento em um ambiente com recursos limitados, utilizando a ferramenta de *benchmark* YCSB. Em [Li and Manoharan 2013] também foi realizada uma análise comparativa de desempenho de sistemas de armazenamento de dados , com o foco em comparar os paradigmas relacional e não relacional. No total foram 6 bancos de dados não relacionais de diferentes modelos e arquiteturas e um banco de dados relacional. Para realizar os experimentos foi desenvolvido um *framework* específico para aquele ambiente. Embora tenham uma boa cobertura com diferentes bancos de dados, estes trabalhos não avaliam cenários distribuídos, que são essenciais para um melhor desempenho dos bancos analisados.

Os autores em [Cooper et al. 2010] propõem a ferramenta de *benchmark Yahoo! Cloud Serving Benchmark* (YCSB), descrita na seção 2.3. Uma análise de desempenho foi realizada em 4 sistemas de armazenamento de dados que estavam na forma distribuída em 6 servidores. Em [Maciel et al. 2014] foi realizada uma análise de desempenho e escalabilidade no *Sheepdog*, que é um sistema de armazenamento de dados distribuídos que fornece alta disponibilidade utilizando máquinas comuns. Embora tenham feito análises com bancos distribuídos, os trabalhos limitaram-se à avaliação de desempenho e não avaliaram aspectos de consumo de energia.

[3]https://github.com/brianfrankcooper/YCSB/wiki (Acessado em 27/04/2017)

Com foco apenas no consumo de energia, [Li et al. 2014] propõem uma nova estratégia de redução do consumo de energia através da diminuição da *Waiting Energy Consumption* (WEC). Este tipo de energia é desperdiçada quando alguns nós do *cluster* estão à "espera" de alguma atividade, devido a isto, não podem ser desligados. Outros trabalhos realizaram uma análise tanto de desempenho quanto do consumo de energia. Em [Gomes et al. 2016] foi feita uma análise comparativa de desempenho e consumo de energia de 3 sistemas de armazenamento de dados, porém desconsiderando cenários distribuídos. Em [Niemann 2015] e [Niemann 2016] foram propostos modelos computacionais utilizando *Queued Petri Nets* (QPN) para modelar o desempenho e o consumo de energia de um sistema de armazenamento de dados distribuídos, representado pelo Cassandra. Esses trabalhos abordaram cenários onde o *cluster* do Cassandra já estava povoado e eram realizadas operações de leitura e escrita sobre os dados já existentes. A Tabela 1 apresenta uma visão geral de cada trabalho relacionado.

Tabela 1. Visão geral dos trabalhos relacionados.

Trabalho	Desempenho	Energia	Dados Distribuídos
[Cooper et al. 2010]	Sim	Não	Sim
[Li and Manoharan 2013]	Sim	Não	Não
[Li et al. 2014]	Não	Sim	Sim
[Abubakar et al. 2014]	Sim	Não	Não
[Maciel et al. 2014]	Sim	Não	Sim
[Niemann 2015]	Sim	Sim	Sim
[Gomes et al. 2016]	Sim	Sim	Não
[Niemann 2016]	Sim	Sim	Sim

Diferente dos trabalhos anteriores, esta pesquisa realiza uma análise integrada de desempenho de consumo de energia de sistemas de armazenamento de dados distribuídos em vários servidores durante a inserção de pequenas e grandes quantidades de dados. O sistema de armazenamento escolhido foi o Cassandra. Esta escolha foi feita pelo fato do Cassandra estar entre os 10 bancos de dados mais utilizado, conforme o *DBEngines*[4] e por ele ter sido projetado para ser distribuído em várias máquinas. Foram abordados diversos cenários variando o número de nós no *cluster* do Cassandra como também a quantidade de operações de inserção.

4. Metodologia

Para realizar a análise de desempenho e consumo de energia de sistemas de armazenamento de dados distribuídos no processo de inserção de dados, são analisados dois fatores: a quantidade de nós no *cluster* e a quantidade de operações de inserção. Os fatores são variados em níveis. A combinação dos níveis dos fatores indicam a quantidade de cenários que devem ser analisados.

A primeira etapa é a montagem de um ambiente isolado, sem interferências de fatores externos. Em seguida, realiza-se os experimentos conforme os fatores e os níveis escolhidos. Nesta etapa, é recomendável utilizar ferramentas de *benchmarks*, que executam tarefas bem definidas no sistema e fornecem várias métricas. A ferramenta de

[4]http://db-engines.com/en/ranking (Acessado em 27/04/2017)

benchmark YCSB é uma das mais utilizadas para analisar o comportamento de sistemas de armazenamento de dados, fornecendo as principais métricas. Também é responsável por gerar carga de trabalho. Já para obter métricas de consumo de energia, pode-se utilizar dispositivos de medição, entre eles, o *Wattsup Meter*[5]. Este dispositivo registra métricas de consumo de energia, inserindo os valores em arquivos textos. Esses valores são capturados em intervalo de tempo ajustável. Quanto menor o intervalo de tempo, mais preciso são os valores do consumo de energia. O *Wattsup Meter* é conectado entre o estabilizador e a rede elétrica, também servindo como condutor de energia ao sistema. A Figura 2 ilustra a distribuição dos componentes no ambiente de medição.

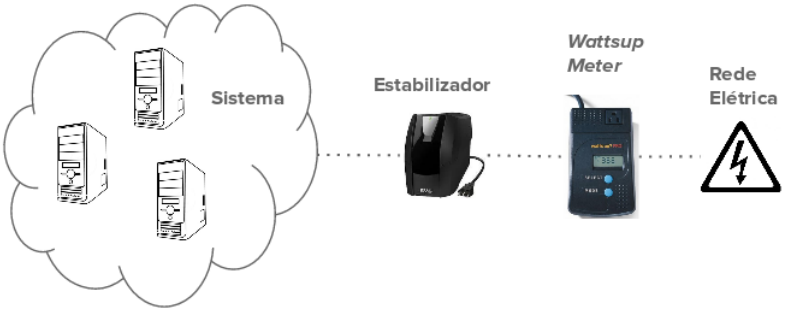


Figura 2. Visão geral dos componentes do ambiente de medição.

4.1. Processo de Obtenção das Amostras

O primeiro passo da metodologia consiste na preparação dos nós do *cluster* do Cassandra, onde todos os *logs*, *commitlogs* e *caches* gerados pelo *cluster* são apagados. Em seguida, realiza-se a remoção dos registros gerados no *Wattsup Meter*, que armazena, sem interrupções, as informações do consumo de energia a cada 1 segundo, por exemplo.

Após os procedimentos de preparo dos componentes, se executa a fase *load* da ferramenta YCSB, ou seja, inicia-se a inserção dos dados no *cluster* do Cassandra. Ao término da inserção de todos os dados, os registros gerados no *Wattsup Meter* são armazenados e para cada amostra é gerada uma tabela em um arquivo texto. A partir dessas informações pode-se computar o consumo de energia da amostra. Em seguida, registra-se as métricas obtidas no YCSB: tempo de execução, vazão e latência média. Este procedimento é repetido ao menos 30 vezes para que se possa ter confiabilidade nos resultados obtidos. Por último, realiza-se a análise de resultado fornecendo suporte para as conclusões dos experimentos. A Figura 3 ilustra a sequência de passos do processo de obtenção das amostras.

[5]https://www.wattsupmeters.com/secure/index.php (Acessado em 27/04/2017)

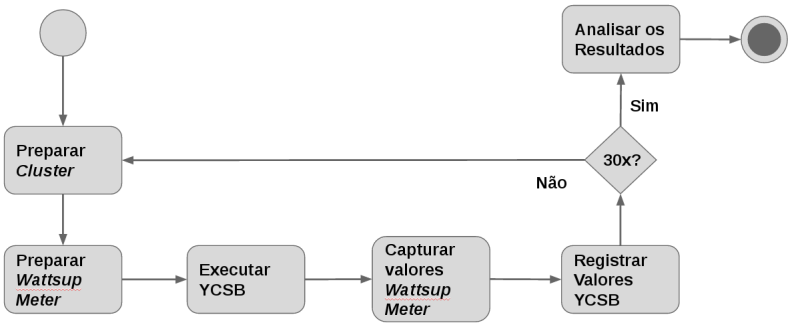


Figura 3. Metodologia utilizada na obtenção das métricas de desempenho e consumo de energia.

5. Estudo de Caso

Esta seção apresenta um conjunto de experimentos realizados com o propósito de analisar o desempenho e o consumo de energia do banco de dados Cassandra no processo de inserção de dados. O fator quantidade de nós no *cluster* tem 4 níveis: 1, 2, 4 e 6 nós. Já o fator quantidade de operações de inserção apresenta 3 níveis: 10.000 (10K), 100.000 (100K) e 1.000.000 (1M) operações. Os registros gerados pelo YCSB eram do tipo texto e inseridos em uma tabela com 10 colunas. Cada registro tinha 1KB de informação. Logo foram inseridos 10MB, 100MB e 1GB de dados para os cenários 10K, 100K e 1M, respectivamente. Foram utilizados 10 usuários simultâneos.

As métricas de desempenho, obtidas pelo YCSB, são: o tempo de execução para inserir o todos os registros (segundos), a vazão do sistema (operação por segundo) e a latência média (milissegundos). O consumo de energia foi obtido ao final do tempo de execução do experimento e a unidade de medida foi *joules*. Para cada combinação dos níveis dos fatores (12 combinações) foram coletadas 33 amostras, as 3 primeiras amostras eram descartadas, pois foi observado através do tempo de execução que elas atingiam valores bem acima das demais, sendo consideradas como período de *warm-up* do sistema de armazenamento. O intervalo de confiança para a média, considerando 95% de confiança, foi calculado e, na maioria dos casos, os intervalos não se sobrepõem, mostrando que os valores das médias são estatisticamente diferentes. Por isso, os intervalos de confiança não serão mostrados nos gráficos, mas os casos em que não foi possível definir a diferença estatística serão pontuados no texto.

5.1. Ambiente

O ambiente montado em laboratório foi composto por quatro computadores, cada um com processador Intel core i5, 8GB de memória RAM, 500GB de espaço em disco e com sistema operacional Ubuntu 14.04 LTS. Todos conectados através de uma rede local *Gigabit Ethernet*. A ferramenta de *benchmark* YCSB foi executada em um computador dedicado. Já os nós do *cluster* do Cassandra eram máquinas virtuais (VM), todas idênticas com 2GB de memória RAM, 30GB de espaço em disco e com sistema operacional Ubuntu 14.04 LTS. Nos três computadores restantes eram hospedados até dois nós (2 VMs) de acordo com o cenário. Logo, para os experimentos com 2, 4 e 6 nós foram utilizados 1, 2 e 3 computadores, respectivamente. Apenas as máquinas físicas necessárias para cada *cluster* eram ligadas, por exemplo, para o *cluster* com 2 nós, apenas 1 máquina física (com as 2 VMs hospedadas) era utilizada, as demais permaneciam desligadas.

Foram utilizados 2 estabilizadores, um alimentava o computador com a aplicação cliente (YCSB) junto com os demais componentes: monitor e *switch*. Já os demais computadores estavam ligados ao segundo estabilizador que, por sua vez, se conectava ao *Wattsup Meter* o qual estava ligado à rede elétrica. Esta abordagem foi feita com o objetivo de isolar todo o sistema de armazenamento para obtenção do consumo de energia. A Figura 4 ilustra uma visão geral dos componentes do ambiente de medição.

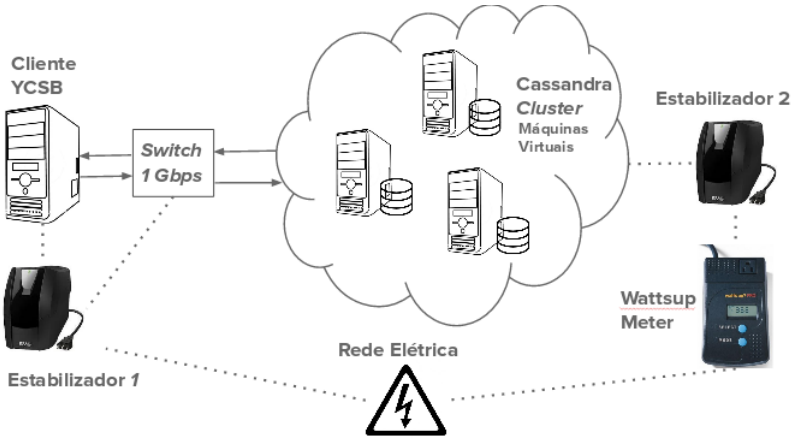


Figura 4. Visão geral do ambiente de experimentação.

5.2. Análise de Desempenho

Foram calculados valores estatísticos como a média, desvio padrão e intervalo de confiança para cada cenário. Além disso, em algumas situações foi realizado Teste-t emparelhado com o objetivo de verificar se as amostras são estatisticamente equivalentes. Cada métrica será abordada separadamente.

Tempo de Execução. A Figura 5 mostra o tempo de execução de acordo com a quantidade de registros inseridos. Para 10K operações de inserção, é possível observar que o tempo permaneceu constante mesmo com o aumento da quantidade de nós. Já para 100K, é possível observar que o tempo de execução para 1 e 2 nós são estatisticamente iguais. Isto pode ser observado através do teste-t emparelhado cujo resultado é mostrado na Tabela 2. Dessa forma, não podemos rejeitar a hipótese nula, a qual indica a igualdade dos tempos de execução. Contudo, a partir de 4 nós, é possível observar uma queda no tempo de execução. Já para 1M de operações, o tempo de execução para 2 nós é um pouco maior do que para 1 nó. Também foi realizado teste-t (Tabela 2) nesta situação, que indicou que os valores são diferentes, visto que o valor-p é menor que o α. Este comportamento pode ser justificado pelo compartilhamento de recursos entre as VMs, em razão de utilizar um único computador para hospedar os 2 nós e com isso os recursos como a placa de rede e disco são compartilhados entre as duas VMs.

Tabela 2. Teste-T no tempo de execução entre *clusters* com 1 e 2 nós.

Qntd Operações	Valor-p	α	Diferença entre as médias
100K	0,1397	0,05	0,54 seg
1M	6,147E-008	0,05	7,35 segs

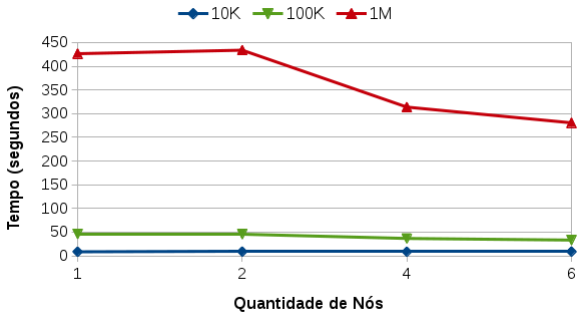


Figura 5. Tempos de execução de acordo com o número de registros inseridos.

Vazão. A Figura 6 mostra o comportamento da vazão em relação à quantidade de operações de inserção no *cluster*. É possível observar que para 10K operações, a vazão apresentou um comportamento uniforme, no entanto para as demais situações foi constatado um crescimento considerável a partir de 4 nós. Em todos os cenários, a vazão permaneceu similar para 1 e 2 nós. Para as inserções de 100K e 1M, houve um aumento significativo, principalmente para 1 milhão de registros, este último apresentou um crescimento de mais de 50% em relação à um nó. Este crescimento pode ser comprovado na Figura 7, que indica a proporção da vazão para 2, 4 e 6 nós em relação a 1 nó. Pode-se concluir que o aumento do *cluster* de 1 para 2 nós, não foi uma estratégia interessante em razão da vazão não apresentar nenhum aumento significativo em nenhuma das situações analisadas. Além de consumir mais recursos computacionais.

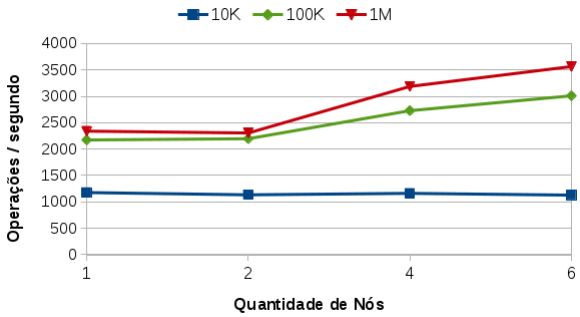


Figura 6. Comportamento da Vazão.

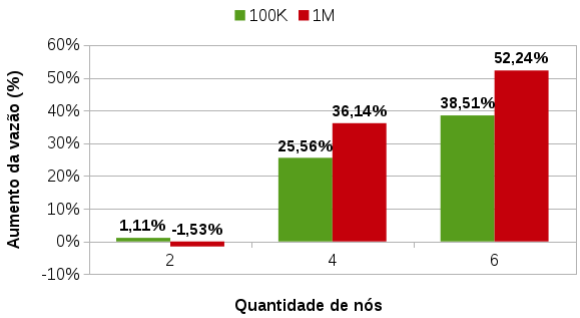


Figura 7. Comparação da vazão em relação a 1 nó para 100K e 1M de operações.

Latência. A latência é o intervalo de tempo entre o envio da requisição para inserir o registro e a chegada da resposta do sistema de armazenamento para a aplicação cliente. A Figura 8 ilustra o comportamento da latência. É possível observar que para 10K operações de inserção, a latência apresentou um comportamento uniforme, onde os tempos tinham diferença média de 2%. No entanto, quando o *cluster* era composto por 2 nós, a latência teve um aumento de 5% em relação a 1 nó. Para 100K e 1M operações de inserção, é constatada uma queda a partir de 4 nós. Esta queda representou uma diminuição de aproximadamente 26,8% quando o *cluster* era composto com 4 nós e 35% para 6 nós, em relação a 1 nó. É possível observar que para as duas maiores quantidades de operações de inserção (100K e 1M), os valores médios quantificados ficaram sobrepostos, indicando que, para os cenários analisados, um *cluster* Cassandra com 4 nós oferece a melhor razão entre latência e uso de recursos.

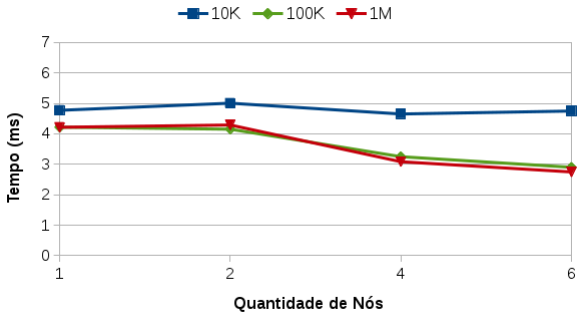
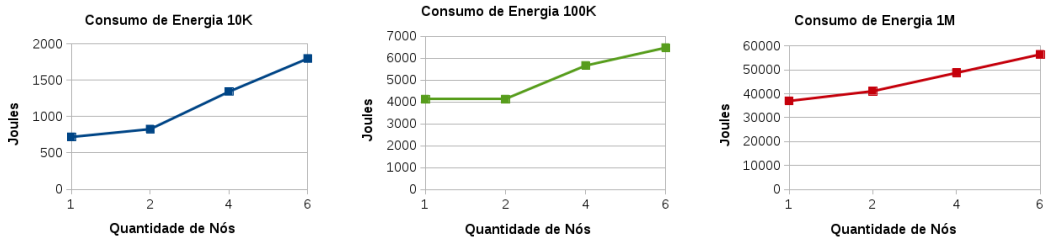


Figura 8. Análise Latência.

5.3. Análise do Consumo de Energia

As Figuras 9(a), 9(b), 9(c) ilustram o consumo de energia de acordo com o aumento do número de nós no *cluster* para, respectivamente, 10K, 100K e 1M operações de inserção. É possível obsevar que para 10K (Figura 9(a)), houve um aumento significativo do consumo de energia para 4 e 6 nós. Para 100K (Figura 9(b)), o consumo de energia para 1 e 2 nós foi similar. Mesmo utilizando 2 nós hospedados em um único computador, o consumo de energia não sofreu alterações. Contudo para 4 e 6 nós houve um aumento significativo, visto que a quantidade de computadores aumentou para 2 e 3 respectivamente.



(a) 10K (b) 100K (c) 1M

Figura 9. Consumo de Energia para 10K (a), 100K (b) e 1M (c)

A Figura 10 mostra o aumento do consumo de energia em relação a 1 nó. É constatado um aumento significativo de 87% e 150% no consumo de energia para realizar 10K

operações de inserção em 4 e 6 nós respectivamente, devido a adição de mais computadores no ambiente (2 computadores para 4 nós e 3 computadores para 6 nós). Já para 100K e 1M, os aumentos foram próximos. Para 4 nós o aumento foi 37% e 32% para 100K e 1M, respectivamente. Enquanto que para 6 nós o aumento foi de 57% e 53% nas mesmas condições.

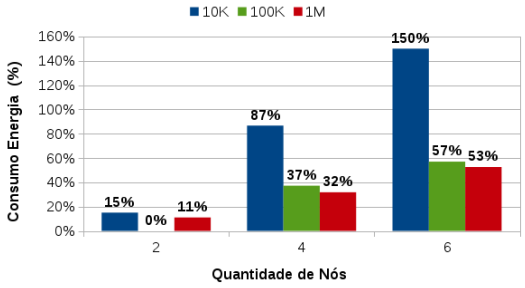


Figura 10. Aumento do consumo de energia em relação a 1 nó.

5.4. Discussão

Analisando de forma conjunta o desempenho e o consumo de energia dos cenários propostos é possível observar que o aumento de 1 para 2 nós no *cluster* não apresentou melhorias significativas para o desempenho. Um dos fatores para este comportamento é o compartilhamento de recursos, como placa de rede e disco, já que as 2 máquinas virtuais estavam hospedadas em um único computador. Observa-se que a introdução de novos nós virtuais sob um mesmo computador físico, pode não apresentar bons resultados. Isso decorre do fato de que os nós virtuais compartilham recursos físicos de entrada e saída que são críticos em sistemas virtualizados. Por outro lado, quando utilizavam-se 2 nós, o aumento do consumo de energia em relação a 1 nó foi de 15% para realizar 10K operações de inserção e 11% para inserir 1M de registros.

Quando o *cluster* era composto por 4 e 6 nós, foi constatada uma melhoria significativa no desempenho em relação a 1 nó, principalmente para realizar inserção dos dois maiores conjuntos de dados, 100K e 1M. Entre as métricas de desempenho, pode-se destacar a vazão, que obteve um aumento de mais de 50% quando inseridos 1M de registros em um *cluster* composto por 6 nós. Entretanto, houve um aumento de 150% no consumo de energia quando inseridos 10K registros. Além disso, ocorreu um aumento, em relação a 1 nó, de 57% e 53% no consumo de energia para inserir, respectivamente, 100K e 1M de registros. A Tabela 3 mostra uma análise na eficiência do desempenho e do consumo de energia para 100K e 1M em um *cluster* composto por 4 e 6 nós em relação a 1 nó. Não foi pontuado 10K operações nas métricas de desempenho, pois não houve alterações significativas, entretanto foi adicionado na métrica de consumo de energia. Os valores negativos indicam uma redução em relação ao valor encontrado em 1 nó.

Baseados nos valores da Tabela 3, pode-se chegar a conclusão da quantidade ideal de nós de acordo com o número de operações de inserção realizadas. O número ideal de nós para inserir 10K registros no *cluster* é 1, pois o desempenho não apresentou grandes melhorias para *clusters* maiores e o consumo de energia nesta situação foi o menor entre os cenários avaliados. Já para 100K e 1M, o número de nós ideal foi 4, pois foi notado um aumento considerável no desempenho enquanto que o consumo de energia atingiu valores

Tabela 3. Análise das métricas para 4 e 6 nós em relação a 1 nó.

Métricas	Qtd Operações	4 nós	6 nós
Tempo de Execução	100K	-20,42%	-27,88%
	1M	-26,42%	-34,20%
Vazão	100K	25,56%	38,51%
	1M	36,14%	52,24%
Latência	100K	-22,95%	-31,24%
	1M	-26,80%	-34,81%
Consumo de Energia	10K	86,67%	150,00%
	100K	37,21%	56,98%
	1M	31,77%	52,56%

abaixo do *cluster* com 6 nós. O *cluster* com 6 nós obteve o melhor desempenho, contudo, houve um aumento significativo do consumo de energia, em todos os casos o aumento foi superior a 50%. Então, se o objetivo é apenas melhorar o desempenho o *cluster* ideal é com 6 nós. No entanto, para avaliação integrada de consumo de energia e desempenho, o melhor seria com 4 nós devido ao aumento registrado com 6 nós em relação a 4 nós no consumo de energia.

6. Conclusão

Devido ao grande aumento de dados gerados, a necessidade por um sistema de armazenamento de grande desempenho e que impacta pouco o meio ambiente está aumentando. Neste trabalho foi realizada uma análise integrada de desempenho e consumo de energia. Como resultado, mostrou-se que para algumas situações não é necessário aumentar os recursos computacionais, pois o desempenho não apresenta melhorias significativas enquanto que o consumo de energia do sistema aumenta consideravelmente.

Como trabalhos futuros, pretende-se desenvolver modelos formais de desempenho e consumo de energia desses ambientes. Com esses modelos validados através dos resultados das medições, é possível obter métricas para cenários não analisados em laboratório, economizando tempo e otimizando o planejamento de recursos.

Agradecimentos

Os autores gostariam de agradecer à FACEPE e ao CNPq pelo suporte financeiro a esta pesquisa.

Referências

Abubakar, Y., Adeyi, T. S., and Auta, I. G. (2014). Performance evaluation of NoSQL systems using YCSB in a resource austere environment. In *Performance Evaluation*, volume 7.

Carniel, A. C., de Aguiar Sá, A., Brisighello, V. H. P., Xavier, M., Ribeiro, R. B., Ciferri, R. R., and de Aguiar Ciferri, C. D. (2012). Query processing over data warehouse using relational databases and nosql. In *Informatica (CLEI), 2012 XXXVIII Conferencia Latinoamericana En*, pages 1–9.

Cooper, B. F., Silberstein, A., Tam, E., Ramakrishnan, R., and Sears, R. (2010). Benchmarking cloud serving systems with YCSB. In *Proceedings of the 1st ACM symposium on Cloud computing*, pages 143–154. ACM.

De Diana, M. and Gerosa, M. A. (2010). Nosql na web 2.0: Um estudo comparativo de bancos não-relacionais para armazenamento de dados na web 2.0. In *IX Workshop de Teses e Dissertações em Banco de Dados*, volume 9.

Enterprise, D. (2017). A brief introduction to apache cassandra. Disponível em https://academy.datastax.com/resources/brief-introduction-apache-cassandra (Acessado em 27/04/2017).

Gomes, C., Tavares, E., and Junior, M. N. d. O. (2016). Energy consumption evaluation of NoSQL DBMSs.

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*, 9.

Lakshman, A. and Malik, P. (2010). Cassandra: a decentralized structured storage system. *ACM SIGOPS Operating Systems Review*, 44(2):35–40.

Li, T., Yu, G., Liu, X., and Song, J. (2014). Analyzing the waiting energy consumption of NoSQL databases. pages 277–282. IEEE.

Li, Y. and Manoharan, S. (2013). A performance comparison of SQL and NoSQL databases. In *Communications, Computers and Signal Processing (PACRIM), 2013 IEEE Pacific Rim Conference on*, pages 15–19. IEEE.

Lóscio, B. F., OLIVEIRA, H. R. d., and PONTES, J. C. d. S. (2011). NoSQL no desenvolvimento de aplicações web colaborativas. In *VIII SIMPÓSIO BRASILEIRO DE SISTEMAS COLABORATIVOS, Paraty, RJ: SBC*.

Maciel, P., Matos, R., Callou, G., Silva, B., Barreto, D., Araujo, J., Araujo, J., Alves, V., and Worth, S. (2014). Performance evaluation of sheepdog distributed storage system. In *Systems, Man and Cybernetics (SMC), 2014 IEEE International Conference on*, pages 3370–3375. IEEE.

McCreary, D. and Kelly, A. (2014). Making sense of NoSQL. *Shelter Island: Manning*, pages 19–20.

Niemann, R. (2015). Evaluating the performance and energy consumption of distributed data management systems. In *Global Software Engineering Workshops (ICGSEW), 2015 IEEE 10th International Conference on*, pages 27–34. IEEE.

Niemann, R. (2016). Towards the prediction of the performance and energy efficiency of distributed data management systems. In *Companion Publication for ACM/SPEC on International Conference on Performance Engineering*, pages 23–28. ACM.

NRDC (2015). America's data centers consuming and wasting growing amounts of energy. Disponível em https://www.nrdc.org/resources/americas-data-centers-consuming-and-wasting-growing-amounts-energy (Acessado em 27/04/2017).

Venkatraman, A. (2012). Global census shows datacentre power demand grew 63% in 2012. Disponível em http://www.computerweekly.com/news/2240164589/Datacentre-power-demand-grew-63-in-2012-Global-datacentre-census (Acessado em 27/04/2017).

An Architecture for Fog Computing Emulation

Antonio A. T. R. Coutinho[1], **Fabíola Greve**[2], **Cássio Prazeres**[2]

[1]Departamento de Tecnologia – Universidade Estadual de Feira de Santana (UEFS)
Feira de Santana, Bahia 44036–900

[2]Departamento de Ciência da Computação – Universidade Federal da Bahia
Salvador, Bahia 40170–110

augusto@ecomp.uefs.br,{fabiola,cprazeres}@ufba.br

Abstract. *Fog computing is an emergent paradigm that integrates cloud computing services into the network on a widely distributed level. It extends cloud capabilities near data sources to overcome limitations of cloud-based IoT platforms such as mobility, location awareness and low latency. In spite of its increasing importance, there exist no readily available fog computing platforms which can help researchers to design and test real world fog applications. To this purpose, simulators and testbeds are adapted to enable the investigation of fog solutions. This paper presents a framework architecture to create fog virtualized environments. Its design meets the requirements of low cost, flexible setup and compatibility with real world technologies. Unlike current approaches, the proposed architecture allows for the testing of fog components with third-party systems through standard interfaces. A scheme to observe the behavior of the environment by monitoring network flow is provided. In addition, future developments and research directions are discussed.*

1. Introduction

A cloud-based model has become the default approach for Internet of things (IoT). Due to the limitations of cloud centralized architecture [Aazam et al. 2014], current IoT proposals are fomenting a major paradigm shift towards a decentralized model. Similar solutions in [Satyanarayanan et al. 2009], [Dinh et al. 2013], [Hu et al. 2015], [Bonomi et al. 2012] are bringing cloud computing capabilities near data sources to overcome limitations such as mobility, location awareness and low latency. We can summarize the initiatives through the following convergent points [Coutinho et al. 2016]: use of small clouds with virtualization support, distribution of computational resources on large scale, provision of IoT infrastructure as a service, and the offer of exclusive services using information that is only present on the local network or in its proximity.

In this scenario, fog computing is an emergent paradigm that extends cloud services to the edge of the network in an integrated way with switches, routers and IoT gateways[Bonomi et al. 2012]. The fog distributed approach reduces the amount of data transferred to the core network by capturing and processing the necessary data locally on each edge device; its low latency support brings real-time analysis to edge-based data by distributing the analytic process across the network[Bonomi et al. 2014].

The integration of the fog computing into the scope of IoT requires optimizing, deploying and spreading the concept of the cloud through a dense and geographically distributed platform. All the benefits of cloud should be preserved with fog,

including containerization, virtualization, orchestration, manageability and efficiency. Very recently, interesting works have appeared [Bonomi et al. 2014], [Yi et al. 2015a], [Dastjerdi et al. 2016], [OpenFog 2017a]; they propose a reference architecture, components, services, design goals and challenges of a fog platform. Nonetheless, as far as we know, there are no readily available fog computing platform that can help researchers to design and verify distributed algorithms on a truly IoT scale. Currently, simulation tools and testbeds are adapted to allow the experimental evaluation of fog solutions in specific scenarios and restricted conditions.

This paper presents a framework architecture to create fog testbeds in virtualized environments. Its design is based on solutions that meet the postulated requirements of low cost, flexible setup and compatibility with real world technologies. The components are based on Containernet [Containernet 2017], which allows Mininet [de Oliveira et al. 2014] to use Docker [Docker 2017] containers as virtual nodes. The main contribution is an architecture that enables the testing of fog components with third-party systems through standard interfaces. In addition, a scheme to observe the behavior of the environment is provided through network flow monitors.

The remainder of the paper is organized as follows. Section 2 introduces fog computing concepts. Section 3 presents related work. Section 4 addresses design requirements and technological choices. Section 5 illustrates the framework architecture. The conclusions and future works are presented in Section 6.

2. Fog Environment

The concept of fog computing is defined in [Bonomi et al. 2012] as a virtualized platform which offers services between end devices and the conventional data centers of cloud computing. This definition implies a series of challenges that make the fog environment a non-trivial extension of the cloud computing paradigm. Their characteristics include edge location, location awareness, low latency, geographical distribution, support for large-scale sensor networks, very large number of nodes, support for mobility, real-time interactions, predominance of wireless access, heterogeneity, interoperability, federation and real-time data analysis.

There is a set of IoT solutions that can be supported by fog computing. The fog concept was introduced in [Bonomi et al. 2012] with applications for connected vehicle networks, smart grids and wireless sensor networks. Later, surveys in [Stojmenovic 2014] [Yi et al. 2015b] [Yi et al. 2015a] discussed scenarios that take advantage of fog computing such as smart health, smart building, smart cities, etc.

The fog computing model involves the running of applications concerning distributed components on the fog nodes between the IoT devices and the cloud data center. Fog instances can offer compute and storage resources to support the extension of the cloud services and real-time analysis to the edge of the network. The fog hierarchy makes the transmitted data actionable, meaningful and valuable by filtering information at different levels of the infrastructure.

Figure 1 presents the distributed physical components of a fog architecture [Yi et al. 2015a]. They are distributed hierarchically in a network infrastructure with at least three layers. At the edge of the network there are different IoT devices. The

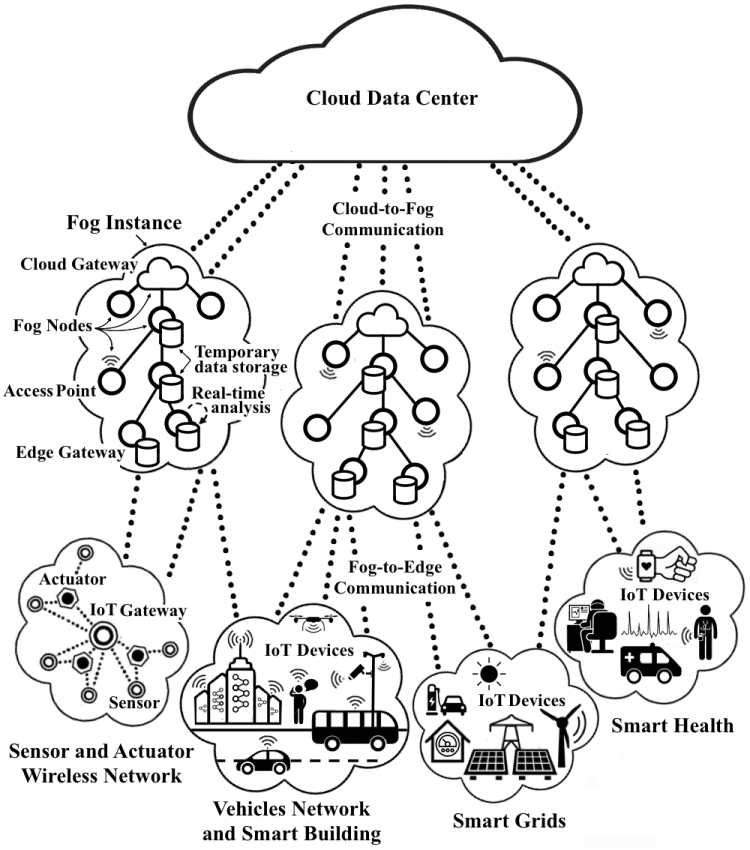


Figure 1. A Fog Architecture.

key components that can provide processing and storage capabilities at the edge of the network are called fog nodes [Tordera et al. 2016]. These fog nodes are involved in the data processing of different IoT applications. For example, sensor and actuator devices installed in parks, buildings and streets can provide valuable streams of data for monitoring, control, analysis and prediction applications. Its location depends on factors such as energy saving or application latency requirements.

Retrieving data from each sensor device is often challenging due to sensor power and communication constraints. An IoT gateway can aggregate and translate data from sensors before sending it onward by supporting different communication protocols. In some applications, embedded smart IoT devices can receive actuation commands or send sensing data without the need for dedicated gateways. Also, they are capable of acting as edge gateways to other IoT devices.

The fog nodes must favor the access of IoT devices to the network, fog and cloud resources. Due to the heterogeneity of fog environments, the fog nodes may present different capabilities, interfaces and hardware configurations. Depending on the application, they can support advanced tasks including temporary storage, preprocessing, data caching, geo-localization, service discovery, data analysis, load balancing, resource management and security. In addition, fog instances can provide access to virtualized resources through local fog nodes. They are also responsible for periodically filtering and sending information to the cloud. In the fog hierarchical organization, cloud gateways can act as proxy servers for larger cloud providers.

The services that support virtually unlimited resources for IoT applications are executed in the cloud. IoT applications can use both cloud and fog infrastructure to provide intelligent and cutting edge solutions to end users. The management software that globally coordinates the fog infrastructure are distributed in a number of services working together. The main purpose of these services is to provide acceptable levels of latency while optimizing the performance of fog applications. This can be achieved by efficient solutions that distribute task execution demand across optimal, well located and available fog nodes, using conventional and non-conventional algorithms.

3. Related Work

Fog computing has been steadily growing since the concept was adopted in 2012 [Bonomi et al. 2012]. As the pioneer in the fog field, Cisco solutions finds itself in a better position compared to other commercial fog products [Dastjerdi et al. 2016]. The Cisco IOx [Cisco IOx 2017] is a network operational system included in routers, switches and compute cards that makes processing and storage available to applications hosted in virtual machines (VMs). Distributed devices with IOx support in a network can function as the compute environment for fog applications. The IOx also supports open source environments based on Linux instances running in a hypervisor or virtual machine manager (VMM). The Cisco IOx Sandbox enables development and testing of fog computing applications using a desktop simulation approach. Together with IOx Fog Director [Fog Director 2017] it is possible to manage large-scale production deployments of IOx-enabled fog applications. In [DevNet 2017] a library with simulation tools, sample codes and templates to help deploy IoT applications for Cisco IOx platform is available. Examples of deployed IOx fog applications include basic services and protocols for IoT support written in Python, Java, Javascript and C.

Existing work has focused on the designing and implementing of an adequate fog computing platform, which can serve as a model development environment for prototyping. Therefore, a standard fog architecture is necessary to allow integration between different solutions. In November 2015, the creation of the OpenFog consortium [OpenFog 2017a] united companies such as Cisco, Microsoft, Dell, Intel, ARM and universities such as Princeton in order to develop a standard fog architecture. The first version of OpenFog reference architecture [OpenFog 2017b] was released in February 2017.

In the absence of ready platforms, a viable alternative is to adapt IoT-specific testbeds to support experimental evaluation of fog solutions. The largest IoT testbed environments, such as [Adjih et al. 2015] and [Sanchez et al. 2014], are equipped with thousands of IoT devices geographically distributed. The authorized users can run applications over a large number of available resources to evaluate low level protocols, analysis techniques and services on a very large scale. However, since they are not intended for fog applications, a major effort is required to configure and deploy experiments.

Although desirable, in many cases the use of a real world testbed facility is expensive, time-consuming and does not provide a repeatable and controllable environment. Also, it has a selective access based on user research impact or political importance. Therefore, low cost-effective alternatives should be considered before a costly experiment so as to eliminate ineffective algorithms, policies and strategies. In [Yi et al. 2015a] a prototyping platform, consisting of two nodes connected to the Ama-

zon EC2 service was presented. The fog capabilities in each node were implemented with OpenStack[Openstack 2017]. A face recognition application was used to evaluate latency and bandwidth performance metrics in a scheme of VM offloading.

The adaptation of existing open source frameworks or simulators is currently a common approach in the fog computing area. Depending on research aims, the setup of testbeds to run specific applications can occur in two ways: adaptation of IoT simulators for fog experiments, or extension of cloud simulators for fog and IoT support.

NS-3 [NS-3 2017] and OMNET++[Varga and Hornig 2008] are examples of open source environments that allow for the extension of their functionality in a specific domain. OMNET++ is not a network simulator itself but an extendable library and framework for building network simulators. It provides a component architecture for simulation models such as Internet standard protocol, wireless sensor networks, etc. There are extensions for real-time simulation, network emulation, database integration, system integration, and several other functions that can be used for modeling a fog environment.

A case study on smart traffic management in [Dastjerdi et al. 2016] extended the CloudSim[Calheiros et al. 2011] with fog computing capabilities to improve the performance of the application in terms of response time and bandwidth consumption. This work was the basis for iFogSim [Gupta et al. 2016], a simulator that supports edge and fog resources under different scenarios in a sense-process-actuate model [Gupta et al. 2016]. Also, it allows for the evaluation of resource management policies by focusing on their impact on latency, energy consumption, network congestion and operational costs.

In summary, fog computing applications continue to be developed as proof-of-concept, implemented in limited environments and applied to specific scenarios. This remains a default approach, given the challenges involved in the implementation of fog environments. Therefore, research concerning fog computing platforms may help to integrate cloud and IoT technologies and accelerate the development of fog applications.

4. Design Requirements and Technologies

In the development of a fog computing platform, the fog nodes and their network infrastructure are the main components to be deployed. In this work, the design of the fog architecture is based on solutions that meet the following requirements: 1) low cost deployment; 2) flexible setup; and 3) support to perform real world protocols and services.

Due to their high cost and restricted availability, proprietary solutions and large-scale IoT testbeds were not considered viable and practical alternatives to the current system architecture. Open source simulations can be considered as cost-effective. However, some solutions do not model fog environments and IoT, where services can be deployed both on edge and cloud resources. Moreover, the use of simulators makes it difficult to support real world IoT protocols and services. Extendible solutions such as OMNET++ do not have a specific framework implementation for the fog computing model. Mainstream network simulators such as NS-3 [NS-3 2017] face the same limitations. Therefore, in the remainder of this section, open source solutions that enable the deployment of the main fog components are described. They allow for the testing of real world technologies in a repeatable and controllable environment. An architecture that employs these solutions to create virtualized fog environments is presented in Section 5.

4.1. Mininet Network Emulator

Mininet [de Oliveira et al. 2014] is a network emulator that uses the Linux kernel to create an experimental network with virtual elements such as hosts, switches and links. Unlike the simulators, it allows for the deployment of protocols and applications in the same way as real world testbeds. Mininet also offers native support to OpenFlow [McKeown et al. 2008] for flexible custom routing and software-defined networking (SDN) [de Oliveira et al. 2014].

Mininet allows the creation of virtual hosts that can run standard Linux software by using Linux namespaces. Each virtual host runs inside a separate network namespace with its own set of network interfaces, IP addresses, and a routing table. A virtual host connects to one or more virtual switches through virtual network links. The network topology and link properties such as bandwidth, loss rate and delay can be configured with a extensible Python API. However, Mininet virtual hosts share the host machine file system by default. This makes it troublesome and challenging to use virtual hosts as fog nodes, since they should not use the same configuration or data files in distributed applications. Although there are ways to work around this limitation, a distinct filesystem for each virtual node can be maid by the integration of Mininet with containers.

4.2. Docker Container Platform

Docker [Docker 2017] is a software platform that allows the creation of containers. Applications running inside a Docker container and using operating-system level virtualization on Linux are similar to traditional VM instances. However, unlike VM images, containers do not bundle a full operating system. A container is a lightweight, stand-alone and executable package that includes everything needed to run an application such as executable code, runtime environments, system tools, system libraries and configuration settings.

Docker isolates the container resources using similar technologies as Mininet does for its virtual hosts, based on Linux cgroups and kernel namespaces. They both use virtual ethernet interface (vEth) pairs to connect the virtual hosts to the virtual switches. Therefore, replacing Mininet virtual hosts with Docker containers is technically feasible.

4.3. Containernet Framework

Containernet [Containernet 2017] is a software project that extends the Mininet framework to allow the use of Docker containers as virtual nodes. In addiction, it introduces important capabilities to built networking and cloud and fog testbeds [Peuster et al. 2016].

Containernet framework API enables adding, connecting and removing containers dynamically from the network topology. These features allow emulate real-world cloud and fog infrastructures in which it is possible to start and stop compute instances at any point in time. Also, it is possible to change at runtime resource limitations for a single container, such as CPU time and memory available.

5. Framework Architecture

The aim of this work is to design an architecture that enables the deployment of the main components of a fog environment with third-party systems through standard interfaces. The flexible setup is achieved by deploying virtual nodes and virtual instances into an emulated fog environment running on a host machine.

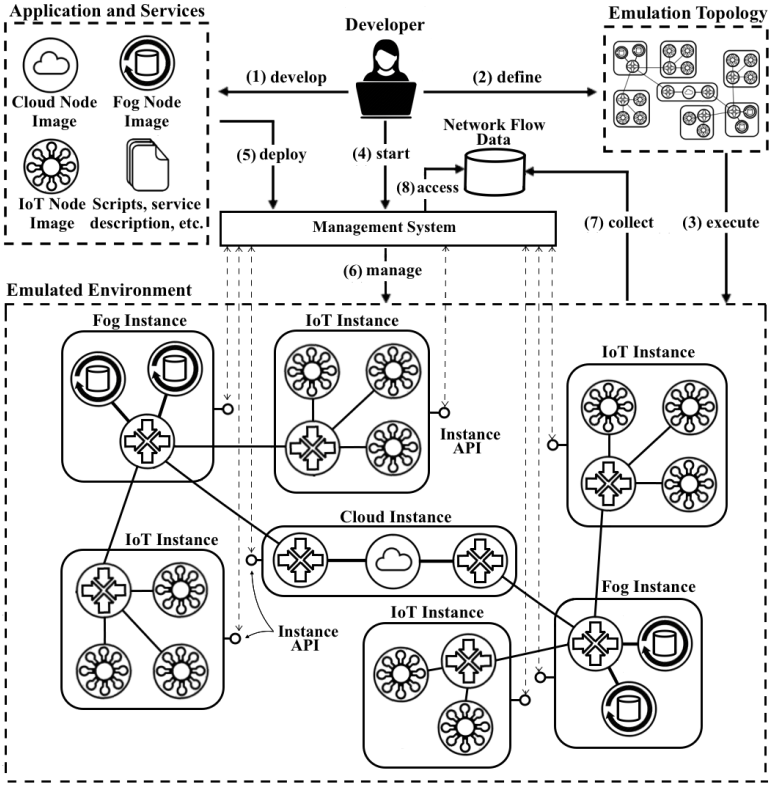


Figure 2. General emulation workflow. A example of a running emulation environment with seven virtual instances. Each virtual instance involves virtual switches and virtual nodes. The management system is responsible for uploading and starting the instances in the emulated environment. The defined topology determines the virtual connections configured between virtual nodes and virtual instances. The communication between a virtual instance and the management system is performed through a well-defined instance API.

A fog environment emulation can be created by using preconfigured container images. Each container image comprises part of a distributed application and its required services and protocols. An example of a fog environment emulation is shown in Figure 2, where three types of container images were used to instantiate different virtual nodes.

1) Cloud container image: Using this type of container image, cloud gateways to support virtualized resources for IoT applications are emulated. These virtual cloud nodes can form a virtual cloud instance (VCI) using open source technologies or act as proxies for cloud services located in remote data centers.

2) Fog container image: Using this type of container, fog nodes capable of processing and storing data collected by the IoT gateways are emulated. These virtual fog nodes can form a virtual fog instance (VFI) to support virtualized resources for IoT devices. Also, it is responsible for filtering and sending information to a VCI.

3) IoT container image: Using this type of container, smart IoT devices or IoT gateways are emulated. In a virtual IoT node, specific protocols and services to allow for communication with real sensor and actuator devices can be installed. It is also possible to use simulated sensors as data sources. These virtual IoT nodes can form a virtual IoT instance (VIoTI) to emulate a sensor network and send sensing data to a VFI.

Figure 2 depicts the high-level workflow of a developer using the environment. First, the developer provides the container images that will be instantiated to setup the emulation (1). Each container includes everything needed to run an application such as executable code, scripts, service descriptions and configuration settings. Second, the developer defines a topology on which he wants to test the application (2) and executes the emulator platform with this topology definition (3). If SDN is required, an SDN controller should be started. After the platform has been initiated, the developer starts the management system (4). The management system connects to the emulated environment by using the provided instance API. The application is deployed on the platform by the management system (5) which starts the required process and services in each virtual node. Then, the application can run inside the platform (6). The network flow statistics can be collected and stored for future analysis (7). Furthermore, the developer and the management system can access arbitrary monitoring data generated by the platform (8).

5.1. Architectural Components

The general architecture, essential components of the system as well as the relationships between those components are shown in Figure 3. In the host machine, the Mininet is the core of the system and implements the emulation environment. The Containernet extends the Mininet functionalities to support Docker containers as virtual nodes. At the top of the Containernet are the framework APIs provided for developers.

The topology API interacts with the Containernet to load and execute topology definitions and create its virtual nodes, switches, instances and connections. The instance API is flexible and allows the system to be extended with different standard interfaces. Each interface can be used by a third-party system to manage applications and services. The resource API allows the developer to apply limitation models that define the available resources for each virtual instance such as CPU time and memory capacity. In addition, the interactive command line interface (CLI) allows developers to interact with the emulated components, change configurations, view log files or run arbitrary commands while the Mininet platform executes the application and services.

The emulated environment contains virtual nodes, virtual switches and virtual connections. The Mininet process instantiates the virtual nodes from pre-created Docker container images. A container image can be instantiated more than once in an emulated environment. For example, Figure 2 shows an IoT container image being used to initiate different VIoTIs, each capable of generating IoT sensor data.

The virtual instance is an abstraction that allows for the management of related set of virtual nodes and virtual switches as a single entity. A distributed fog application and its services run in one or more virtual nodes inside a virtual instance. It is assumed that the management system has full control over which applications are executed on each virtual instance. However, it is not concerned with the internal execution details of the virtual instance. In the virtual switches, settings for flow monitoring and QoS controls are configured from the host machine. The controller process generates routing and QoS settings to enable virtual connections between virtual instances in the emulated network.

The network traffic monitoring can be implemented by running flow monitors for each network interface on virtual nodes and virtual switches. The run-time data from these processes are saved in the folders of the virtual node and host machine. Each virtual

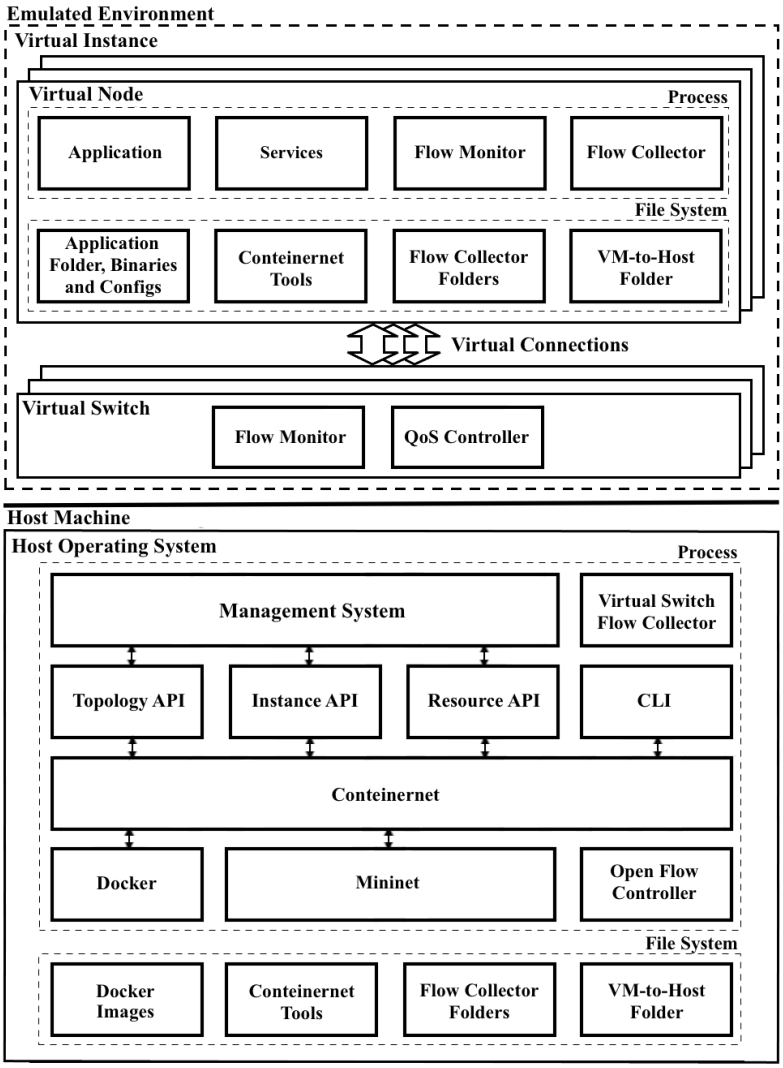


Figure 3. An overview of the architecture and principal components. The system design is customizable and provides APIs for its functionalities such as virtual instance management, container resource limitation or topology generation.

node contains a VM-to-Host folder which is connected to the same folder on the host machine. This facilitates data exchange between the virtual nodes and the host machine.

5.2. Topology API

Using the concept of virtual instances, different components can be emulated, including its links and properties. For example, a virtual node with constrained resources attached to a virtual switch can represent a virtual gateway for IoT application testing. In an upper layer, a set of virtual switches that connect multiple virtual nodes with sufficient resources to run applications can represent a virtual fog node. Depending on the hardware configuration of the host machine, small clouds can be emulated. With the use of network address translation, the virtual instance is able to communicate with external devices on the Internet or act as proxies for cloud services located in remote data centers.

The hierarchical organization of the fog architecture favors the intelligent network routing based on SDN. The switches in the emulated network are configured by standard

```
1  # create and connect virtual instances to SDN switches with different link properties
2  g1= net.addIoTInstance("IoT gateway")
3  f1= net.addFogInstance("Fog device")
4  c1= net.addCloudInstance("Cloud gateway")
5  s1 = net.addSwitch("switch from IoT to Fog")
6  s2 = net.addSwitch("switch from Fog to Cloud")
7  net.addLink(g1, s1, delay="10ms", loss=2)
8  net.addLink(f1, s1, delay="50ms")
9  net.addLink(f1, s2, delay="50ms")
10 net.addLink(c1, s2, delay="100ms")
11 # assign resource models for each virtual instance (cpu, memory and storage)
12 r1 = ResModelA(max_cu=20, max_mu=40, max_su=60)
13 r1.assignInstance(g1)
14 r2 = ResModelB(max_cu=200, max_mu=150, max_su=200)
15 r2.assignInstance(f1)
16 r3 = ResModelC(max_cu=1000, max_mu=1500, max_su=2000)
17 r3.assignInstance(c1)
18 # instantiate, connect and start a particular interface for each virtual instance
19 api1 = IoTApi(port=8001)
20 api1.connectInstance(g1)
21 api1.start()
22 api2 = FogApi(port=8002)
23 api2.connectInstance(f1)
24 api2.start()
25 api3 = CloudApi(port=8003)
26 api3.connectInstance(c1)
27 api3.start()
28 # run the emulation
29 net.start()
```

Listing 1. An example of topology script which defines virtual instances of distinct types. They are connected with different link properties. Each virtual instance uses a specific resource model and connects to a particular instance API.

SDN controllers as part of the Mininet emulation environment. Mininet supports several SDN controllers and can work with external OpenFlow controllers. With a high-level programmatic interface, sophisticated network protocols and forwarding setups can be implemented by developers. Also, these and other custom controller implementations provided by third-party solutions can be integrated with the management system.

An address scheme based on an invalid subnet allows for the support of millions of devices with IP addresses, which is sufficient to evaluate large-scale solutions. An arbitrary number of virtual switches can be added between virtual instances. Listing 1 shows an example script for defining a network topology where three instances are initiated. The topology API is based on the Mininet Python API. With this interface, developers can use executable scripts to generate different topologies in a simple and sample-based way.

5.3. Instance API

The management system needs to interact with the emulated environment to control virtual nodes within the virtual instances. The instance API provides an infrastructure-as-a-service (IaaS) semantics to manage virtual nodes in an adaptable way. It is designed as an abstract interface to allow the integration and testing of third-part management systems.

The developers can implement their own management interfaces on top of a virtual instance API. The default approach is adding one specific instance API to each type of virtual instance. These instance APIs are assigned to virtual instances in the topology scripts (Listing 1, lines 19–27). With this flexible conception, it is possible the execution of different management strategies for each virtual instance in the emulated environment.

5.4. Resource API

Fog environments are characterized by the existence of network nodes with different capacities and resources. While cloud services provide virtually infinite compute resources to their clients, fog devices and IoT gateways offer limited computing, memory, and storage resources to applications. These different scenarios must be considered by a management system when offloading and resource allocation decisions are taken.

To emulate such resource limitations, the emulation architecture offers the concept of resource models assigned to each virtual instance (Listing 1 lines 12–17). These models are invoked when containers are allocated or released. They are used to compute CPU time, memory, and storage limits for each virtual instance. Therefore, it is possible for a virtual instance to reject allocation requests from the management system if there are no free resources available. The resource API allows developers to create specific testing scenarios involving instances with different capacities in the emulated environment.

5.5. Flow Monitoring and Collection

A scheme to observe the behavior of the environment can be implemented by running network flow monitors and collectors. For implementation example, it is possible employs NetFlow [B. Claise, Ed. 2004] as the flow monitoring technology and NFDUMP [Haag, P. 2011] as the NetFlow collector and analysis tools. In this monitoring scheme, flow records can be sent to a specified *nfcapd* flow capture daemon using the command *fprobe* in virtual nodes and the command *ovs-vsctl* in virtual switches. The *nfdump* or other compatible flow analysis tool can be used to study the collected traffic.

The capture daemon running on a virtual node is responsible for collecting and saving flow statistics from the flow monitors of the virtual node. It is necessary to start one flow monitor process for each network interface. The traffic on the virtual nodes may overlap with the traffic monitored on virtual switches interfaces.

In the Containernet, virtual switches are not Docker containers. Just like in the Mininet, they are created in the host machine. Consequently, the flow monitor commands and capture daemons should be run in the host machine. A flow capture daemon running in the host machine is responsible for collecting and saving flow statistics from a specific virtual switch. Each flow capture daemon saves the flow records to a different directory. In this manner, is possible separate the collected data from different virtual switches.

6. Conclusion and Future Work

In this work, the design of a fog emulation testbed based on feasible deployment features of existing technologies was presented. The strengths of the proposed architecture are low cost, flexibility and support to perform real world protocols and technologies.

The current components were designed to prototype and test complex network services in a realistic environment running on low-cost machines. Such computers may not have processors with advanced virtualization functions to support the installation of the current cloud middleware. However, the proposed components can be extended to enable its connection to real cloud data centers.

The use of the same pre-created container images to start more than one container instances makes the environment setup process flexible. New types of container images

can be deployed to allow for the testing of different IoT solutions inside a fog architecture. Also, it makes easier the testing of solutions that deal with massive amounts of data streams such as big data applications. In a simple procedure, it is possible to build fog environments with multiple IoT data sources from a few container images.

The provision of standard interfaces allows for the testing of the third-party systems for functions such as resource management, virtualization, and service orchestration. For example, emerging systems with support to network function virtualization (NFV) and virtual network functions (VNF) can be connected with fog platforms to create complex network services (NS), which are controlled by a management and orchestration (MANO) system [Karl et al. 2016] [Sonkoly et al. 2015]. Also, it allows developers to use their tested applications into real world environments with minimal changes. In addition, the future progress in developing standard interfaces for fog computing can facilitate the integration of the third-party systems in different application scenarios.

Another important aspect of the architecture is the fidelity to the fog computing model. Using containers, it is possible to offer virtual fog nodes with computing resources in the emulated network. An advance in this regard is the use of containers not only in virtual nodes but also in the virtual switches created by Mininet. However, such capabilities require significant changes in Mininet software and as such were not considered. The proposed architecture remains faithful to the fog model by enabling the developers to define fog instances that treat virtual nodes and switches as a single entity.

The principal future objective of this work is to enable the testing of distributed algorithms. Possible experiments may range from fog applications to the services related to the fog infrastructure itself. The provided features include a built-in scheme to observe the behavior of the environment by monitoring network flow inside the virtual nodes and switches. It provides insightful network traffic information of a running application so that researchers can evaluate their algorithms and network configurations in terms of system performance. It is possible to incorporate techniques for data visualization and real-time analysis in different levels of the network hierarchy to help developers debug problems and measure the performance of applications and services. Aspects such as security, fault tolerance and reliable management can be investigated by developing functionalities to allow the insertion of simulated attacks and problems in the emulated environment. Finally, further research concerning the presented architecture and its components can allow for the use of emulators for realistic and reproducible fog experiments.

References

Aazam, M., Khan, I., Alsaffar, A. A., and Huh, E.-N. (2014). Cloud of things: Integrating internet of things and cloud computing and the issues involved. In *Applied Sciences and Technology (IBCAST), 2014 11th International Bhurban Conference on*, pages 414–419. IEEE.

Adjih, C., Baccelli, E., Fleury, E., Harter, G., Mitton, N., Noel, T., Pissard-Gibollet, R., Saint-Marcel, F., Schreiner, G., Vandaele, J., et al. (2015). Fit iot-lab: A large scale open experimental iot testbed. In *Internet of Things (WF-IoT), 2015 IEEE 2nd World Forum on*, pages 459–464. IEEE.

B. Claise, Ed. (2004). Cisco systems netflow services export version 9. Available: https://tools.ietf.org/html/rfc3954. Accessed: 20 fev. 2017.

Bonomi, F., Milito, R., Natarajan, P., and Zhu, J. (2014). Fog computing: A platform for internet of things and analytics. In *Big Data and Internet of Things: A Roadmap for Smart Environments*, pages 169–186. Springer.

Bonomi, F., Milito, R., Zhu, J., and Addepalli, S. (2012). Fog computing and its role in the internet of things. In *Proceedings of the first edition of the MCC workshop on Mobile cloud computing*, pages 13–16. ACM.

Calheiros, R. N., Ranjan, R., Beloglazov, A., De Rose, C. A., and Buyya, R. (2011). Cloudsim: a toolkit for modeling and simulation of cloud computing environments and evaluation of resource provisioning algorithms. *Software: Practice and experience*, 41(1):23–50.

Cisco IOx (2017). product page. Available: http://www.cisco.com/products/cloud-systems-management/iox/. Accessed: 20 fev. 2017.

Containernet (2017). project and source code. Available: https://github.com/containernet/containernet. Accessed: 20 fev. 2017.

Coutinho, A. A. T. R., Greve, F. G. P., and Carneiro, E. O. (2016). Computação em névoa: conceitos, aplicações e desafios. In *Minicursos - XXXIV Simpósio Brasileiro de Redes de Computadores e Sistemas Distribuídos*, pages 266–315. SBC.

Dastjerdi, A. V., Gupta, H., Calheiros, R. N., Ghosh, S. K., and Buyya, R. (2016). Fog computing: Principles, architectures, and applications. *arXiv preprint arXiv:1601.02752*.

de Oliveira, R. L. S., Shinoda, A. A., Schweitzer, C. M., and Prete, L. R. (2014). Using mininet for emulation and prototyping software-defined networks. In *Communications and Computing (COLCOM), 2014 IEEE Colombian Conference on*, pages 1–6. IEEE.

DevNet (2017). Cisco IOx developer tools and applications templates for download. Available: https://developer.cisco.com/site/iox/. Accessed: 20 fev. 2017.

Dinh, H. T., Lee, C., Niyato, D., and Wang, P. (2013). A survey of mobile cloud computing: architecture, applications, and approaches. *Wireless communications and mobile computing*, 13(18):1587–1611.

Docker (2017). home page. Available: https://www.docker.com. Accessed: 20 fev. 2017.

Fog Director (2017). product page. Available: http://www.cisco.com/c/en/us/products/cloud-systems-management/fog-director/. Accessed: 20 fev. 2017.

Gupta, H., Dastjerdi, A. V., Ghosh, S. K., and Buyya, R. (2016). ifogsim: A toolkit for modeling and simulation of resource management techniques in internet of things, edge and fog computing environments. *arXiv preprint arXiv:1606.02007*.

Haag, P. (2011). Nfdump-netflow processing tools. Available: http://nfdump.sourceforge.net. Accessed: 20 fev. 2017.

Hu, Y. C., Patel, M., Sabella, D., Sprecher, N., and Young, V. (2015). Mobile edge computing-a key technology towards 5g. *ETSI White Paper*, 11.

Karl, H., Dräxler, S., Peuster, M., Galis, A., Bredel, M., Ramos, A., Martrat, J., Siddiqui, M. S., van Rossem, S., Tavernier, W., et al. (2016). Devops for network function vir-

tualisation: an architectural approach. *Transactions on Emerging Telecommunications Technologies*, 27(9):1206–1215.

McKeown, N., Anderson, T., Balakrishnan, H., Parulkar, G., Peterson, L., Rexford, J., Shenker, S., and Turner, J. (2008). Openflow: enabling innovation in campus networks. *ACM SIGCOMM Computer Communication Review*, 38(2):69–74.

NS-3 (2017). Simulator home page. Available: https://www.nsnam.org. Accessed: 20 fev. 2017.

OpenFog (2017a). Consortium home page. Available: http://www.openfogconsortium.org/. Accessed: 20 fev. 2017.

OpenFog (2017b). Reference architecture for fog computing. Available: http://www.openfogconsortium.org/ra/. Accessed: 20 fev. 2017.

Openstack (2017). home page. Available: https://www.openstack.org. Accessed: 20 fev. 2017.

Peuster, M., Karl, H., and Van Rossem, S. (2016). Medicine: Rapid prototyping of production-ready network services in multi-pop environments. *arXiv preprint arXiv:1606.05995*.

Sanchez, L., Muñoz, L., Galache, J. A., Sotres, P., Santana, J. R., Gutierrez, V., Ramdhany, R., Gluhak, A., Krco, S., Theodoridis, E., et al. (2014). Smartsantander: Iot experimentation over a smart city testbed. *Computer Networks*, 61:217–238.

Satyanarayanan, M., Bahl, P., Caceres, R., and Davies, N. (2009). The case for vm-based cloudlets in mobile computing. *IEEE pervasive Computing*, 8(4).

Sonkoly, B., Czentye, J., Szabo, R., Jocha, D., Elek, J., Sahhaf, S., Tavernier, W., and Risso, F. (2015). Multi-domain service orchestration over networks and clouds: a unified approach. *ACM SIGCOMM Computer Communication Review*, 45(4):377–378.

Stojmenovic, I. (2014). Fog computing: A cloud to the ground support for smart things and machine-to-machine networks. In *Telecommunication Networks and Applications Conference (ATNAC), 2014 Australasian*, pages 117–122. IEEE.

Tordera, E. M., Masip-Bruin, X., Garcia-Alminana, J., Jukan, A., Ren, G.-J., Zhu, J., and Farre, J. (2016). What is a fog node a tutorial on current concepts towards a common definition. *arXiv preprint arXiv:1611.09193*.

Varga, A. and Hornig, R. (2008). An overview of the omnet++ simulation environment. In *Proceedings of the 1st international conference on Simulation tools and techniques for communications, networks and systems & workshops*, page 60. ICST (Institute for Computer Sciences, Social-Informatics and Telecommunications Engineering).

Yi, S., Hao, Z., Qin, Z., and Li, Q. (2015a). Fog computing: Platform and applications. In *Hot Topics in Web Systems and Technologies (HotWeb), 2015 Third IEEE Workshop on*, pages 73–78. IEEE.

Yi, S., Li, C., and Li, Q. (2015b). A survey of fog computing: concepts, applications and issues. In *Proceedings of the 2015 Workshop on Mobile Big Data*, pages 37–42. ACM.

XV Workshop de Computação em Clouds e Aplicações

SBRC 2017

Sessão Técnica 3

Geração de carga de trabalho transiente para aplicações de *e-commerce* multicamadas

Lourenço Alves Pereira Júnior[1,2], Flávio Luiz dos Santos de Souza[2,3], Edwin Luis Choquehuanca Mamani[2], Francisco José Monaco[2]

[1]Instituto Federal de São Paulo — IFSP

ljr@ifsp.edu.br

[2]Universidade de São Paulo — USP

{ljr,flaviols,edwin,monaco}@icmc.usp.br

[3]Universidade de Franca — UNIFRAN

flavio.souza@unifran.edu.br

Abstract. *Considering the great adoption of cloud computing as operational platform by e-commerce retailers and its dynamical environment setup, workload on this systems is basically time-varying. Caused by user bahavioral patterns, as well as flash crowds and virals, the system trespass differents operational regions leading to transients. In a scenario where transients is commonplace, the proper study of this phenomenon is important. Although, in spite of many benchmarks whose purpose is load systems with synthetic work, it is not usual to find an interface to express how workload must change on the fly during an experimental execution, as a matter of concern. Thus, this papers presents the implementation of a time-varying workload generator as an extension of the Bench4Q tool. The results point towards the importance of applying such kind of performance tests by allowing the apprisal of system's transients when using IBM DB2 and PostgreSQL, revealing the dynamic behavior of the performance metric response time.*

Resumo. *Dado o ambiente dinâmico proporcionado por ambientes de computação e sua grande adoção como plataforma operacional de lojas virtuais (*e-commerce*), a carga de trabalho característica desses sistemas sofre flutuações ao logo de sua execução. Ocasionada por padrões de acesso dos usuários, promoções relâmpago e virais de redes sociais, a carga de trabalho imposta ao sistema o expõe a diferentes patamares de utilização. A mudança entre esses patamares faz com que regimes operacionais transientes sejam comuns, o que torna seu estudo importante. No entanto, apesar de existirem muitas ferramentas de* benchmark*, poucas tem a preocupação de disponibilizar uma interface em que seja possível expressar as variações que devem ocorrer durante os experimentos. Assim, este trabalho apresenta a implementação de um gerador de carga de trabalho variante no tempo. Os resultados evidenciam a importância de se aplicar testes de desempenho com uma ferramenta como essa, proporcionando a apreciação de diferentes padrões comportamentais ao utilizar os banco de dados IBM DB2 e PostgreSQL, revelando o comportamento dinâmico da métrica tempo de resposta.*

1. Introdução

A caracterização e a geração de carga de trabalho para sistemas computacionais é uma área de pesquisa consolidada, abrangendo vários tipos de aplicações como, por exemplo, cargas para servidores web, redes sociais, serviços de distribuição de vídeo, dispositivos móveis, computação em nuvem e *datacenter* [Calzarossa et al. 2016]. Observa-se que cada vez mais aplicações Web são migradas para ambientes de computação em nuvem, cuja motivação se dá pela elasticidade, que permite o correto dimensionamento da capacidade computacional em função da demanda [Qu et al. 2016]. As flutuações na demanda de serviços são cada vez mais frequentes (oriundas de padrões de utilização de serviços, campanhas virais, promoções relâmpago etc.), e novas abordagens são experimentadas em ambientes reais de empresas de grande porte, como é o caso da Netflix [Yuan et al. 2013]. Nesse cenário, destaca-se a natureza variante das cargas de trabalhos das aplicações atuais, de forma que começa a ser um requisito para avaliação de desempenho propostas de mecanismos que permitam expressar alterações na carga de trabalho em execução.

A abordagem tradicional para avaliação de desempenho consta da representação de sistemas com modelos regressivos ou baseados em Teoria de Filas que capturam o comportamento do sistema em regime estacionário [Jain 1990, Menasce et al. 2004]. De forma análoga, a caracterização da carga de trabalho é feita pela obtenção de traços de execuções reais, sua análise e respectivo ajuste da funções de probabilidade estatística que melhor adéqua (*fit*) aos dados. Essa caracterização também é estacionária, e sua aplicação é realizada por meio de um gerador de números pseudo-aleatório [Feitelson 2015]. Nesses casos, a prática consiste em descartar os dados de regime transiente e aproveitar somente aqueles estacionários. No entanto, no domínio deste trabalho, as aplicações estão suscetíveis a cargas variantes no tempo e, por consequência, estados operacionais transientes são comuns e podem conter informações importantes.

Iniciado em meados da década de 1990, um novo paradigma de implementação de controladores de recursos de sistemas computacionais baseado em Teoria de Controle vem sendo aplicado até os dias atuais nos mais diversos domínios de aplicação, como é o caso de sistemas Web de grande porte [Hellerstein et al. 2004b]. Esse paradigma tem a vantagem de ser possível representar o transiente dos sistemas computacionais, trazendo todo um arcabouço numérico que possibilita a utilização de métricas de desempenho antes inéditas em ciência da computação. Por exemplo, tempo de assentamento (*settling time*), sobre-passagem (*overshoot*) e ganho. No entanto, observa-se uma abordagem pontual (ou *ad-hoc*) para solução de problemas [Huang et al. 2014]. Ressalta-se a necessidade da execução de experimentos específicos para a correta modelagem de sistemas para aplicação de Teoria de Controle.

Como proposto por [Mamani et al. 2015], pode-se derivar um arcabouço genérico para avaliação de desempenho de sistemas computacionais, de modo que seja possível estabelecer um conjunto de requisitos para sua efetiva implementação [Pereira et al. 2015a], bem como tratar adequadamente os dados experimentais [da Luz et al. 2016]. Assim, pode-se utilizar de um conjunto de ferramentas para experimentação e análise genéricas o suficiente para serem aplicadas em diversas aplicações. Ressalta-se principalmente o caso em que o desempenho seja em função da carga de trabalho [Pereira 2016]. É importante mencionar que, para este tipo de avaliação, o sistema é modelado como um sistema dinâmico o que permite um novo olhar sobre a geração de carga traba-

lho [Pereira et al. 2017]. O modelo conceitual MEDC (*Monitor, Effector, Demand, and Capacity*) [Pereira et al. 2015a, Pereira et al. 2015b] especifica responsabilidades que permitem a realização de um modelo dinâmico do sistema. Consistindo do monitoramento periódico de métricas de desempenho que são passadas para as responsabilidades *demand* e *capacity* a fim de que seja possível implementar flutuações na demanda do serviço em modelagem e aplicar políticas de ajuste de capacidade em tempo de execução. *Effector* aplica de fato as alterações no sistema, podendo inserir elementos que deturpam a dinâmica do sistema (como adição de falhas).

Dada a importância desse tema, evidenciado pela característica de elasticidade da computação em nuvem e a escassez de ferramentas que permitem gerar transientes de forma sintética em serviços computacionais, neste trabalho procuramos entender como gerar tais variações no regime operacional. Este artigo, portanto, tem o objetivo *1) propor uma interface que permita a especificação do comportamento de uma carga de trabalho que varia durante o experimento e 2) demonstrar sua utilização por meio de um estudo de caso com os SGBDs PostgreSQL e IBM DB2*. Os resultados foram relevantes, expondo que o transiente pode levar o sistema condições extremas de curta duração. Observamos que transiente é absorvido pelo sistema e segue uma certa "inércia". Seu negligenciamento pode levar à especificação de políticas de *auto-scaling* que desconsideram o comportamento do sistema, o que implicaria em aumento de custos para a aplicação.

A seguir, na Seção 2 sao discutidos trabalhos relacionados. Na Seção 3 é a presentado o OnlineBench4Q e as alterações necessárias para que o gerador de carga de trabalho variante no tempo. Na Seção 4 é detalhado como a geração de carga de trabalho é feita. A Seção5 apresenta um estudo de caso com IBM DB2 e PostgreSQL. Por fim, na Seção 6 conclui-se este estudo.

2. Trabalhos relacionados

Os serviços de carga automatizada em ambientes de nuvem são muito comuns como pode-se citar o Loader.io [Loader.io 2017] e NewRelic [NewRelic 2017]. Eles não implementam funcionalidades de permitem expressar cargas transientes com características periódicas e formatos específicos (senoidal, degrau, PRBS etc.) que auxiliam no processo de identificação do sistema — NewRelic e Loader.io apresentam uma demanda crescente progressiva (rampa). De mesma forma, trabalhos científicos para geração de carga de trabalho distribuída e em nuvem ainda não contemplam nativamente a alteração das características da carga de trabalho em tempo de execução [Ferreira et al. 2016]. O presente trabalho tem a preocupação de fornecer uma interface flexível na qual o usuário seja capaz de especificar como as variações na carga serão executadas em ambiente de *e-commerce*.

Kraken, uma ferramenta de teste de carga para *datacenters* do Facebook [Veeraraghavan et al. 2016], realiza testes realísticos ao permitir que uma fração da carga real do sistema em produção seja direcionada a um *cluster* e assim verificar qual a carga máxima suportada pelo sistema. Os resultados evidenciam que a capacidade individual de uma máquina não corresponde proporcionalmente a capacidade de todas as máquinas no *cluster*. Destacaram que os resultados dos testes serviram como *feedback* para a equipe de desenvolvimento promover melhorias. O trabalho do Facebook corrobora para o presente artigo mostrando a relevância do tema, uma vez que descrevem a presença de transientes no ambiente real deles e as alterações realizadas não impactam

instantaneamente no desempenho observado. O contexto e a motivação para nossa proposta estão alinhados com o que foi descrito nos experimentos realizados com o Kraken. A saber, um cenário com comportamento emergente (o desempenho do todo é resultado da combinação das partes) e transiente apreciável. Nosso trabalho difere em relação à carga de trabalho (nossa sintética e a deles a carga real do sistema), à escala (nossa célula computacional é composta por apenas 24 máquinas físicas) e à aplicação (deles rede social e nosso uma aplicação de *e-commerce*).

Trabalhos como [Babu et al. 2009] e [Zheng et al. 2009] exploram a execução de experimentos em ambientes reais em que a carga é oriunda dos próprios clientes. Isso é um ponto positivo, no entanto, a abordagem é concentrada em produzir resultados que respondam a questões do tipo "e-se" (*what-if*). O contexto de nossa proposta é que experimentos *off-line* podem ser realizados a fim de se derivar um modelo capaz de representar o desempenho sistema em diferentes cenários e com diferentes cargas. O princípio é que, após a obtenção de um modelo de Função de Transferência, simulações são executadas e seu resultado é acurado o suficiente para que não seja necessária a execução de casos *what-if* no sistema real, evitando custos para a obtenção das respostas. Uma limitação desta proposta é que o desempenho pode ser predito em uma região de operação especificada. Detalhes desta abordagem podem ser observados em [Pereira et al. 2017] e [Yang and Liu 2012]. Nossa proposta também serve como ferramental para a expressão dos experimentos específicos executados de forma *off-line*.

Ao focar em soluções específicas para aplicação de *e-commerce*, destaca-se o TPC-W, um *benchmark* para avaliação de sistemas de *e-commerce* [Menasce 2002], que utilizada o grafo CBMG (*Custormer Behavioral Model Graph*) [Menascé et al. 1999] para especificar as transições que um cliente realiza no sistema em teste. O Bench4Q [Zhang et al. 2011] é uma extensão do TPC-W que adiciona métricas baseadas em sessões de usuários para garantir Qualidade de Serviço (QoS). O presente trabalho faz uso do Bench4Q como base de implementação de um *benchmark* capaz de gerar carga variante no tempo. Tanto o TPC-W quanto o Bench4Q não oferecem suporte adequado para análise de desempenho transiente.

Como foi evidenciado, o presente trabalho diferencia-se dos demais por dois motivos: 1) o contexto de sua aplicação, avaliação de desempenho não estacionário e 2) sua flexibilidade de configuração permitindo especificação de cargas de trabalho clássicas para identificação de sistemas (degrau, rampa, senoidal, trem de pulsos e PRBS), bem como especificadas pelo usuário. A implementação segue o modelo de requisitos MEDC aplicada ao Bench4Q, dando origem ao OnlineBench4Q destalhado a seguir.

3. Implementação do gerador de carga para o OnlineBench4Q

Com o proposito de estimular o sistema a apresentar a sua dinâmica e analisar o desempenho transiente do sistema, o presente trabalho estende o gerador de carga de trabalho do *benchmark* Bench4Q. O Bench4Q é uma extensão do TPC-W [Menasce 2002], e tem por objetivo o *tuning* de aplicações *e-commerce* com requisitos de QoS. As principais características do Bench4Q incluem: apoio à análise de métricas baseada em sessão que simula carga sensível a QoS para uma análise da capacidade.

O Bench4Q oferece uma arquitetura distribuída para a geração de carga através de seus agentes que são conectados a um único console que os gerencia, por meio do

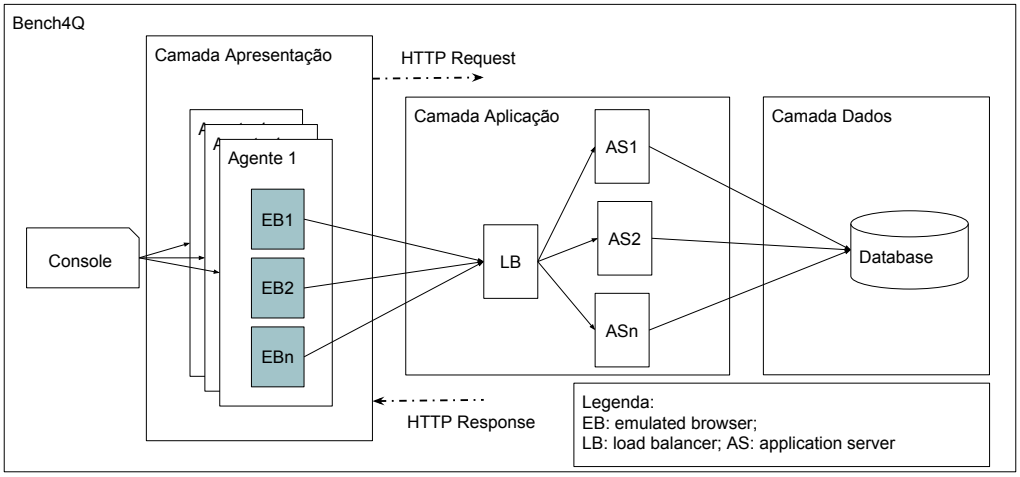
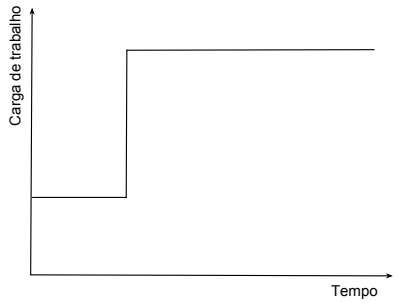


Figura 1. Arquitetura do Bench4Q. Console é uma interface de configuração de experimentos. A camada de apresentação é formada por agentes que possuem *browsers* emulados que fazem e recebem requisições ao/do sistema em teste (SUT). A camada de aplicação é composta por um balanceador de carga que repassa as requisições aos servidores de aplicação. A camada de dados é instanciada por um SGBD. O SUT é composto pelas camadas Aplicação e Dados.

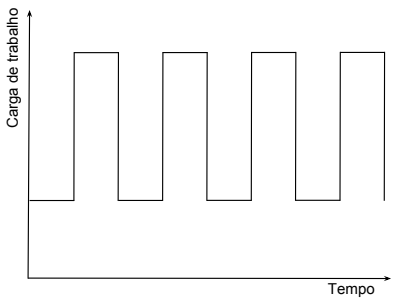
qual é possível ajustar separadamente as configurações para cada agente. Esses agentes geram carga (requisições HTTP) para o servidor de aplicação onde está hospedado o *e-commerce*, como ilustrado na Figura 1. Os resultados da avaliação de carga aplicada ao *e-commerce* são coletados pelo Console, que apresenta alguns gráficos, os quais facilitam a interpretação da avaliação mediante as diretrizes do TPC-W [Zhang et al. 2011].

A ferramenta é composta por três partes: Console, Agente e SUT (Figura 1), e também disponibiliza interfaces para o monitoramento de recursos para o servidor de aplicação e para o banco de dados; este monitoramento inclui CPU, memória, rede e tempo de resposta. O `console` é a entidade que recebe as configurações do experimento, permitindo que seja especificada a quantidade de `agentes` a serem instanciados, bem como a quantidade de *browsers* emulados (EB — *emulated browsers*) em cada agente. Os EBs comportam-se como clientes para o sistema em estudo (SUT — *System Under Test*). No entanto, o Bench4Q não é capaz de modular a carga de trabalho gerada a fim de permitir observação, análise e estudo do comportamento dinâmico do sistema. A proposta da extensão do Bench4Q é manter o padrão de usabilidade e possibilitar a modulação da carga de trabalho atrás de uma interface gráfica. Sendo assim, com o preenchimento de um conjunto de parâmetros será possível a geração modulada da carga:

- **Tempo de planejamento de carga:** Um período de tempo em que a carga de trabalho é modulada, caracterizando a mudança do comportamento das requisições de maneira programada;
- **Tipo de modulação:** conforme as funções ou sinais propostos por [Hellerstein et al. 2004b];
- **Tempo de interrupções:** Período de interrupções/pausa após o *Tempo de planejamento de carga*;
- **Quantidade de clientes na modulação:** reservar uma quantidade de clientes EBs, que estão com dedicação exclusiva para a modulação da carga.



(a) Formato Degrau. Pode ser positivo ou negativo.



(b) Trem de pulsos. Pode alterar a duração e a amplitude dos pulsos.

Figura 2. Modulação das cargas. O formato degrau permite realizar uma alteração brusca e duradoura. O trem de pulsos permite a criação de oscilações na carga de trabalho. A rampa pode ser derivada do trem de pulsos ao alterar progressivamente as amplitudes e manter fixa a duração das mudanças.

Através dessa interface é possível parametrizar a modulação da cargas conforme os exemplos apresentados na Figura 2. Esses exemplos são derivados das funções apresentadas por [Hellerstein et al. 2004b]. Com esses tipos de variações em tempo de execução, é possível estimular o sistema de maneira a expor sua dinâmica. É importante criar cargas de trabalho que possam ser utilizadas em estudos de avaliação de desempenho não estacionária e emular alterações no regime operacional da aplicação.

A solução proposta é simples e parte da especificação de uma unidade mínima de perturbação denominada pulso. Com ela é possível especificar o **início** em que a perturbação ocorrerá, sua **duração**, quanto tempo em **pausa** e a **amplitude** da perturbação. Deve-se considerar que a execução e um experimento iniciará em uma patamar estacionário de carga e após o tempo estipulado em *início* a programação das perturbações acontecerão. Assim, a especificação de uma perturbação do tipo degrau (Figura 2(a)), carga em que há uma perturbação que dura até o final do experimento, é feita escolhendo um instante de tempo para seu início, sua duração em conformidade com o tempo de execução do experimento, pausa como nula (valor zero) e amplitude a intensidade de carga de trabalho a ser adicionada ao sistema. A unidade mínima serve como bloco (*building block*) para a especificações de outras cargas de trabalho como: pulso unitário, trem de pulsos, PRBS etc. O PRBS (*pseudo-random binary signal*) é uma sequência de pulsos com variações na duração e tem como característica comportar-se como uma carga de trabalho neutra e ao mesmo tempo capaz de expor a dinâmica do sistema em estudo, maiores detalhes disponíveis em [Ljung 1999].

Ao ler a configuração do experimento, o `console` calcula a quantidade máxima de EBs a serem utilizados no experimento — denominada `tEBs`. Na sequência, instancia os `agentes` e cria uma quantidade de EBs balanceada em cada agente. Os EBs possuem dois estados básicos: ativos e inativos. Após a instânciação inicial, um subconjunto de `tEBs` é configurado no estado de ativo e o restante como inativo. Após o período inicial, alterna-se aqueles em estado inativo para ativo, cuja consequência é aumentar a carga. De modo análogo, pode-se diminuir a quantidade de EBs ativos na intenção de impor uma carga menos severa ao sistema. A Figura 3 ilustra a criação dos dois grupos de modo que os EBs ativos e inativos transitam de um grupo para outro.

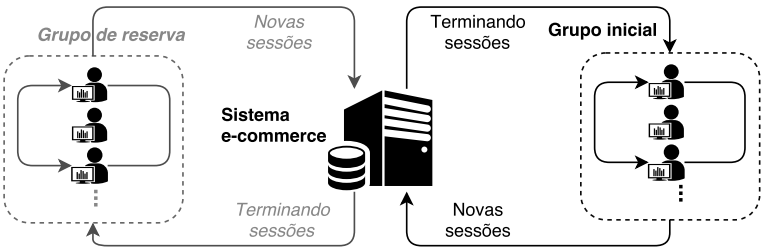


Figura 3. O mecanismo de modulação de carga prevê a criação de dois grupos: um contendo clientes ativos e outro com inativos. Durante a execução do experimento os clientes são alternados entre os grupos o que gera aumento/diminuição na carga de trabalho imposta ao sistema.

A validação desse modelo de geração de carga foi feito através da implementação de um protótipo denominado OnlineBench4Q. O termo *online* refere-se ao fato de ser possível alterar a estacionariedade da carga de trabalho em tempo de execução. Como base para sua implementação foi utilizado o modelo conceitual MEDC [Pereira et al. 2015a]. A alteração descrita neste artigo corresponde à responsabilidade *demand* do MEDC. A implementação completa conta com elasticidade de recursos computacionais (*capacity*), gerenciamento eficiente dos estados das máquinas virtuais (*effector*) e monitoramento integrado para o ambiente controlado de testes. O detalhamento da implementação encontra-se em [Mamani 2016, Souza 2016].

4. Geração da carga de trabalho

Como uma extensão do Bench4Q, o OnlineBench4Q adiciona as responsabilidades especificadas pelo MEDC e aproveita as funcionalidades da versão original. Dessa forma, iniciamos com a descrição das propriedades da carga de trabalho sintética gerada pelo Bench4Q que corresponde àquelas utilizadas em nossos experimentos de validação.

Conforme descrito em [Zhang et al. 2011], o Bench4Q utiliza configurações para descrever como a carga de trabalho impactará no SUT. O primeiro parâmetro diz respeito aos clientes que geram a caga, podendo ser fechada, com um número fixo de clientes que realizam requisições e recebem respostas até o final do experimento, e aberta, com a criação de um cliente que realiza um número k de requisições e depois deixa o sistema. Nossos experimentos baseiam-se na configuração aberta, com o intuito de representar o comportamento típico de um site de *e-commerce* que recebe clientes ao longo de sua operação.

Outro parâmetro para a carga de trabalho diz respeito à quantidade de operações de navegação e compras realizadas pelos clientes. As operações de navegação tem um custo menor para o sistema, pois resultam em consultas, cujos resultados podem ser armazenados temporariamente (*cache*) e não requerem tratamento especial para lidar com concorrência, haja vista que o tipo de sistema de gerenciamento de banco de dados mantém as propriedades ACID (atomicidade, consistência, isolamento e durabiliadde). Há três configurações padrão: `browsing` (95% navegação e 5% compra), `shopping` (80% e 20%), `ordering` (50% e 50%). Nossa configuração foi feita como `browsing`. Os valores que determinam a convergência para navegação ou compra são especificados através

Tabela 1. Configuração do hardware utilizado nos experimentos.

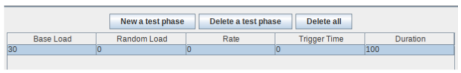
Nome	CPU (GHz)	Mem. (GB)	HD (Mb/s)	SO	#
Load Balancer	INTEL i5-3330-(3.0) x4	7,8	87,74	ClearOS 6.6	1
DB server (pool)	AMD Q6600-(2.4) x4	7,8	71,21	Ubuntu 14.04.2	8
DB server (master)	FX-8320-(3.5) x8	23	285,58	Ubuntu 14.04.3	1
Clientes	AMD Q6600-(2.4) x4	7,8	76,96	Ubuntu 14.04.2	9
Hipervisor	X5660-(2.8) x12	11	251,98	Ubuntu 15.04	1
Servidor de VMs	X5660-(2.8) x12	11	251,98	Ubuntu 15.04	4
VMs	QEMU Virtual x1	1	178,60	Ubuntu 12.04.5	-

do grafo CBMG que especifica como as transições entre as páginas ocorrerão durante as sessões dos clientes. O CBMG utilizado foi aquele especificado por padrão no Bench4Q.

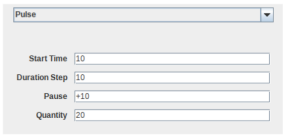
O tamanho das sessões e o intervalo entre as requisições foram configurados como uma distribuição exponencial, sem alterações aos valores padrão do Bench4Q. A taxa de tolerância a atrasos é descrita por uma distribuição normal. Ao iniciar a execução de um experimento, requisições começarão ser realizadas no ponto de acesso do SUT e assim gerarão carga ao sistema. Haverá um período de tempo no início e no final de cada experimento denotado como aquecimento (*warm-up*) e resfriamento (*cool down*), no entanto, a carga imposta ao sistema é tipicamente estacionária. A configuração da capacidade pode ser realizada de forma empírica, realizando experimentos na implementação real e verificando uma métrica de desempenho, por exemplo a taxa de utilização do sistema (CPU) em função da quantidade de requisições por segundo.

A prova de conceito ocorreu por experimentos que geraram degrau, pulso positivo e negativo e trem de pulsos. Por limitações de espaço, listamos aqui a validação por meio de trem de pulso.A descrição completa pode ser encontrada em [Souza 2016]. O experimentos realizados consideraram uma carga inicial de 30 EBs (`tEBs`) com uma perturbação do tipo trem de pulsos em que a duração dos pulsos corresponde a 10 s, e a quantidade de EBs que serão ativados/desativos é de 20 EBs. O ambiente físico foi configurado conforme Tabela 1, interligadas por três enlaces Ethernet de 1 Gbps. O Balanceador de Carga possui duas interfaces, uma para a rede cliente e outra para o *cluster* do *hypervisor* e servidores de VMs; as máquinas de BD são isoladas e podem ser acessadas a partir dos servidores de aplicação. Foram configuradas oito máquinas para agentes, um balanceador de carga, quatro máquinas virtuais (VMs) como servidores de aplicação e um banco de dados. A Figura 4 apresenta os resultados obtidos. As imagens apresentam o conteúdo de telas do OnlineBench4Q geradas a partir do `console`. A configuração de uma carga de trabalho do tipo pulso (Figura 4(a)) e que a quantidade de requisições foi conforme especificado durante a execução do experimento (Figura 4(b)). Desse modo, conclui-se que a ferramenta executou a carga de trabalho conforme especificada.

Os resultados da validação apenas demonstram como é feita a configuração do sistema, mas não revelam sua potencial utilidade como ferramenta de avaliação de desempenho de modo a levar o sistema a condições em que seja possível apreciar seu comportamento dinâmico. Na próxima Seção tratamos desta questão e mostramos a comparação do desempenho do sistema ao alterar o banco de dados de PostgreSQL para IBM DB2.

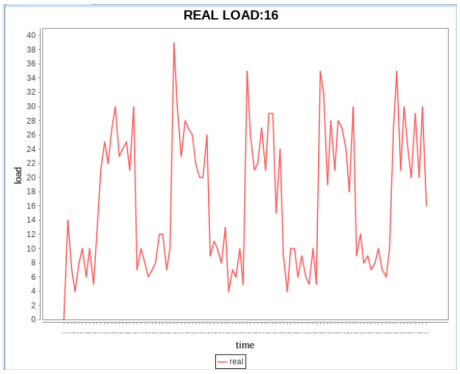


Configuração da carga no Bench4Q, para uma onda quadrada



Configuração para modular a carga como uma onda quadrada

(a) Interface gráfica do `console` para a configuração da carga de trabalho.



(b) Gráfico apresentado no `console` contendo os resultados da execução do experimento. Eixo das coordenadas o tempo e nas abssissas a quantidade de requisições por segundo.

Figura 4. Resultados da validação. O `console` apresenta uma interface gráfica para configuração da carga e após a realização de um experimento um gráfico pode ser gerado contendo os resultados obtidos.

5. Comparação de PostgreSQL e DB2

A função de persistência dos dados é importante para sistemas de *e-commerce* e serve como meio para armazenamento de transações. Esses sistemas são tradicionalmente implementados em uma arquitetura de múltiplas camadas, sendo a escolha mais simples a de três camadas. A camada de dados, que implementa a persistência das informações, representa o principal gargalo do sistema, pois seu desempenho é geralmente orientado ao de entrada e saída (*i/o-bound*). Como principal ponto de estrangulamento, seu comportamento dinâmico influencia toda a aplicação. Nesse sentido, os experimentos descritos nesta Seção têm por objetivo identificar como o comportamento transiente de dois sistemas gerenciadores de banco de dados (SGBD) impactam no desempenho percebido pelo usuário. Os SGBDs apresentam comportamentos distintos. A comparação entre os tempos de resposta deixou evidente que o desempenho do PostgreSQL foi menor e mais previsível do que o DB2. O DB2 teve um desempenho melhor do que o PostgreSQL e, devido a mecanismos adaptativos que otimizam sua operação, seu desempenho apresentou uma comportamento dinâmico significante. As configurações do sistema são:

- Quatro `agentes`;
  - Cada agente gerecia 20 EBs inicias passando para 40 EBs (carga alta);
  - As requisições utilizadas pelos EBs são do tipo *browsing*;
  - Configurações padrão do Bench4Q para geração das requisições.
- Quatro VMs foram utilizadas na camada de aplicação;
  - O SUT fornecido pelo *benchmark* foi projetado para suportar o DB2, porém foi adicionado suporte ao PostgreSQL.
- A métrica utilizada na avaliação é o tempo de resposta em milissegundos mesurada nos clientes;
- A máquina que hospedou os banco de dados é **DB Server (master)**, conforme descrito na Tabela 1;
- Cada experimento foi replicado quatro vezes, e, nos gráficos, o intervalo de confiança de 95% é representado pela sombra em torno da média.

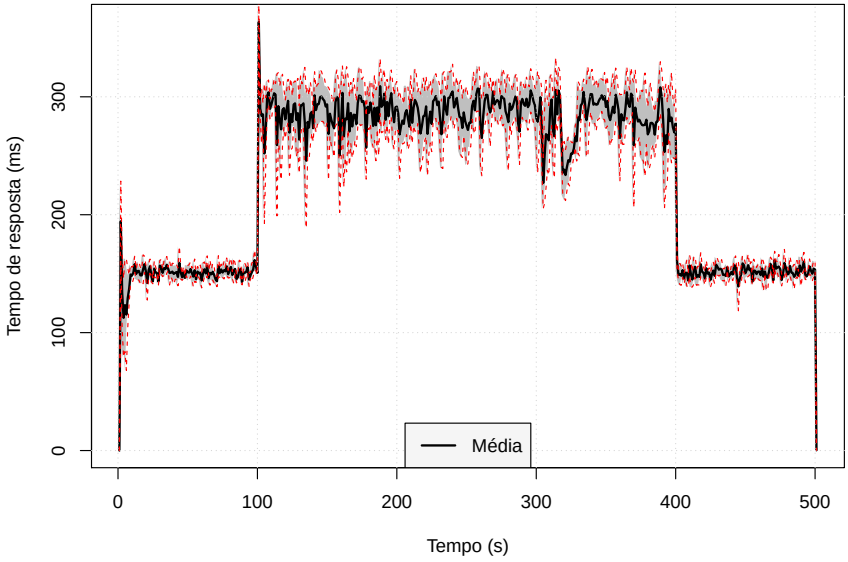


Figura 5. Tempo de resposta das requisições nos clientes utilizando o banco de dados PostgreSQL. Área cinza corresponde ao intervalo de confiança de 95%.

5.1. PostgreSQL

PostgreSQL é um conhecido banco de dados de código aberto frequentemente utilizado na computação em nuvem, e para fins de alcançar o melhor desempenho possível do banco de dados, foi utilizada a ferramenta *pgtune* [1] que permite redefinir os valores dos parâmetros definidos inicialmente para um contexto de alto desempenho. A Figura 5 apresenta o tempo de resposta de 500 s de experimento. A estratégia foi executar o *benchmark* inicialmente com a metade da quantidade de EBs disponíveis, após o segundo 100, o módulo *Demand* do *Load balancer* indica ao *Effector* que a carga deve ser dobrada. O mecanismo consiste em ativar o restante dos EBs que se encontravam em estado inativo. O resultado da mudança é uma perturbação que eleva o tempo de resposta do sistema de 150 milissegundos para um valor estacionário aproximado de 286 milissegundos.

O tempo de resposta manteve-se estacionado em 286 milissegundos aproximadamente durante o pulso. Com a execução do experimento foi verificado que a versão do PostgreSQL utilizada não apresenta um comportamento adaptativo para mudanças abruptas. No segundo 400, há a diminuição da carga para o patamar inicial do experimento. Observa-se que o desempenho é proporcional à quantidade de carga (requisições) nos dois patamares observados. Percebeu-se a ocorrência de um pico que degrada o desempenho logo após alterações bruscas, observar os instantes iniciais e por volta dos segundos 101 e 110. Muito embora, de magnitude considerável (aproximadamente 30% do patamar estacionário final), seu impacto é instantâneo, não ocasionado em maiores degradações.

5.2. IBM DB2

O DB2 é um *software* de caráter comercial, mas com uma ampla influência na área acadêmica. Vários trabalhos foram desenvolvidos visando um funcionamento adaptativo, principalmente abordagens relacionadas com a gestão automática de memória [Hellerstein et al. 2004a, Mateen et al. 2009, Storm et al. 2006].

[1] `http://pgfoundry.org/projects/pgtune`

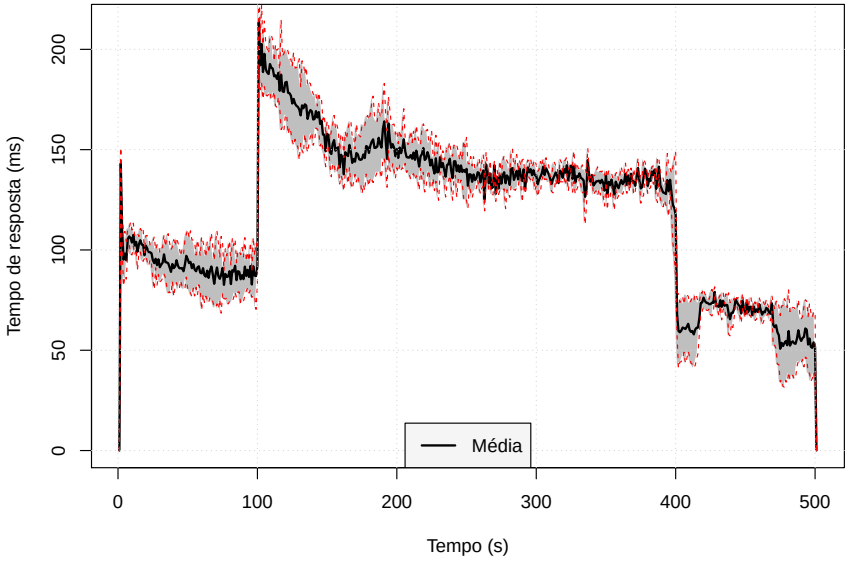


Figura 6. Tempo de resposta das requisições nos clientes utilizando o banco de dados DB2. Área cinza corresponde ao intervalo de confiança de 95%.

A Figura 6 mostra o comportamento do tempo de resposta obtido do experimento da Seção 5.1, alterando-se o SGBD para DB2. Os resultados deixam claro que há um período no qual o desempenho encontra-se em estado transiente. No início, esse período corresponde a aproximadamente 60 s. Após a perturbação, o transiente dura por volta de 140 s, dividido em duas fases: 1) uma que dura por volta de 60 s na qual o sistema experiencia uma perca de desempenho expressiva e 2) outra de 80 s em que os mecanismos internos otimizam o sistema como um todo para lidar com a carga imposta.

Na perspectiva do monitor de recursos do sistema operacional, a memória virtual ocupada pelo banco de dados aumentou de 3 GB para 11 GB após a mudança na carga de trabalho. Como a carga utilizada é do tipo *browsing*, há fortes indícios de que há *cache* de dados na memória principal. Diferentemente do PostgreSQL, não foi realizado nenhum *tuning* no banco de dados, pois a documentação explicita que o próprio *software* realiza todo o *tuning* necessário em tempo real e conforme a demanda recebida.

A carga de trabalho imposta nos intervalos (1,100) e (401, 500) são as mesmas. Assim, uma ressalva importante é que, após o degrau terminar no segundo 400, o tempo de resposta foi menor do que aquele obtido no início do experimento, mesmo que sobre condições operacionais equivalentes. Como o DB2 foi forçado a otimizar seu desempenho ao excursionar em uma região de operação mais severa, ao retornar ao patamar inicial, seu desempenho pôde ser melhor. Em outros experimentos preliminares, observamos que o DB2 mantém a característica de alterar seu desempenho durante o tempo.

Estes experimentos evidenciam a utilidade de um *benchmark* para avaliação do regime não-estacionário. Ressaltando a utilidade de um *benchmark* capaz de expor os comportamentos dinâmico de sistemas computacionais. Estes cenários não são tão particulares como aparentam, e podem ser observados comumente em ambientes de computação em nuvem, em que existem uma ampla variedade de provedores de *software* e *hardware* com restrições de desempenho diferentes.

6. Conclusões

Atualmente a maioria dos sistemas computacionais de grande porte são compostos por multicamadas, arquitetura que permite escalabilidade e amplamente adotada em aplicações web. Para estas, o planejamento de capacidade é um método que permite especificar a quantidade de recursos necessária para oferecer um determinado nível de QoS. No entanto, esse planejamento é basicamente uma decisão de longo prazo e com características estáticas. Geralmente, os recursos são determinados por métricas e parâmetros sem contar com a mudança nos patamares operacionais planejados. Desta forma, os recursos podem sofrer uma sobrecarga em situações em que há perturbação na carga de trabalho e, dependendo do caso, por levar a descumprimento dos níveis de acordo do serviço. Para este artigo, a carga de trabalho possui a característica de grande variação, fato que exige uma avaliação de desempenho focada a regime transiente.

Detalhamos os problemas e requisitos para a implementação do módulo capaz de promover flutuações na carga de trabalho do OnlineBench4Q. Foi proposto um mecanismos que permite a criação de dois grupos de clientes (EBs), para os estados ativos e inativos, em que parte desses clientes migram de um grupo para o outro, aumentando ou diminuindo a carga. Essa movimentação causa um efeito do tipo pulso na carga e serve como unidade mínima de perturbação para a construção dos mais diversos tipos de cargas de trabalho. Sua utilidade foi mostrada em um estudo de caso em que foi possível verificar o comportamento dinâmico diferente entre os banco de dados PostgreSQL e IBM DB2. Se neste artigo detalhamos o mecanismo para geração de carga variante no tempo, em [Pereira et al. 2017] realizamos a análise de resposta em frequência do OnlineBench4Q em uma configuração de porte maior. Os conceitos aqui apresentados podem ser generalizados e podem ser explorados em outros contextos, como em [Rodrigues et al. 2015].

Como trabalhos futuros, pretende-se elaborar outras cargas de trabalho como onda senoidal, especificada pelo usuário e também em função da experiência dos clientes, implementando os conceitos de abandono (*bouncing*) e convergência. No sentido da reprodutibilidade dos experimentos, pode-se criar um mecanismo que exporta as cargas de trabalho para reuso em outras ferramentas. Outro ponto a ser explorado é a avaliação do custo extra para tratar dos transientes, considerando sua efemeridade na perturbação.

Agradecimentos

Agradecemos CAPES, FAPESP, CNPq, LaSDPC/USP e IFSP pelo apoio financeiro.

Referências

Babu, S., Borisov, N., Duan, S., Herodotou, H., and Thummala, V. (2009). Automated experiment-driven management of (database) systems. In *Proceedings of the 12th Conference on Hot Topics in Operating Systems*, HotOS'09. USENIX.

Calzarossa, M. C., Massari, L., and Tessera, D. (2016). Workload characterization: A survey revisited. *ACM Comput. Surv.*, 48(3):48:1–48:43.

da Luz, H. J. F., Júnior, L. A. P., dos Santos de Souza, F. L., and Monaco, F. J. (2016). Modelagem analítica de sobrecarga transiente em sistemas computacionais por meio de parâmetros dinâmicos obtidos empiricamente. In *XIV Workshop de Computação em Clouds e Aplicações*, Salvador, BA. Sociedade Brasileira de Computação.

Feitelson, D. G. (2015). *Workload Modeling for Computer Systems Performance Evaluation*. Cambridge U. Press.

Ferreira, C. H. G., Nunes, L. H., Pereira, L. A., Nakamura, L. H. V., Estrella, J. C., and Reiff-Marganiec, S. (2016). Peesos-cloud: A workload-aware architecture for performance evaluation in service-oriented systems. In *IEEE World Congress on Services*.

Hellerstein, J., Storm, A., Surendra, M., Lightstone, S., Parekh, S., and Garcia-Arellano, C. (2004a). Incorporating cost of control into the design of a load balancing controller. *Proceedings. RTAS 2004. 10th IEEE Real-Time and Embedded Technology and Applications Symposium, 2004.*, pages 376–385.

Hellerstein, J. L., Diao, Y., Parekh, S., and Tilbury, D. M. (2004b). *Feedback Control of Computing Systems*. John Wiley & Sons, Inc., Hoboken, NJ, USA, 1 edition.

Huang, D., He, B., and Miao, C. (2014). A survey of resource management in multi-tier web applications. *IEEE Communications Surveys Tutorials*, 16(3):1574–1590.

Jain, R. (1990). *The art of computer systems performance analysis: techniques for experimental design, measurement, simulation, and modeling*. John Wiley & Sons.

Ljung, L. (1999). *System Identification: Theory for the User*. Pearson Education.

Loader.io (2017). Application load testing tools for api endpoints. `https://loader.io/`. Accessado: 31/03/2017.

Mamani, E. L. C. (2016). *Metodologia de benchmark para avaliação de desempenho não-estacionária: um estudo de caso baseado em aplicações de computação em nuvem*. PhD thesis, PPG-CCMC ICMC USP.

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.

Mateen, A., Raza, B., Hussain, T., and Awais, M. (2009). Autonomicity in Universal Database DB2. In *2009 Eighth IEEE/ACIS International Conference on Computer and Information Science*, pages 445–450. IEEE.

Menasce, D. (2002). TPC-W: a benchmark for e-commerce. *IEEE Internet Computing*.

Menascé, D. A., Almeida, V. A. F., Fonseca, R., and Mendes, M. A. (1999). A methodology for workload characterization of e-commerce sites. In *Proceedings of the 1st ACM Conference on Electronic Commerce*, EC '99, New York, NY, USA. ACM.

Menasce, D. A., Dowdy, L. W., and Almeida, V. A. F. (2004). *Performance by Design: Computer Capacity Planning By Example*. Prentice Hall, Upper S River, NJ, USA.

NewRelic (2017). Newrelic: Application performance management and monitoring. `https://newrelic.com/`. Accessado: 31/03/2017.

Pereira, L. A. (2016). *Uma abordagem baseada em resposta em frequência para modelagem e avaliação de desempenho não estacionária em sistemas computacionais*. PhD thesis, PPG-CCMC - ICMC-USP.

Pereira, L. A., Mamani, E. L. C., Santana, M. J., Santana, R. H. C., Nobile, P. N., and Monaco, F. J. (2015a). 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.

Pereira, L. A., Mamani, E. L. C., Santana, R. H. C., Santana, M. J., and Monaco, F. J. (2017). 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)*.

Pereira, Jr., L. A., Mamani, E. L. C., Santana, M. J., Santana, R. H. C., Monaco, F. J., and Nobile, P. N. (2015b). Extending discrete-event simulation frameworks for non-stationary performance evaluation: Requirements and case study. In *Proceedings of the 2015 Winter Simulation Conference*, WSC '15, Piscataway, NJ, USA. IEEE Press.

Qu, C., Calheiros, R. N., and Buyya, R. (2016). Auto-scaling web applications in clouds: A taxonomy and survey. *arXiv Computing Research Repository (CoRR)*, abs/1609.09224.

Rodrigues, R. A., Pereira, L. A., Santana, R., Santana, M., and Monaco, F. J. (2015). Benchmark para análise comportamental do sistema de memória virtual do linux. In *XIV Workshop em Desempenho de Sistemas Computacionais e de Comunicação (WPerformance)*, Recife, PE. Sociedade Brasileira de Computação.

Souza, F. L. S. (2016). Extensão da geração de carga do Bench4Q para benchmark de desempenho em regime transiente. Master's thesis, PPG-CCMC ICMC USP.

Storm, A. J., Garcia-Arellano, C., Lightstone, S. S., Diao, Y., and Surendra, M. (2006). Adaptive self-tuning memory in DB2. *Proceedings of the 32nd international conference on Very large data bases*, pages 1081–1092.

Veeraraghavan, K., Meza, J., Chou, D., Kim, W., Margulis, S., Michelson, S., Nishtala, R., Obenshain, D., Perelman, D., and Song, Y. J. (2016). Kraken: Leveraging live traffic tests to identify and resolve resource utilization bottlenecks in large scale web services. In *12th USENIX Symposium on Operating Systems Design and Implementation (OSDI 16)*, pages 635–651, GA. USENIX Association.

Yang, F. and Liu, J. (2012). Simulation-based transfer function modeling for transient analysis of general queueing systems. *European Journal of Operational Research*.

Yuan, D., Joshi, N., Jacobson, D., and Oberai, P. (2013). Scryer: Netflix's predictive auto scaling engine - part 2. `https://goo.gl/6tlHdb`. Accessado: 01/12/2016.

Zhang, W., Wang, S., Wang, W., and Zhong, H. (2011). Bench4q: A qos-oriented e-commerce benchmark. In *Computer Software and Applications Conference (COMPSAC), 2011 IEEE 35th Annual*, pages 38–47.

Zheng, W., Bianchini, R., Janakiraman, G. J., Santos, J. R., and Turner, Y. (2009). Justrunit: Experiment-based management of virtualized data centers. In *Proceedings of the 2009 Conference on USENIX Annual Technical Conference*, USENIX'09. USENIX.

Uma Arquitetura de Reputação de Confiança Aplicada ao Ambiente de Computação em Nuvem

Luís Felipe Bilecki, Adriano Fiorese

[1] Departamento de Ciência da Computação (DCC) – Centro de Ciências Tecnológicas
Universidade do Estado de Santa Catarina (UDESC) – 89219-710 – Joinville/SC

luis.bilecki@gmail.com, adriano.fiorese@udesc.br

Abstract. *Cloud computing provides computational resources to users in a scalable and dynamic way. The use of these resources are impacted by issues related to privacy, security and trust. In this sense, this paper presents a trust reputation architecture applied in the cloud computing environment. The reputed trust is based on two data sources: objective and subjective ones. In order to evaluate the proposed architecture, a scenario was developed in a P2P Network Simulator. The evaluation results show the architecture applicability with low overhead.*

Resumo. *De forma dinâmica e escalável a computação em nuvem fornece recursos computacionais para seus usuários. A utilização desses recursos é impactada por problemas relativos a privacidade, segurança e confiança. Nesse sentido, este trabalho pretende apresentar uma arquitetura de reputação de confiança aplicada a computação em nuvem. A reputação dos provedores de nuvem será composta por duas fontes de confiança: objetiva e subjetiva. Um cenário para validar a arquitetura foi desenvolvido utilizando-se um simulador de redes P2P. Os resultados apresentados demonstram a aplicabilidade da arquitetura com um overhead tolerável.*

1. Introdução

A computação em nuvem (CN), de forma dinâmica e escalável, fornece recursos computacionais (processamento, armazenamento, aplicativos) sobre a Internet aos usuários (empresas, entidades governamentais e indivíduos) [Zhang et al. 2010]. Para tanto, *data centers* são utilizados pelos provedores de serviços de nuvem. Com isso, pequenas e médias empresas, estão utilizando os recursos fornecidos pela CN (ex.: armazenamento, banco de dados, entre outros) para construir seus sistemas comerciais e disponibilizar seus próprios serviços [Tang et al. 2016].

A completa adoção de recursos e serviços disponibilizados pela CN é impactada por problemas relativos a privacidade, segurança e a confiança. Os contratos firmados entre o provedor de serviços de nuvem computacional (ou simplesmente provedor de nuvem (PN) para simplificar, como será usado no decorrer do texto) e os usuários, chamados de *Service Level Agreement* (SLA), podem auxiliar no tratamento desses problemas utilizando-se de alguns aspectos monitoráveis (métricas). Contudo, o SLA torna-se insuficiente para o estabelecimento completo da confiança entre os provedores de nuvem e os usuários, uma vez que não leva em consideração aspectos subjetivos da utilização desses serviços por parte do usuário [Noor et al. 2016].

Buscando estabelecer a confiança entre os PNs e os seus usuários, diversas abordagens podem ser utilizadas [Habib et al. 2011]. Muitos pesquisadores citam que os *feedbacks* dos usuários configuram-se como uma boa fonte para avaliar a confiança dos provedores de nuvem [Noor et al. 2016] e outros abordam a combinação desses *feedbacks* com a confiança objetiva (referente ao desempenho - Qualidade de Serviço (QoS)) [Tang et al. 2016]. Porém, tais abordagens não se preocupam em utilizar os valores históricos, relativos a interação entre os usuários e os provedores, bem como não apresentam ou apresentam parcialmente métodos para identificação de ataques ao valor das avaliações do usuário.

No cenário de computação em nuvem, a confiança pode ser entendida como um valor numérico que indica a confiabilidade dos PNs. Desta forma, tal valor pode ser utilizado como um critério para a tomada de decisão relativa a interação com tais provedores [Sabater and Sierra 2005].

O problema da confiança existente pode ser auxiliado pela aplicação de um arquitetura de reputação, que visa calcular, gerenciar e disseminar a reputação dos provedores de nuvem. A reputação é vista como uma medida agregada de um ou mais indicadores históricos a respeito das interações realizadas entre as entidades [Resnick and Zeckhauser 2002].

Desta forma, este trabalho tem por objetivo apresentar uma arquitetura de reputação de confiança, que visa auxiliar seus usuários em processos de tomada de decisão baseados em reputação. Tal arquitetura será responsável por: (i) Compor o valor de reputação baseado em dois indicadores de confiança: objetiva (indicadores de QoS dos PNs) e subjetiva (*feedback* do usuário ao PN); (ii) Fornecer uma abordagem centralizada para disseminar o conhecimento (reputação) e receber requisições dos seus usuários; (iii) Prover uma interface que realize o monitoramento dos indicadores de QoS; (iv) Receber os *feedbacks* fornecidos pelos usuários em relação aos provedores de nuvem, com a finalidade de atualizar a reputação de um PN.

O restante deste trabalho é organizado como se segue. A Seção 2 apresenta os conceitos relacionados a computação em nuvem, confiança e reputação e os trabalhos correlatos ao escopo deste trabalho. Na Seção 3, a arquitetura de reputação é apresentada. A Seção 4 apresenta o cenário dos experimentos e os resultados obtidos. Por fim, a Seção 5 apresenta a conclusão e os trabalhos futuros.

2. Revisão da Literatura

2.1. Computação em Nuvem

A Computação em Nuvem é definida como um conjunto de recursos computacionais disponíveis através da Internet e que podem ser rapidamente fornecidos sem praticamente nenhuma intervenção humana. Tais recursos são provisionados de acordo com a necessidade do usuário. Suas características principais são: acesso aos serviços feito sob-demanda, *pool* de recursos, amplo acesso à rede, elasticidade rápida e serviços mensuráveis [Mell and Grance 2011]. Os serviços são ofertados aos usuários, por meio de três modelos de negócio diferentes: SaaS (*Software* como Serviço), PaaS (Plataforma como Serviço) e IaaS (Infraestrutura como Serviço) [Zhang et al. 2010].

Para avaliar, comparar, classificar e construir um indicador de confiança, o

Cloud Services Measurement Initiative Consortium (CSMIC) desenvolveu um modelo de métricas universalmente aceito, denominado de *Service Measurement Index* (SMI) [Siegel and Perdue 2012]. O SMI é visto como um conjunto de indicadores de desempenho dos PNs, divididos em sete categorias: *accountability, agility, assurance, financial, performance, security, privacy* e *usability* [Garg et al. 2013]. Os indicadores são classificados em duas abordagens, quantitativa que refere-se a valores mensuráveis e qualitativa, baseada na experiência do usuário (*feedback*).

Neste sentido, a reputação dos PNs pode ser medida através de alguns indicadores de qualidade de serviço (QoS), presentes nas categorias mencionadas anteriormente [Tang et al. 2016]. Conforme os autores em [Garg et al. 2013] estes indicadores são definidos como:

- **Disponibilidade**: fração do tempo total em um intervalo padronizado (ex: 30 dias) que o serviço está disponível para atender as requisições;
- **Tempo de Resposta**: diferença de tempo entre a requisição do serviço e o momento que ele está disponível;
- **Segurança**: os PNs apresentam diferentes mecanismos de segurança, como, algoritmos de criptografia e gerenciamento de identidades, segurança dos dados, entre outros [Baranwal and Vidyarthi 2014]. No entanto, neste trabalho este indicador representa o nível de segurança do PN, em uma escala de 1 a 10;
- **Estabilidade**: desvio no desempenho do serviço, por exemplo, em serviços de armazenamento é a variação no tempo médio das operações de leitura e escrita, enquanto em serviços computacionais é o desvio do nível de desempenho especificado no SLA;
- **Preço**: valor cobrado pelo uso dos recursos computacionais.

Estes indicadores são usados pela arquitetura proposta e não são exclusivos de nenhum modelo de serviço. A abrangência da computação em nuvem, por meio dos modelos de serviço, é grande, possibilitando o atendimento das necessidades da grande maioria das entidades envolvidas com tecnologia da informação, por exemplo, as empresas podem utilizar os recursos fornecidos pela computação em nuvem para desenvolver e implantar sistemas comerciais.

A utilização dos serviços disponibilizados pelos provedores de nuvem é, muitas das vezes, dependente da confiança depositada no provedor. Portanto, a confiança representa um elemento chave para provedores de serviço que disponibilizam seus serviços a terceiros através da nuvem. Assim, a Seção 2.2 apresenta os conceitos de confiança, reputação e a relação desses conceitos com a nuvem e seus clientes (provedores de serviço).

2.2. Confiança e Reputação

O conceito de confiança é originário das ciências sociais que estudam o comportamento do ser humano em sociedade. A confiança é objeto de pesquisa nas mais variadas áreas, como por exemplo psicologia, sociologia, economia e computação [Firdhous et al. 2012]. Diversos conceitos de confiança são encontrados na literatura.

Segundo [Rousseau et al. 1998] a confiança é definida como um estado psicológico que compreende a intenção de aceitar a vulnerabilidade, baseada em expectativas positivas das intenções ou comportamento dos outros. Um dos conceitos mais aceitos na

definição de confiança de um modo geral, é o de Gambetta [Gambetta et al. 2000], que define a confiança como um nível particular da probabilidade com a qual um agente avalia outro agente ou grupo de agentes, que irá/irão executar uma ação particular, tanto antes dele poder monitorar tal ação (ou independentemente da sua capacidade de monitorar) e em um contexto no qual isto afeta a sua própria ação.

Não obstante a confiança apresente definições diferentes em diversos contextos, não existe um consenso entre essas definições pois, todas possuem a mesma base conceitual e se inter-relacionam. Por exemplo, em uma implementação computacional, a confiança é definida como um valor numérico que indica quão confiável é um provedor de nuvem, por exemplo [Sabater and Sierra 2005].

No ambiente de computação em nuvem, a reputação pode ser empregada para assegurar a confiança existente entre os consumidores e os provedores de nuvem. A reputação pode ser definida como uma coleção de valores agregada a respeito do comportamento passado dos participantes de uma comunidade [Resnick and Zeckhauser 2002]. Assim, a reputação, de forma geral, pode ser informada ao usuário através de duas abordagens: qualitativa e quantitativa [Mousa et al. 2015]. A qualitativa refere-se a valores categóricos, como, alta, baixa, moderada, entre outros. Na quantitativa, é representada por valores numéricos em um determinado intervalo.

No contexto deste trabalho, a reputação do provedor de nuvem é calculada como um valor referente a dois indicadores históricos de confiança: objetiva (indicadores de QoS) e subjetiva (avaliações dos usuários). Deste modo, um usuário que necessita interagir com um provedor de nuvem, utiliza a reputação como base para a tomada de decisão, com o propósito de utilizar os recursos disponibilizados. Portanto, reputar a confiança, significa disponibilizar através de uma arquitetura de reputação, a reputação dos provedores de nuvem para que os usuários utilizem esse valor para tomada de decisão.

2.3. Trabalhos Relacionados

Alguns trabalhos estabelecem metodologias para a seleção de PNs de acordo com as necessidades do usuário. [Garg et al. 2013] apresentam um *framework* para o ranqueamento e classificação dos PNs através dos indicadores de desempenho e das necessidades dos usuários. Neste *framework* o método multicritério AHP é aplicado no conjunto de dados reais que representa o desempenho dos provedores. Em [Baranwal and Vidyarthi 2014] são apresentadas métricas para avaliar PNs e também um *framework* para a seleção de PNs por meio do método de votação por ranqueamento.

Em outros trabalhos, os indicadores de QoS são aplicados na seleção e identificação de serviços confiáveis. Em [Yau and Yin 2011] são identificados os serviços que apresentam um valor de QoS satisfatório conforme os requerimentos e preferências dos usuários. Em [Tang et al. 2016] é apresentado um *framework* para a seleção de serviços de nuvem, com base na avaliação da confiança destes serviços, considerando o monitoramento de QoS e as avaliações dos usuários. Em [Nguyen et al. 2010] é apresentado um modelo de reputação baseado em redes bayesianas para a avaliação da confiabilidade de *Web Services*. Este modelo integra em um único valor de confiança, as avaliações dos usuários, monitoramento de QoS e a experiência direta do solicitante.

Além das abordagens que consideram o QoS unicamente e a combinação com outra fonte, os autores em [Noor et al. 2016] apresentaram uma plataforma responsável por

buscar serviços de nuvem, coletar avaliações dos usuários e medir a credibilidade dessas avaliações. Dessa forma, uma plataforma de recomendação de serviços de nuvem com base na sua reputação foi apresentada. [Habib et al. 2011] apresentam uma arquitetura de um sistema de gerenciamento de confiança, que avalia a confiança nos provedores de nuvem considerando outras fontes de confiança além do *feedbacks* dos usuários, como por exemplo o QoS, visando auxiliar os consumidores na escolha e identificação de provedores de nuvem confiáveis.

De um modo geral, os trabalhos relacionados utilizam o monitoramento de QoS e as avaliações dos usuários para fornecer meios de selecionar PNs com base na sua confiança, ou consideram somente as avaliações dos usuários como métrica para reputação. A arquitetura proposta calcula a reputação dos provedores de nuvem, usando duas fontes de indicadores de confiança: objetiva (histórico de QoS + QoS monitorado) e subjetiva (avaliações dos usuários com base no QoS). Além deste diferencial em relação aos trabalhos relacionados, a arquitetura de reputação proposta identifica alguns ataques ao valor de reputação e fornece métodos para o tratamento destes.

3. Arquitetura Proposta

A confiança que os usuários estabelecem nos provedores de nuvem (PN), por meio de sua reputação, apresenta-se como um importante fator para a contratação e utilização dos recursos fornecidos pela nuvem. Os sistemas tradicionais de reputação, como, eBay, consideram somente o *feedback* dos usuários para avaliar a reputação. No contexto da computação em nuvem, a reputação dos PNs deve ser baseada em outras fontes de informação [Habib et al. 2011]. Assim, de forma que os usuários possam avaliar a confiança relacionada aos PNs, pretende-se combinar as avaliações dos usuários com indicadores de qualidade de serviço (QoS) para a composição de um valor para reputação dessa confiança.

Em vista disso, a arquitetura de reputação proposta é apresentada na Figura 1. Assim, para determinar o valor de reputação dos provedores de nuvem, as requisições e os dados referentes ao PN serão processados e coletados de forma centralizada. Uma abordagem centralizada fornece um nível maior de privacidade e segurança porém, provê um ponto único de falha, que não é objeto de estudo neste trabalho [Habib et al. 2011]. Através desta arquitetura, pretende-se fornecer um mecanismo às empresas/clientes para tomada de decisão, com base no valor de reputação dos PNs.

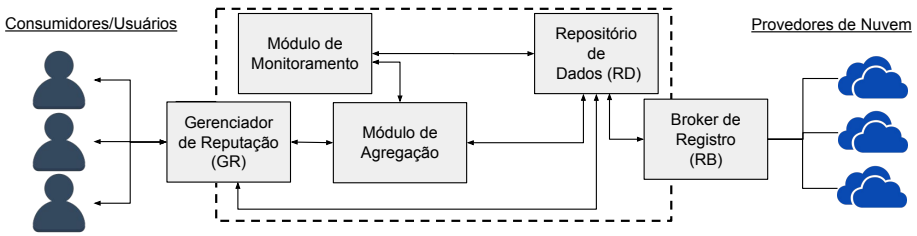


Figura 1. Arquitetura de reputação proposta

A arquitetura de reputação proposta é composta por diversos módulos e elementos, sendo eles:

- **Módulo de Monitoramento**: responsável pelo monitoramento e atualização dos indicadores de QoS do PN. Tal ação pode ser feita por intervalos fixos ou através das solicitações do usuário. O monitoramento dos indicadores de QoS representa um importante papel, pois verifica se a qualidade de serviço prometida no SLA está sendo oferecida;
- **Gerenciador de Reputação (GR)**: interface externa que realiza a comunicação com outros membros, por exemplo, um usuário deseja saber a reputação de um determinado PN ou avaliar subjetivamente um PN;
- **Módulo de Agregação**: responsável pelo cálculo da reputação por meio dos indicadores de QoS e as avaliações dos usuários. Os indicadores de QoS utilizados são: disponibilidade, tempo de resposta, estabilidade, segurança e preço [Baranwal and Vidyarthi 2014]. Este módulo calcula a reputação de serviços similares, embora estes indicadores avaliam qualquer modelo de serviço de nuvem prestado (SaaS, PaaS e IaaS), sendo que não existe um indicador que seja exclusivo para um modelo de serviço. Também, estes indicadores expressam a qualidade de serviço do provedor de nuvem de forma quantitativa.
- **Repositório de Dados (RD)**: armazena os valores históricos e atuais dos indicadores de QoS dos PNs e também as avaliações subjetivas históricas referentes aos *feedbacks* dos usuários aos PNs;
- ***Broker* de Registro (RB)**: para que os PN sejam reputados, eles devem fornecer à arquitetura de reputação as especificações dos serviços bem como os valores dos indicadores de QoS. O RB fornece essa interface.

3.1. Módulo de Agregação

O modulo de agregação utiliza os indicadores históricos de confiança objetiva e subjetiva para calcular a reputação dos provedores de nuvem. Para o cálculo, valores monitorados podem ser considerados juntamente com os valores históricos para verificar a reputação instantânea.

O indicador de confiança objetiva refere-se aos indicadores de QoS, anteriormente mencionados, que refletem no desempenho do provedor de nuvem computacional. Já o indicador de confiança subjetiva é referente às avaliações (*feedbacks*) fornecidas pelos usuários em relação a qualidade do serviço prestado pelos provedores de nuvem.

Dessa forma, a reputação de um provedor de nuvem s, representada na Equação 1, é calculada como a combinação do indicador de confiança objetiva (T_{obj}) com o indicador de confiança subjetiva histórica (T_{sub}) ponderada pelos respectivos pesos de importância (w_{obj} e w_{sub}), definidos de acordo com a preferência do usuário que requisita a reputação.

$$R_s = w_{obj} * T_{obj}(s) + w_{sub} * T_{sub}(s) \quad (1)$$

3.1.1. Indicador de Confiança Objetiva

Este indicador, representado na Equação 2, é calculado através de duas abordagens: analisando a eficiência ($Eff\ (s)$) dos provedores de nuvem baseada no histórico de QoS das interações passadas com usuários e a pontuação ($Esc\,(s)$), que representa a importância relativa desses indicadores (chamada nesse trabalho de confiança multicritério).

$$T_{obj}\,(s)\;=\;Eff\,(s)*Esc\,(s) \tag{2}$$

A eficiência dos provedores de nuvem é útil para verificar o seu histórico de QoS em interações passadas, sendo que quanto maior é a variabilidade no histórico, menor é a eficiência e consequentemente menos confiável será este provedor. Deste modo, a eficiência destes provedores pode ser calculada usando uma abordagem denominada Análise Envoltória de Dados (do inglês *Data Envelopment Analysis* - DEA). A DEA é um método não paramétrico que calcula a eficiência relativa de um conjunto de unidades, em que cada unidade é uma DMU (do inglês *Decision Making Unit*), capaz de converter entradas em saídas [Charnes et al. 1978]. Existem diversos modelos para a DEA, como o CCR (Charnes, Cooper e Rhodes) e BCC (Banker, Charnes e Cooper) [Banker et al. 1984].

Sendo assim, para a aplicação da DEA, as entradas e saídas devem ser modeladas. Neste trabalho as entradas e saídas são compostas pelos indicadores de QoS mencionados. As saídas (O_{kj}) de cada provedor de nuvem (DMU), representadas na Equação 3, são entendidas como a média dos dados históricos de cada indicador de QoS. Os dados históricos compreendem todas as interações já realizadas do PN com os seus usuários. Além disso, ao valor médio é acrescentado o desvio padrão, pois um indicador de QoS do PN, pode apresentar flutuações no conjunto de interações já realizadas.

$$O_{kj} = \overline{H_{kj}}\;+\;\sigma(H_{kj}) \tag{3}$$

As entradas (I_{ki}) são compostas pela média dos valores estimados para cada indicador (i) de QoS do PN (k). Para efetuar a geração dos valores estimados, a regressão linear é usada, através das duas primeiras interações históricas estima-se o valor da terceira, e assim por diante, até a enésima interação passada. Esta abordagem é necessária para verificar o comportamento do PN em futuras interações. A Equação 4 representa a entrada calculada para cada indicador usando o método mencionado.

$$I_{ki} = \overline{X_{ki}}\;-\;\sigma(X_{ki}) \tag{4}$$

Após definidas as entradas e saídas, a eficiência de um provedor de nuvem pode ser calculada. A eficiência é resolvida calculando um modelo da DEA por meio de programação linear. Assim, o modelo BCC da DEA, orientado à saída, é indicado para o contexto, pois os valores de saída (históricos) não dependem das entradas (estimadas).

Além da eficiência calculada pela DEA ($Eff\,(s)$), o indicador de confiança objetivo ($T_{obj}(s)$), contempla um ponderador da eficiência relativa, indicando a importância assimilada pelo provedor de serviço a cada indicador de QoS. Tal ponderador, chamado de confiança multicritério ($Esc\,(s)$), é calculado através da matriz de julgamento do método *Analytical Hierarchy Process* (AHP).

Para a determinação dos pesos de importância de cada indicador, a escala de Saaty [Saaty 1990] é usada, sendo composta por níveis de importância, de 1 a 9, em que

1 representa uma igual importância entre os indicadores e 9 representa uma enorme discrepância entre o significado dos indicadores. Através desta escala, a matriz de julgamento é criada para desempenhar a comparação pareada entre os indicadores de QoS. Desta forma, esse julgamento é realizado através de processos de normalização e cálculo de média, resultando nos pesos de cada indicador ($w1 \ldots w5$). Por fim, a $Esc\,(s)$ representada na Equação 5, é calculada pela multiplicação da média do histórico dos indicadores pelos seus pesos de importância.

$$Esc\,(s) \;=\; (w1 * \overline{D}) \;+\; (w2 * \overline{RT}) \;+\; (w3 * \overline{S}) \;+\; (w4 * \overline{E}) \;+\; (w5 * \overline{P}) \qquad (5)$$

Em que $\overline{D}$, $\overline{RT}$, $\overline{S}$, $\overline{E}$, $\overline{P}$, representam a média dos valores históricos normalizados para os indicadores de disponibilidade, tempo de resposta, segurança, estabilidade e preço, respectivamente, enquanto $w1$ até $w5$ referem-se aos pesos de importância.

3.1.2. Indicador de Confiança Subjetiva

O método usado nesse indicador é baseado nas avaliações dos usuários em relação aos provedores de nuvem, acerca dos indicadores de QoS. Adotou-se a metodologia proposta em [Noor et al. 2016], acrescentando uma abordagem ponderada no cálculo da avaliação subjetiva e o fator de identificação de ataques do tipo avaliação injusta.

Assim, o usuário envia seu *feedback*, na forma de um conjunto de avaliações, a respeito da transação realizada com o PN. O conjunto de avaliações contempla um valor de 0 a 5 para cada indicador de QoS. Portanto, a avaliação subjetiva ($Q_c(c, s)$) de um PN s fornecida pelo usuário c, calculada na Equação 6, é vista como a soma ponderada dos valores atribuídos aos indicadores de QoS na avaliação subjetiva pelos pesos de importância desses indicadores (mesma forma de cálculo adotada na confiança multicritério).

$$Q_c\,(c, s) = (ind_1 * w_1) + (ind_2 * w_2) + \ldots + (ind_n * w_n) \qquad (6)$$

O indicador de confiança subjetiva ($T_{sub}(s)$), representado pela Equação 7, é calculado como a soma ponderada de cada avaliação subjetiva ($Q_c\,(c, s)$) do usuário c ao PN s pelo fator de credibilidade ($C_f\,(c, s)$). Ainda, n representa o total de usuários que avaliaram o PN s e $|V(s)|$ é o total de avaliações subjetivas ao PN s.

$$T_{sub}(s) = \frac{\sum_{c=1}^{n} Q_c\,(c, s) * C_f\,(c, s)}{|V(s)|} \qquad (7)$$

Em qualquer cenário que utiliza o *feedback* dos usuários, o valor da confiança subjetiva está sujeito a ataques. Os ataques buscam desfigurar o comportamento real da entidade que está sendo reputada, que nesse trabalho são os provedores de nuvem. Essa manipulação visa por exemplo, promover entidades com baixa reputação para que sejam selecionados em uma futura utilização/interação. Segundo [Jøsang and Golbeck 2009] são exemplos de alguns ataques:

- Avaliações injustas: ocorre quando um usuário envia um *feedback* de confiança que não reflete a realidade do objeto avaliado, ou seja, quer promover ou prejudicar a reputação de uma determinada entidade;

- Colusão de avaliações: conjunto de múltiplos *feedbacks* (independente do seu valor) que buscam manipular a reputação de uma entidade.

Desta forma, quando os *feedbacks* são enviados à arquitetura de reputação, via GR ao final da interação, eles devem ser analisados, tratados e armazenados no repositório de dados para manutenção do histórico subjetivo. A análise é feita pelo fator de credibilidade ($C_f\,(c,s)$), entendido como a média aritmética dos fatores que identificam alguns ataques que podem ocorrer ao valor total de reputação em função do indicador de confiança subjetiva. Neste trabalho, os fatores são a densidade das avaliações ($D\,(s)$ - ataque de colusão de avaliações) e o ataque de avaliações injustas ($S_{id}(c,s)$).

$$C_f\,(c,s) = \frac{D\,(s) + S_{id}(c,s)}{2} \tag{8}$$

O fator de densidade, representado na Equação 9, modela a situação em que os usuários enviam diversas avaliações em sequência para manipular os resultados do indicador de confiança subjetiva de uma entidade. Este fator consiste na razão entre a quantidade de usuários ($M(s)$) que avaliaram um PN s, e o total de avaliações que o PN s recebeu $|V(s)|$ multiplicado pelo fator de colusão das avaliações subjetivas $L(s)$.

$$D(s) = \frac{M(s)}{|V(s)| * L(s)} \tag{9}$$

O fator de colusão das avaliações subjetivas, representado pela Equação 10, busca reduzir a credibilidade dos usuários que enviam múltiplas avaliações (independente do valor) ao mesmo PN. É calculado como a proporção entre o número de avaliações subjetivas emitidas pelos usuários $|V_c(c,s)|$, que enviaram mais avaliações do que o especificado no limite de colusão $e_v\,(s)$ sobre o total de avaliações ($|V(s)|$) relativas ao PN s.

$$L(s) = 1 + \left(\frac{1}{|V(s)|} \sum_{c=1}^{n} |V_c(c,s)|_{\;[\,|V_c(c,s)|>e_v(s)\,]} \right) \tag{10}$$

Nas avaliações injustas, os usuários com comportamento malicioso enviam várias avaliações para tentar manipular o indicador de confiança subjetiva, buscando aumentá-lo ou diminuí-lo. Este fator, apresentado na Equação 11, compreende a razão entre a quantidade de avaliações fornecidas pelo usuário c ao PN s, que estão acima ou abaixo de um limite k (ex: $k = 4.5$), e a quantidade de avaliações dadas pelo usuário c ao PN s ($|V(c,s)|$).

$$S_{id}(c,s) = 1 - \left(\frac{\sum_{z=1}^{n} |V(c,s)|_{\;[|V(c,s)|\geq k]}}{|V(c,s)|} \right) \tag{11}$$

4. Experimentos e Resultados

Para avaliar a arquitetura de reputação proposta foi construído um cenário no simulador de redes *Peer-to-Peer* (P2P) PeerFactSim.KOM. O cenário, apresentado pela Figura 2, simula a troca de mensagens entre os usuários (provedores de serviço) e a arquitetura proposta. Neste trabalho, os provedores de serviço utilizam os recursos da nuvem de forma colaborativa para fornecer um serviço aos seus clientes. Deste modo, nós de rede apresentando diferentes funcionalidades foram criados e são definidos como:

- **Nó da Arquitetura de Reputação (`ARQ`)**: Este tipo de nó implementa as funcionalidades do Gerenciador de Reputação (GR) e comunica-se com o restante dos módulos da arquitetura;
- **Nó que representa o Provedor de Serviço (`PS`)**: configurados na forma de uma *Service Overlay Network* (SON), estes nós representam os provedores de serviço que utilizam os recursos fornecidos pela nuvem para disponibilizar seus serviços. Assim, este tipo de nó interage com a arquitetura, podendo solicitar a reputação de um PN, enviar avaliações subjetivas referentes a um PN e requisitar informações de monitoramento;
- **Nó que representa um provedor de nuvem (`PN`)**: Este tipo de nó representa um provedor de nuvem computacional;
- **Nó de `Monitoramento`**: Responsável por interagir com o PN e coletar informações de QoS durante a utilização da arquitetura.

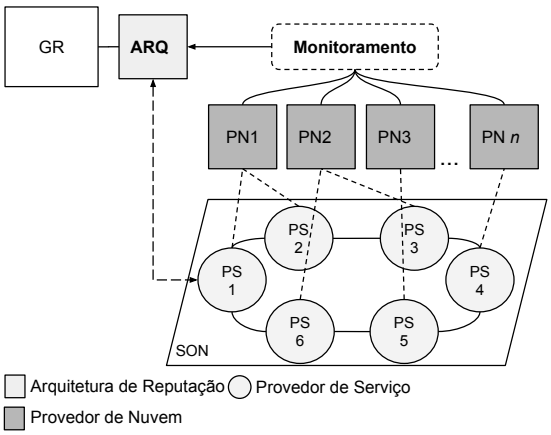


Figura 2. Cenário da Implementação

Na execução dos experimentos alguns parâmetros foram definidos como o tempo de execução da simulação (10080 min), número de PNs (10), número de PS entre 5 e 25 para verificar a escalabilidade da arquitetura e os pesos para cada indicador de QoS, D (0.3830), TR (0.2317), E (0.1861), S (0.1350) e P (0.0642), obtidos através da matriz de julgamento (Seção 3.1.1).

4.1. Avaliação da Reputação

A reputação dos provedores de nuvem é composta de dois indicadores de confiança: objetiva e subjetiva. O indicador objetivo dos PNs teve seu histórico de interações com os PSs composto através de valores aleatórios gerados seguindo uma distribuição de probabilidade linear. Os valores, apresentados na Tabela 1, estão relacionados com a média das interações anteriores de cada indicador de QoS dos PNs, em que D, RT, E, S e P referem-se aos indicadores de disponibilidade, tempo de resposta, estabilidade, segurança e preço.

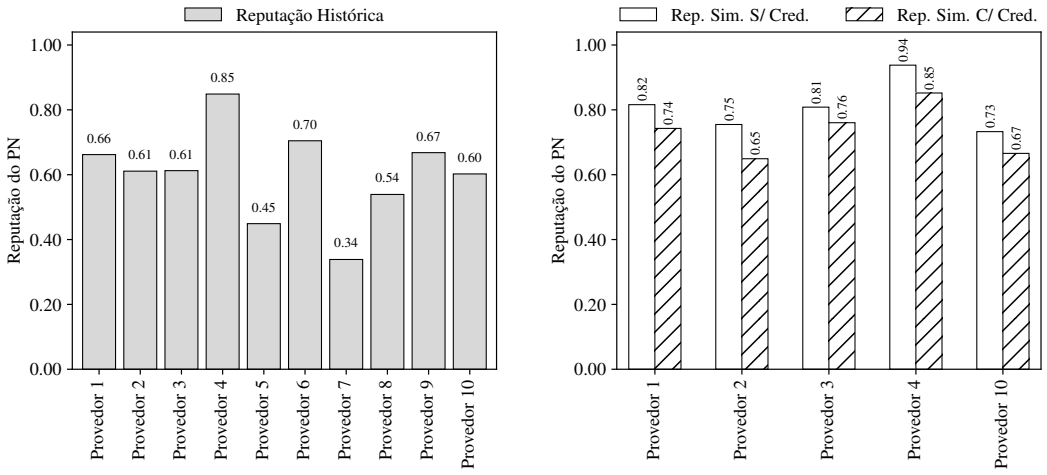
O indicador da confiança subjetiva dos provedores fictícios teve seu histórico composto por meio da base de avaliações reais presente em [Noor et al. 2016]. Essa base contém dez mil avaliações de 113 PNs. Uma etapa de pré-processamento foi realizada para extrair 10 PNs com dez avaliações subjetivas cada um, sendo que cada avaliação

Tabela 1. Valores dos indicadores de QoS

Provedor	$\overline{D}$	$\overline{RT}$	$\overline{E}$	$\overline{S}$	$\overline{P}$
Provedor 1	0,9648	99ms	62	3	$ 0,75
Provedor 2	0,9439	647ms	58	4	$ 0,65
Provedor 3	0,9188	114ms	57	7	$ 0,53
Provedor 4	0,9683	20ms	59	8	$ 0,79
Provedor 5	0,8924	535ms	64	7	$ 0,84
Provedor 6	0,5743	250ms	36	5	$ 0,32
Provedor 7	0,5604	869ms	67	7	$ 0,64
Provedor 8	0,7809	691ms	12	2	$ 0,52
Provedor 9	0,4959	332ms	33	9	$ 0,48
Provedor 10	0,6370	696ms	49	8	$ 0,64

subjetiva é composta por um conjunto de avaliações contendo 5 valores, cada qual para um dos indicadores de QoS utilizados.

Através dos históricos para cada indicador, a reputação dos PNs foi calculada. Para este cálculo, considerou-se o peso de 0.85 para o indicador objetivo e 0.15 para o indicador subjetivo, indicando que o PS atribui maior importância ao indicador objetivo. Os resultados propostos para o cálculo inicial da reputação são ilustrados na Figura 3 (a).



(a) Reputação Inicial (b) Reputação Após a Simulação

Figura 3. Análise da Reputação

Portanto, os provedores de nuvem que apresentam as melhores reputações tem uma melhor qualidade de serviço e tem recebido avaliações positivas em relação ao seu desempenho durante as interações com o usuário. É o caso, por exemplo, dos provedores de nuvem (1, 4, 6 e 9). Contudo, analisando os valores objetivos históricos, é possível notar que esses PNs apresentam melhores valores nos indicadores de QoS mais relevantes (disponibilidade e tempo de resposta), indicando que apresentam uma maior reputação em relação aos demais, ou seja, são capazes de fornecer um serviço confiável, sem muita variação e com melhor qualidade.

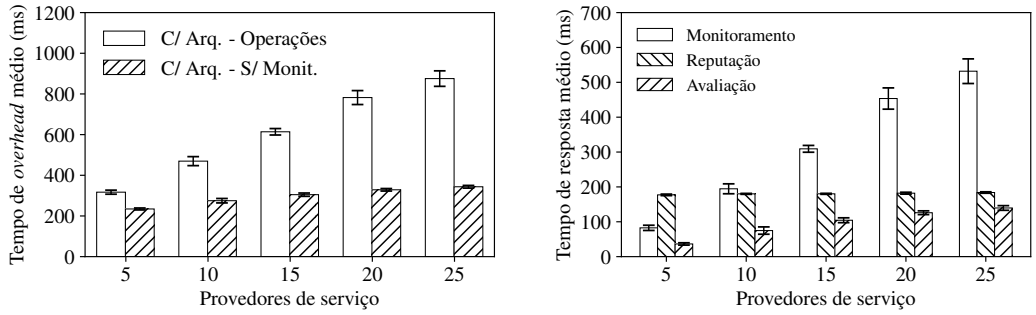
Após a apresentação dos dados históricos e a reputação inicial, uma nova interação entre alguns provedores de nuvem e seus usuários foi simulada. Para isso, construiu-se um ambiente SON com dez provedores de serviço que utilizaram os recursos de cinco

provedores de nuvem (1, 2, 3, 4 e 10). Ao final de cada interação, cada provedor de serviço (usuário) avaliou o provedor de nuvem utilizado, sendo que alguns provedores de serviço enviaram algumas avaliações maliciosas. Desse modo, os resultados são apresentados pela Figura 3 (b), em que apresenta duas opções: (i) Rep. Sim. S/ Cred: reputação atualizada sem considerar o fator de credibilidade do indicador de confiança subjetivo e (ii) Rep. Sim. C/ Cred: reputação atualizada considerando a credibilidade.

Através dos resultados da reputação simulada, pode-se notar que ao desconsiderar a credibilidade no cálculo da reputação, o valor de reputação aumenta desproporcionalmente ao comportamento real, pois um ataque de avaliações maliciosas ocorreu. Desse modo, o fator de credibilidade visa ponderar os valores das avaliações para filtrar quando ocorrem ataques.

4.2. Avaliação da Arquitetura

A avaliação da arquitetura de reputação é composta pelo *overhead* de utilização e o tempo de resposta médio para cada operação. Analisaram-se as operações de requisição de reputação (Reputação), envio de avaliações subjetivas (Avaliação - do PS ao PN) e o monitoramento dos indicadores de QoS (Monitoramento). Os resultados para o *overhead* e tempo médio são apresentados na Figura 4 considerando um intervalo de confiança de 95 %.



(a) *Overhead* de utilização (b) Tempo de resposta médio

Figura 4. Avaliação da arquitetura de reputação proposta

O *overhead* de utilização, ilustrado na Figura 4 (a), foi avaliado por meio de diversas simulações no cenário proposto, sendo que o número de provedores de serviço variou entre 5 e 25 e as requisições à arquitetura de reputação foram uniformemente distribuídas em relação ao tempo de simulação. O *overhead* médio analisado considerou as seguintes opções: (i) *C/ Arq. - Operações*: usando a arquitetura de reputação e considerando todas as operações mencionadas e (ii) *C/ Arq. - S/ Monit.*: usando a arquitetura de reputação e desconsiderando a operação de monitoramento.

Observando a opção *C/ Arq. - Operações*, pode se notar que o tempo de *overhead* aumenta em relação ao número de PSs que estão utilizando a arquitetura. Esse aumento é motivado pelas operações de monitoramento e envio de avaliações subjetivas pois, nesse caso quando o número de PSs aumenta, mais avaliações subjetivas são enviadas a respeito das transações com os PSs. Além do mais, os PNs são mais utilizados, impactando no tempo de resposta da operação de monitoramento simulada dos indicadores de QoS.

A operação de monitoramento foi desconsiderada nos experimentos realizados na opção *C/ Arq. - S/ Monit.*. O monitoramento é uma operação simulada que consiste na

troca de mensagens com dados de indicadores de QoS, entre os nós de monitoramento, PNs e o nó ARQ. Portanto, conclui-se que o monitoramento consome muito tempo.

Outras simulações foram efetuadas usando a opção *C/ Arq. - Operações* para avaliar o tempo médio de resposta de cada operação da arquitetura durante o uso pelos PSs. Os resultados presentes na Figura 4 (b), demonstram que o envio de avaliações subjetivas está fortemente associado com o número de PSs e a operação de monitoramento consome a maior fração do tempo e está relacionada com a quantidade de PSs existentes no ambiente pois, quanto maior o número de PSs maior será o uso dos recursos de nuvem, necessitando assim de mais operações de monitoramento. Ainda, neste caso, o tempo de resposta da operação de monitoramento inclui a troca de mensagens e o tempo de processamento (geração dos valores de QoS e atualização no repositório de dados), na qual a troca de mensagens consome em média 8 ms no cenário proposto.

5. Conclusão

Este trabalho apresentou uma arquitetura de reputação de confiança de provedores de computação em nuvem para auxiliar os processos de tomada de decisão. A arquitetura de reputação proposta é centralizada e composta por diversos elementos, como: módulo de monitoramento, gerenciador de reputação, módulo de agregação, repositório de dados e o *broker* de registro.

Com o propósito de avaliar a arquitetura de reputação proposta, um cenário foi desenvolvido no simulador de redes P2P PeerFactSim.KOM, contemplando os elementos da arquitetura bem como os provedores de nuvem e os usuários, entendidos neste caso, como provedores de serviço que utilizam os recursos da nuvem para disponibilizar seus serviços.

Por meio dos resultados apresentados pode-se notar que arquitetura apresenta meios para tratar ataques que ocorrem com o valor de reputação. Também observa-se que a utilização da arquitetura de reputação no ambiente de computação em nuvem, apresenta um *overhead* aceitável em relação às funcionalidades disponibilizadas e dada a importância da confiança neste contexto. Em um cenário real, o *overhead* pode ser diminuído devido a abordagem de monitoramento ser realizada por ferramentas de hardware e *software*, e não de forma simulada como foi feita na avaliação da arquitetura.

Como trabalhos futuros, pretende-se aplicar esta arquitetura em um cenário real, considerando a replicação do nó de gerenciador de reputação como uma das soluções para resolver o ponto único de falha, e analisar outros aspectos referentes ao desempenho da arquitetura. Além disto, pretende-se introduzir o conceito de bonificação ao cálculo da confiança objetiva de forma a gratificar a reputação dos provedores de nuvem que mantiverem regularidade no conjunto histórico dos indicadores de QoS.

Agradecimentos

Os autores agradecem a UDESC PROMOP pelo suporte financeiro e ao LabP2D.

Referências

Banker, R. D., Charnes, A., and Cooper, W. W. (1984). Some models for estimating technical and scale inefficiencies in data envelopment analysis. *Management Science*, 30(9):1078–1092.

Baranwal, G. and Vidyarthi, D. P. (2014). A framework for selection of best cloud service provider using ranked voting method. In *IACC 2014*, pages 831–837.

Charnes, A., Cooper, W. W., and Rhodes, E. (1978). Measuring the efficiency of decision making units. *European Journal of Operational Research*, 2(6):429–444.

Firdhous, M., Ghazali, O., and Hassan, S. (2012). Trust Management in Cloud Computing: A Critical Review. *ICTer Journal*, 4(2):24–36.

Gambetta, D. et al. (2000). Can we trust trust. *Trust: Making and breaking cooperative relations*, 13:213–237.

Garg, S. K., Versteeg, S., and Buyya, R. (2013). A framework for ranking of cloud computing services. *Future Generation Computer Systems*, 29(4):1012–1023.

Habib, S. M., Ries, S., and Muhlhauser, M. (2011). Towards a trust management system for cloud computing. In *TrustCom*, pages 933–939. IEEE.

Jøsang, A. and Golbeck, J. (2009). Challenges for robust trust and reputation systems. In *SMT 2009, Saint Malo, France*.

Mell, P. M. and Grance, T. (2011). SP 800-145. *The NIST Definition of Cloud Computing, National Institute of Standards & Technology, Gaithersburg, MD*.

Mousa, H., Mokhtar, S. B., Hasan, O., Younes, O., Hadhoud, M., and Brunie, L. (2015). Trust management and reputation systems in mobile participatory sensing applications: A survey. *Computer Networks*, 90:49–73.

Nguyen, H. T., Zhao, W., and Yang, J. (2010). A trust and reputation model based on bayesian network for web services. In *ICWS 2010*, pages 251–258. IEEE.

Noor, T. H., Sheng, Q. Z., Yao, L., Dustdar, S., and Ngu, A. H. (2016). CloudArmor: Supporting reputation-based trust management for cloud services. *IEEE TPDS*, 27(2):367–380.

Resnick, P. and Zeckhauser, R. (2002). Trust among strangers in internet transactions: Empirical analysis of ebay's reputation system. *The Economics of the Internet and E-commerce*, 11(2):23–25.

Rousseau, D. M., Sitkin, S. B., Burt, R. S., and Camerer, C. (1998). Not so different after all: A cross-discipline view of trust. *Academy of management review*, 23(3):393–404.

Saaty, T. L. (1990). How to make a decision: the analytic hierarchy process. *European Journal of Operational Research*, 48(1):9–26.

Sabater, J. and Sierra, C. (2005). Review on Computational Trust and Reputation Models. *Artificial Intelligence Review*, 24(1):33–60.

Siegel, J. and Perdue, J. (2012). Cloud services measures for global use: the service measurement index (SMI). In *SRII Global Conference (SRII), 2012 Annual*, pages 411–415. IEEE.

Tang, M., Dai, X., Liu, J., and Chen, J. (2016). Towards a trust evaluation middleware for cloud service selection. *Future Generation Computer Systems*.

Yau, S. S. and Yin, Y. (2011). Qos-based service ranking and selection for service-based systems. In *SCC 2011*, pages 56–63. IEEE.

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.

Experimentos no uso do modelo de atores para simulações numéricas distribuídas baseadas em elementos finitos

Antônio Tadeu Azevedo Gomes[1], **Franklin Zillmer**[1]

[1] Laboratório Nacional de Computação Científica (LNCC)
25.651-075 – Petrópolis – RJ – Brasil
{atagomes,fzillmer}@lncc.br

Abstract. *The numerical simulation of complex phenomena through the use of finite element methods (FEM) is widely disseminated in industry and academy. This paper is interested in a specific family of FEM called multiscale hybrid mixed (MHM) methods, given its potential applicability in cloud computing environments. To better explore this potential, however, more flexibility and reconfigurability than those provided by traditional strategies—notably those based on the MPI standard—is needed when implementing the orchestration of the numerical processes in a distributed simulator. This paper aims at presenting and assessing objectively an innovative multi-language approach, based on the actor model, to develop distributed MHM-based simulators. The experimental results presented herein show, through strong and weak scalability measures, the feasibility of this approach when compared with a traditional single-language approach based on MPI.*

Resumo. *A simulação numérica de fenômenos complexos usando métodos de elementos finitos (MEFs) é bastante disseminada na indústria e academia. O presente artigo se interessa por uma família específica de métodos denominada MEFs multiescala hibridizados mistos (MHM), dado seu potencial de aplicação em ambientes de nuvens computacionais. Para melhor explorar esse potencial, no entanto, são necessárias estratégias de implementação mais flexíveis e reconfiguráveis para a orquestração dos processos numéricos em um simulador distribuído do que aquelas usualmente empregadas em implementações de MEFs – notadamente aquelas baseadas no padrão MPI de troca de mensagens. Este artigo se propõe a apresentar e avaliar objetivamente uma abordagem multilinguagem inovadora, baseada no modelo de atores, para o desenvolvimento de simuladores distribuídos baseados no MHM. Os resultados experimentais apresentados neste artigo demonstram, por meio de medidas de escalabilidade forte e fraca, a viabilidade dessa abordagem quando comparada a uma abordagem tradicional, mono-linguagem, baseada em MPI.*

1. Introdução

A simulação numérica de fenômenos complexos, tais como o escoamento de fluidos em materiais porosos e a propagação de ondas em nanoestruturas, provê meios efetivos para resolver problemas relevantes em diversas áreas do conhecimento. Nesse contexto, o uso de métodos de elementos finitos (MEFs) é bastante disseminado na indústria e academia para simular numericamente esses fenômenos [Ciarlet 1978, Hughes 1987, Pironneau 1989]. O fundamento dos MEFs é a discretização de um domínio contínuo,

com uma certa geometria, em uma malha de subdomínios discretos, usualmente chamados de elementos. Através dessa discretização em malha, é possível obter com MEFs soluções aproximadas do sistema de equações diferenciais representando o fênomeno de interesse no domínio em questão. Em última instância, MEFs obtêm a partir desse processo de discretização um sistema linear da forma $\mathbf{Ax} = \mathbf{b}$ (sendo $\mathbf{A}$ uma matriz, $\mathbf{x}$ e $\mathbf{b}$ vetores). Tal sistema é resolvido por métodos de álgebra linear computacional, como os métodos diretos de decomposição LU, LLt e LDLt, por exemplo.

Apesar dos MEFs clássicos serem apropriados para diferentes problemas, sua acurácia pode ser comprometida seriamente quando usados em problemas multiescala – problemas que envolvem fenômenos em diferentes escalas de espaço e tempo e fortemente inter-relacionados. A qualidade da aproximação da solução nos MEFs depende da granularidade da malha; quanto mais fina a malha, melhor a aproximação, contudo o sistema linear resultante e a demanda por recursos computacionais tanto de processamento como de memória serão maiores. Independente de qual método de álgebra linear computacional seja usado, a característica altamente acoplada desses métodos impõe dificuldades na paralelização especialmente em ambiente distribuído, pois há continua troca de mensagens nos processos envolvidos. A fim de mitigar tais dificuldades computacionais novos MEFs vêm sendo desenvolvidos [Hou and Wu 1997, Efendiev et al. 2015], entre eles destacamos uma família de métodos denominada MEFs multiescala hibridizados mistos (MHM) [Harder et al. 2013, Araya et al. 2013]. Do ponto de vista matemático, os métodos MHM incorporam naturalmente múltiplas escalas (chamados de 'níveis MHM') e provêm soluções com precisão de alta ordem em malhas grossas. Do ponto de vista computacional, o nível MHM inferior é formado por um conjunto de problemas completamente independentes – ditos 'locais' – que podem ser facilmente resolvidos em paralelo. As soluções independentes desses problemas locais (que podem ser obtidas, por exemplo, por meio da resolução direta de sistemas lineares menores) alimentam o nível MHM superior, onde um sistema linear dito 'global', de complexidade muito menor (comparativamente ao que se obteria de forma equivalente usando um MEF clássico), pode ser então resolvido e cuja solução, ao ser combinada com as soluções dos problemas locais, gera a aproximação da solução para o problema em questão.

Os métodos MHM se tornam particularmente atraentes de serem aplicados em ambientes de nuvens computacionais, dado que, diferentemente dos MEFs clássicos, no MHM a atuação da tecnologia de interconexão do ambiente de execução subjacente tem um impacto muito menor no desempenho global das simulações, além de que a comunicação entre os processos ocorre somente no momento em que os processos responsáveis pelos problemas locais enviam suas soluções ao processo responsável pelo problema global. A execução dessas simulações em nuvens traz oportunidades relacionadas sobretudo à flexibilidade e dinamicidade de alocação de recursos, quando comparada a execução dessas mesmas simulações em ambientes de supercomputação. Uma oportunidade advém do fato do problema global só poder ser resolvido após os problemas locais terem terminado, o que permitiria a um simulador baseado no MHM liberar recursos computacionais associados à resolução dos problemas locais antes do término de uma simulação. Além disso, em função da maior ou menor disponibilidade de recursos computacionais (por exemplo devido a questões financeiras), os problemas locais podem ser redistribuídos durante uma simulação entre mais ou menos processadores sem nenhum impacto na implementação dos algoritmos numéricos associados ao MHM no simulador.

Por fim, em cenários de simulações de larga escala, o envolvimento de um conjunto maior de processadores aumenta a possibilidade de falhas durante a execução das simulações, de modo que a flexibilidade e dinamicidade de alocação de recursos deve também poder ser explorada com o objetivo de conferir ao simulador maior tolerância a falhas no ambiente de execução subjacente.

Contudo, o desenvolvimento de um simulador com as características de flexibilidade e dinamicidade acima demanda um ferramental apropriado de implementação. Tipicamente, simuladores baseados em MEFs são implementados sobre bibliotecas de passagem de mensagens – notadamente aquelas baseadas no padrão MPI[1] – especificamente adaptadas para uso em ambientes de supercomputação. Essas bibliotecas, pelo padrão, não são dotadas de primitivas que ofereçam facilidades de ampla reconfiguração da topologia de troca de mensagens entre os processos constituientes de um simulador. Extensões ao padrão MPI têm sido propostas para dar esse tipo de suporte, com algumas implementações como por exemplo AMPI [Huang et al. 2003]. No entanto, tais extensões ainda apresentam uma série de desafios no que se refere à sua implantação em nuvens [Acun et al. 2014].

Este artigo se propõe a apresentar uma nova abordagem, multi-linguagem, para o desenvolvimento de simuladores baseados em MEFs que possam executar de forma eficiente e flexível em nuvens computacionais, tendo como foco inicial os métodos MHM. Como objetivo específico, este artigo explora o uso do modelo de atores [Agha 1985] presente na linguagem Erlang para desenvolver um simulador MHM que seja flexível e dinâmico no que se refere a alocação de recursos em nuvens. A despeito de suas qualidades como linguagem para desenvolvimento de sistemas distribuídos altamente escaláveis e tolerantes a falhas, Erlang não foi projetada para computação numérica e são de fato poucas as instâncias de uso de Erlang para esse tipo de cenário documentadas na literatura [Scalas et al. 2008]. Porém, sua qualidade de integração permite que módulos Erlang possam se comunicar com relativa facilidade com módulos de *software* desenvolvidos em outras linguagens. Essa estratégia é explorada no presente trabalho, sendo C++ a linguagem escolhida para a implementação dos algoritmos numéricos associados ao MHM.

Uma série de experimentos foram conduzidos e seus resultados são apresentados neste artigo com relação ao *overhead* decorrente do uso de Erlang na implementação do simulador MHM, quando executado sobre uma tecnologia de interconexão Gigabit Ethernet (típica de nuvens), em comparação com uma implementação tradicional, baseada em MPI, quando executada sobre uma tecnologia de interconexão Infiniband. Esses resultados demonstram que, conforme esperado, a execução baseada em MPI sobre Infiniband é mais rápida, mas com uma vantagem pouco substancial em relação a execução baseada em Erlang sobre Gigabit Ethernet, o que sinaliza a viabilidade do uso do modelo de atores combinado à execução em nuvens para resolver problemas complexos em áreas onde tradicionalmente os ambientes de supercomputação são predominantes.

O restante deste artigo está estruturado como se segue. Na Seção 2 a linguagem Erlang e o modelo de atores são apresentados. A Seção 3 faz uma apresentação sucinta dos principais aspectos computacionais relacionados à família de métodos MHM. A Seção 4 descreve a arquitetura do simulador MHM baseado no modelo de atores. A

[1] `http://mpi-forum.org`

Seção 5 apresenta os resultados experimentais que aferem a escalabilidade do simulador MHM baseado no modelo de atores comparativamente a uma implementação baseada em MPI. Finalmente, a Seção 6 resume as contribuições obtidas e potenciais trabalhos futuros.

2. Erlang e o modelo de atores

A forma tradicional de se implementar concorrência em uma linguagem de programação é por meio de entidades chamadas *threads*, que podem ser vistas como comportamentos (fluxos de execução de instruções) operando sobre regiões compartilhadas de memória. O compartilhamento de memória entre *threads* e a consequente possibilidade de condições de corrida leva a necessidade de uso de semáforos, que por sua vez podem levar a outros problemas relacionados a concorrência, como *deadlocks*.

O modelo de atores [Agha 1985] propõe uma abordagem distinta para atacar problemas resultantes de concorrência. Nesse modelo, entidades chamadas atores também encapsulam comportamentos, mas estes não compartilham memória. Cada ator tem uma fila de mensagens que é usada para trocar mensagens com outros atores. Ao receber uma mensagem, um comportamento é disparado no ator, podendo ocasionar: (i) o envio de mensagens a outros atores; (ii) a criação de novos atores; ou (iii) a mudança no comportamento do ator para próximas mensagens a serem recebidas. A comunicação entre atores é assíncrona e não se pressupõe no modelo a ordenação ou mesmo garantia de entrega de mensagens. Disso decorre que sistemas baseados em atores precisam saber lidar explicitamente com reconfigurações e falhas, que são duas das principais características da linguagem Erlang.

Atores são denominados 'processos' em Erlang. Essa linguagem oferece recursos para criação e gerenciamento desses processos com o objetivo de simplificar a programação concorrente. Os processos Erlang geralmente seguem um padrão comum: uma vez instanciados eles chamam uma função recursiva para processar e produzir dados. A função recursiva geralmente segue a seguinte sequência de operações: (i) receber uma mensagem de um processo, (ii) tratar a mensagem, (iii) atualizar o estado do processo, e (iv) passar o estado atualizado para uma nova chamada da função através de sua chamada recursiva. Finalmente, uma mensagem de parada especial é geralmente definida para permitir que o processo receptor possa ser encerrado normalmente.

Erlang é capaz de compilar em um formato intermediário (*bytecodes*) e pode ser executada em máquinas virtuais (nós Erlang). Normalmente, cada nó Erlang é executado em um *host* diferente (p.ex. uma máquina física em um *cluster* ou uma máquina virtual em uma nuvem).

Erlang é uma linguagem com estrutura funcional, projetada para facilitar a implementação de *software* distribuído devido a existência de bibliotecas com estruturas configuráveis, incluindo um sistema de gerenciamento de banco de dados embarcado eficiente e tolerante a falhas, particularmente útil para implementação de estratégias de *checkpointing* de aplicações. Outras bibliotecas possuem mecanismos de manipulação de erros e de monitoramento de exceções sendo conhecidas como OTP (*Open Telecom Platform*) e oferecem uma série de comportamentos (*behaviors*) genéricos, tais como servidores (`gen_server`), máquinas de estados finitos (`gen_fsm`), manipuladores de eventos entre outros.

Um desses comportamentos, denominado supervisor (`gen_supervisor`), tem como principal tarefa gerenciar, monitorar e, eventualmente, reiniciar processos em caso de falha. Os supervisores podem gerenciar qualquer tipo de processo em Erlang, incluindo outros supervisores. Esta organização cria uma estrutura hierárquica chamada árvore de supervisão. Finalmente, uma aplicação Erlang é um tipo especial de comportamento que permite a toda uma árvore de supervisão ser iniciada e interrompida como uma unidade. Para fornecer tolerância a falhas de *hardware*, uma aplicação Erlang pode ser controlada de maneira distribuída, através do emprego de dois ou mais nós Erlang executados em *hosts* diferentes. Este modo de organização configura uma aplicação distribuída. Se o nó Erlang onde uma aplicação distribuída executa falhar, por exemplo, devido a uma falha de rede ou de máquina, a aplicação pode ser reiniciada em outro nó Erlang.

Erlang oferece ainda um sistema de gerenciamento de banco de dados (SGBD) embarcado chamado *Mnesia*. Dentre as características do *Mnesia*, a tolerância a falhas e a possibilidade de manutenção dos dados geridos pelo SGBD em memória (para fins de desempenho) são duas das mais importantes, sendo largamente empregadas neste trabalho. Conforme [Mattsson et al. 1999], *Mnesia* tem a capacidade de replicar uma tabela entre vários nós Erlang. Todas as réplicas são iguais e não há nenhum conceito de cópia primária e de salvaguarda. Se uma tabela é replicada todas as operações de escrita são aplicadas a todas as réplicas para cada transação. Se houver falha em alguma das réplicas as operações de escrita ainda serão bem-sucedidas e os dados destas réplicas serão posteriormente atualizados.

3. A família de métodos numéricos MHM

Os métodos MHM são adaptados à resolução de modelos com múltiplas escalas ou com grandes contrastes, caracterizando-se por alta ordem de precisão (baixas taxas de erro) e por incorporar a granularidade típica de ambientes de execução massivamente paralelos. Para tanto, dada uma malha (grossa) $\mathcal{T}_H$ que discretiza o domínio em um conjunto de elementos $\{K_t\}_{t\in T}$, com $T \subset Z^+ = \{0, \ldots, N_t - 1\}$, o método MHM é composto de uma formulação global definida no esqueleto $\mathcal{E}_H$ da malha (que também é discretizada em um conjunto de faces $\{F_e\}_{e\in E}$, com $E \subset Z^+ = \{0, \ldots, N_e - 1\}$), e de uma coleção de problemas locais definidos para cada elemento K_t da malha, problemas estes dirigidos pelos dados do problema original e por um processo chamado de hibridização, que relaxa a continuidade da solução no esqueleto da malha. A Figura 1 ilustra esses conceitos de discretização da malha $\mathcal{T}_H$ e de seu esqueleto $\mathcal{E}_H$ no $\mathbb{R}^2$.

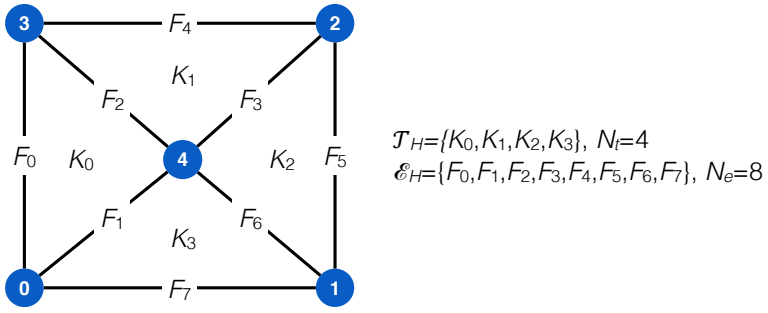


Figura 1. Exemplo de malha no $\mathbb{R}^2$.

A aproximação de solução entregue pelos métodos MHM tem a forma usual

$$u_H := u_0 + \sum_{K_t \in \mathcal{T}_H} \Big[\sum_{i=0}^{N_I - 1} k_i \eta_t^i + \eta_t^f \Big], \tag{1}$$

onde u_H é a solução aproximada do problema (o índice H refere-se ao nível de refinamento da malha grossa), u_0 e k_i são computados no problema global, e η_t^i e η_t^f são computados nos problemas locais. N_I é diretamente proporcional ao número de faces de um elemento da malha. Por questões de espaço e também de enfoque, outros detalhes da formulação matemática dos métodos MHM são omitidos neste artigo. O leitor interessado pode se remeter as referências [Harder et al. 2013] e [Araya et al. 2013]. Para um melhor entendimento dos aspectos computacionais desses métodos, é apresentado na Listagem 1 o pseudocódigo para o algoritmo numérico de um simulador baseado nos métodos MHM.

Listagem 1 Pseudocódigo para o algoritmo numérico de um simulador MHM.

```
Require: T_H = {K_t}_{t∈T}                          ▷ Malha com elementos indexados
  P_{T_H} ← SPLITPROBLEM(T_H)                       ▷ P_{T_H} := {(K_t, {F_t})}_{t∈T}
  S_{T_H} ← ∅
  for all p_t ∈ P_{T_H} do                          ▷ p_t := (K_t, {F_t})
      s_t ← SOLVELOCALPROBLEM(p_t)                  ▷ s_t := (η_t^f, {η_t^i}_{i∈{0,...,N_I−1}})
      S_{T_H} ← S_{T_H} ∪ s_t
  end for
  (u_0, k_i) ← REDUCELOCALPROBLEMS(S_{T_H})
  u_H ← COMPUTESOLUTION(u_0, k_i, S_{T_H})
```

Um simulador MHM é construído sobre um conjunto de primitivas que são diretamente mapeadas nos blocos matemáticos principais dos métodos MHM: SPLITPROBLEM, SOLVELOCALPROBLEM, REDUCELOCALPROBLEMS, e COMPUTESOLUTION.

A primitiva SPLITPROBLEM divide o problema original definido sobre uma malha (grossa) $\mathcal{T}_H$ em um problema global (o primeiro nível MHM) e um conjunto de problemas locais (o segundo nível MHM), um para cada elemento K_t da malha. Essa primitiva retorna um conjunto $P_{\mathcal{T}_H}$ de pares $(K_t, \{F_t\})$, onde $\{F_t\}$ representa o conjunto de todas as faces de K_t. (Todas as faces de todos os elementos, em conjunto, formam o esqueleto $\mathcal{E}_H$ da malha.) A primitiva SOLVELOCALPROBLEM computa as contribuições η_t^f e $\{\eta_t^i\}_{i\in\{0,\ldots,N_I-1\}}$ de um elemento específico K_t na malha, a serem usadas na montagem do problema global. A primitiva REDUCELOCALPROBLEMS combina as contribuições η_t^f e $\{\eta_t^i\}_{i\in\{0,\ldots,N_I-1\}}$ dos problemas locais para computar u_0 e k_i na Equação (1). Finalmente, na primitiva COMPUTESOLUTION, a aproximação u_H definida pela Equação (1) é computada.

4. Arquitetura do simulador MHM baseado no modelo de atores

A Figura 2 mostra como foi estruturada e criada a arquitetura do simulador MHM distribuído baseado no modelo de atores. Em nossa implementação empregou-se a arquitetura mestre-escravo referente ao modelo de comunicação adotado para os atores Erlang. Isso não limita, a princípio, o uso de estilos arquiteturais diversos induzidos pelos métodos numéricos. No caso do MHM, por exemplo, o estilo em questão é o *fan-out/fan-in*.

O processo `main_supervisor` implementa o comportamento `supervisor`. Esse processo gerencia os processos `main_server`, `regis_server`, e `fsm_supervisor`, sendo todos disparados na inicialização da aplicação. Todos esses processos executam em um mesmo nó Erlang; se o nó se torna indisponível, esses processos são reiniciados em outro nó Erlang com seus respectivos estados preservados graças ao uso do *Mnesia*. Esta é a primeira característica que permite ao simulador MHM ter suporte de tolerância a falhas.

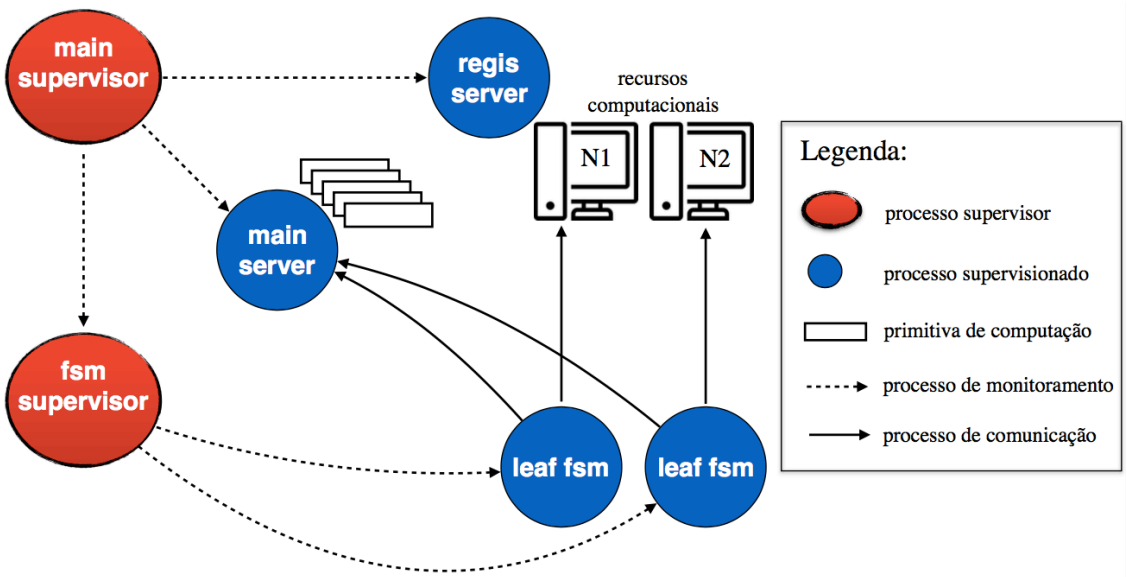


Figura 2. Arquitetura do simulador

O processo `main_server` mantém um conjunto de tarefas computacionais, que são dinamicamente alocadas nos recursos computacionais disponíveis. Cada tarefa é relacionada a um problema local ou global do MHM, e tem um estado associado – "Not Solved", "Being Solved", "Being Reduced" (para problemas globais somente) ou "Solved". Além disso, cada tarefa é mapeada em uma instância de uma das primitivas descritas na Seção 3. Essas primitivas são implementadas em C++ e executam em recursos computacionais geridos pelo `regis_server`, que implementa o comportamento `gen_server`. O `regis_server` coleta informações de status dos recursos computacionais no ambiente de execução subjacente (como máquinas virtuais em uma nuvem) e, para cada recurso, solicita ao `fsm_supervisor` (outra implementação do comportamento `supervisor`) a criação/destruição de processos 'escravos' associados ao recurso toda vez que o mesmo se torna disponível/indisponível. Esta é a segunda característica que permite ao simulador MHM ter suporte de tolerância a falhas.

Os processos escravos (`leaf_fsm`) implementam o comportamento `gen_fsm`. Eles executam no mesmo nó Erlang que os demais processos da aplicação Erlang. Uma vez iniciado, um `leaf_fsm` continuamente solicita ao `main_server` novas tarefas computacionais. O `main_server` responde à requisição do `leaf_fsm`: (i) selecionando uma tarefa apropriada, (ii) mudando o estado da tarefa para "Being Solved"/"Being Reduced", e (iii) respondendo ao `leaf_fsm` com a primitiva MHM a ser executada. O `leaf_fsm` despacha então a execução da primitiva no recurso computacional a ele associado, e monitora essa execução. Quando o `leaf_fsm` detecta que a primitiva terminou,

ele envia os resultados da primitiva para o `main_server`, que pode mudar o estado da tarefa ou criar novas tarefas. Se o `leaf_fsm` detecta que a primitiva não completou por causa de uma falha no recurso computacional ou de uma partição de rede, ele notifica o `main_server` e o `regis_server`, que podem então realocar dinamicamente a tarefa para um recurso diferente.

A Figura 3 ilustra como o estado das tarefas evolui no simulador. O simulador inicia com uma única tarefa representando o problema global – $\mathcal{T}_H$ na Listagem 1 – sendo publicada no `main_server` com o estado "Not Solved". Nesse ponto, um único `leaf_fsm` será capaz de pegar essa tarefa, o que dispara o despacho da primitiva SPLITPROBLEM no recurso computacional associado a esse `leaf_fsm` e a mudança de seu estado para "Being Solved". O resultado dessa primitiva – $P_{\mathcal{T}_H}$ na Listagem 1 – é tratado pelo `leaf_fsm` como um conjunto de requisições – uma para cada $p_t \in P_{\mathcal{T}_H}$ – para a criação de novas tarefas no `main_server`. Cada uma dessas novas tarefas é relacionada a um problema local diferente e adicionada ao `main_server` com o estado "Not Solved". Para simplificar a ilustração da evolução desses estados, na Figura 3 só são apresentados dois problemas locais derivados da primitiva SPLITPROBLEM.

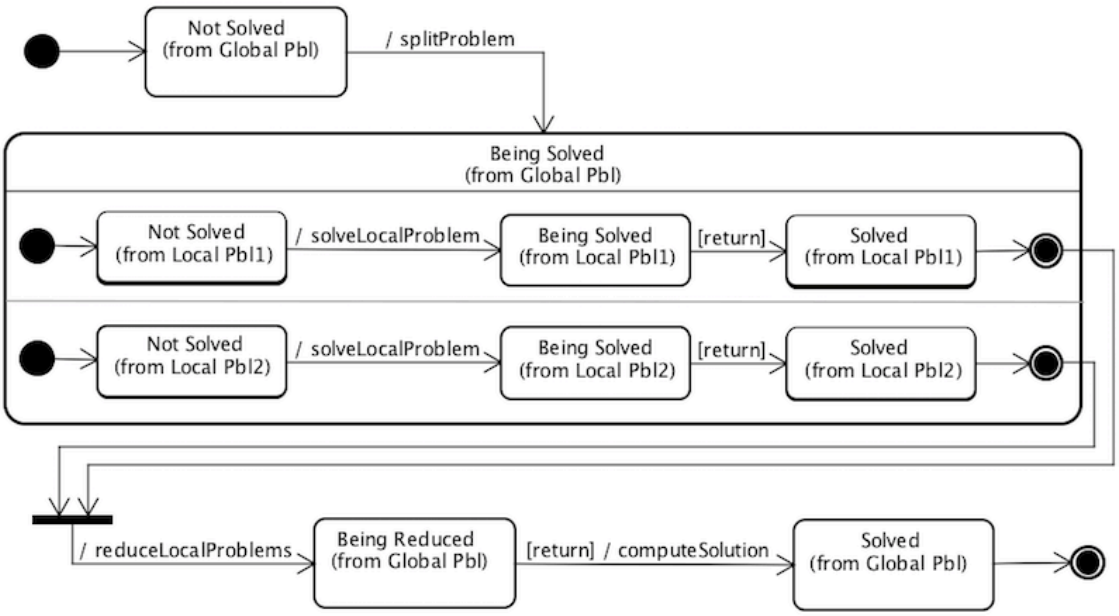


Figura 3. Evolução das tarefas.

A partir desse ponto cada `leaf_fsm` passa a obter diferentes tarefas relacionadas aos problemas locais, mudando seus estados para "Being Solved" e despachando a execução da primitiva SOLVELOCALPROBLEM em seu recurso computacional correspondente. Para computar η_t^f e $\{\eta_t^i\}_{i\in\{0,\ldots,N_I-1\}}$ em cada problema local, é usado um algoritmo sequencial de decomposição LU oferecido pela biblioteca Eigen,[2] de modo que o recurso computacional associado a um `leaf_fsm` possa resolver em paralelo tantos problemas locais quanto a quantidade de *cores* disponíveis nesse recurso. Quando essas invocações da primitiva SOLVELOCALPROBLEM são completadas, cada `leaf_fsm` envia seu conjunto de resultados – $s_t \in S_{\mathcal{T}_H}$ na Listagem 1 – para o `main_server`, que modifica o

[2] `http://eigen.tuxfamily.org`

estado das tarefas correspondentes para “Solved”. Cada `leaf_fsm` então prossegue sua execução obtendo outras tarefas e as despachando para seu recurso até que não haja mais tarefas no estado “Not Solved”.

Quando todos os problemas locais estão no estado “Solved”, a próxima requisição de um `leaf_fsm` é respondida pelo `main_server` com um primitiva REDUCELOCALPROBLEMS, e o estado associado ao problema global é modificado de “Being Solved” para “Being Reduced”. Novamente, é empregada a decomposição LU nesta primitiva para resolver o sistema linear associado ao problema global. No caso da primitiva REDUCELOCALPROBLEMS, no entanto, é usada uma versão paralela, de memória compartilhada, oferecida pela suíte Pardiso,[3] uma vez que há um único sistema linear a ser resolvido no problema global. O resultado obtido a partir da primitiva REDUCELOCALPROBLEMS dispara a execução imediata da primitiva COMPUTESOLUTION e a mudança do estado da tarefa associada ao problema global para “Solved” no `main_server`. Após isso, a simulação é encerrada. A Figura 3 não contempla estados excepcionais pois estes são implicitamente tratados pelo mecanismo de tolerância a falhas de Erlang, inclusive o caso de parada total.

5. Experimentos

Para avaliar de forma comparativa o uso de Erlang na implementação do simulador MHM, foi implementada uma segunda versão do simulador MHM, chamada neste artigo de ‘implementação de referência’, empregando somente C++ e o padrão MPI. A Figura 4 ilustra os processos principais que compõem a implementação de referência. Nessa implementação, os R processos MPI que a compõem são onipotentes, em contraste com a versão Erlang, que diferencia mestre e escravos. Cada processo MPI $r \in R$ é responsável por executar parcialmente a primitiva SPLITPROBLEM sobre uma partição reduzida $\mathcal{T}_H^r$ de $\mathcal{T}_H$. Esse particionamento configura um escalonamento estático dos processos MPI no que se refere à solução dos problemas locais pela primitiva SOLVELOCALPROBLEM, escalonamento este que é obrigatoriamente mantido até o fim da simulação. Como observado na Figura 4, a implementação de referência é estruturalmente mais simples do que a versão Erlang, o que se deve sobretudo à ausência de um esquema de escalonamento dinâmico dos processos bem como de quaisquer mecanismos de tolerância a falhas.

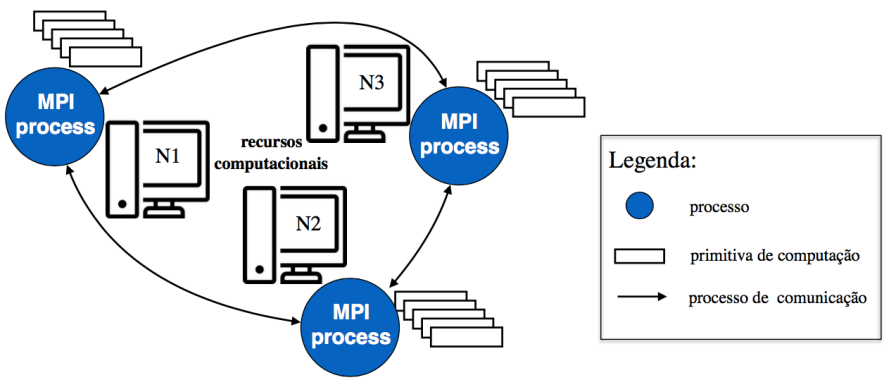


Figura 4. Arquitetura da implementação de referência.

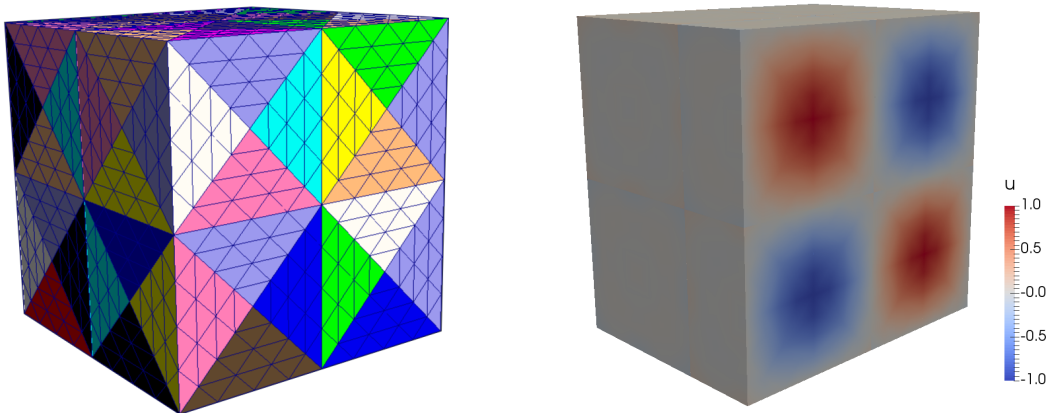
[3] `http://www.pardiso-project.org`

Para este artigo foram coletadas medidas de desempenho relacionadas a escalabilidade forte e fraca do simulador MHM baseado em Erlang e da implementação de referência com MPI. Para essas medidas, as duas versões foram executadas no supercomputador Santos Dumont (SDumont) do LNCC, que possui uma arquitetura de *cluster* com as seguintes configurações por nó de processamento: 2x CPU Intel Xeon E5-2695v2 Ivy Bridge (12 *cores* por CPU), 2.4GHZ, 64GB DDR3 RAM. Os nós do *cluster* são interconectados por meio de uma rede Infiniband FDR, com topologia *fat-tree*, e compartilham um sistema de arquivos distribuídos baseado no Lustre v2.1. As razões pelas quais foi utilizado o SDumont ao invés de uma nuvem para as simulações com a versão Erlang do simulador incluem a disponibilidade de um número maior de *cores* que viabilizassem as medidas de escalabilidade e o uso de um mesmo hardware e de uma mesma suíte de *software* para compilação e execução das duas versões do simulador. Para imitar os efeitos do uso de uma rede de interconexão mais típica de nuvens no caso da versão Erlang, para as simulações com esta versão do simulador foi utilizada a rede de interconexão 10 Gigabit Ethernet do SDumont, que é usualmente empregada para procedimentos de acesso remoto e de administração do *cluster*.

O código C++ foi compilado com o compilador Intel icpc versão 16.0.2 *build* 20160204, com as *flags* de otimização '$-O2$' e '$-ipo$'. Foram usadas a biblioteca Eigen versão 3.3-rc1 e a implementação do resolvedor de sistemas lineares Pardiso presente na biblioteca MKL da suíte Intel's Parallel Studio XE 2016.2.062. O código Erlang foi executado com o sistema aberto Erlang/OTP da Ericsson, versão 17 erts-6.1, com a compilação em código nativo HiPE (*High Performance Erlang*) habilitada.

Para as simulações, foi usado um modelo de fluxo difusivo (equação de Darcy: $\kappa \nabla u(x, y, z) = f(x, y, z)$) como problema e um cubo unitário como domínio do problema, a ser discretizado em uma malha grossa. O coeficiente de difusão na equação foi setado como $\kappa = 1$ e seu termo de fonte como $f(x, y, z) = 12\pi^2 \sin(2\pi x) \sin(2\pi y) \sin(2\pi z)$, e as condições de fronteira do domínio foram estabelecidas como $u = 0$ (Dirichlet homogêneo). A solução exata para esse problema é $u(x, y, z) = \sin(2\pi x) \sin(2\pi y) \sin(2\pi z)$. As simulações foram setadas com 24, 192 e 1.536 elementos na malha de tetraedros que discretiza o domínio. Em cada um desses elementos uma submalha com 59 ou 6.046 tetraedros é gerada em tempo de execução como parte das operações efetuadas pela primitiva SOLVELOCALPROBLEM. A malha de 192 tetraedros com 59 tetraedros por submalha é ilustrada na Figura 5(a), e a aproximação numérica correspondente ao modelo escolhido sobre esta mesma malha é ilustrada na Figura 5(b).

A configuração descrita acima gera 6 parametrizações diferentes para o simulador MHM, mas só foram experimentadas 5 delas para os propósitos deste artigo, uma vez que a configuração com 1.536×59 tetraedros gera problemas pequenos e sujeitos a variações estatísticas importantes devido ao ambiente de execução. As 5 parametrizações apresentadas foram testadas em 3 situações de disponibilidade de recursos distintas: 24, 192 e 768 *cores*. O valor de 24 *cores* corresponde ao uso de um único nó do SDumont, sendo usado como *baseline* de comparação. O valor de 192 *cores* equivale a 8 nós do SDumont, e esse número de *cores* foi utilizado para mostrar a escalabilidade fraca comparando as malhas grossas de 1.536 e 12.288 tetraedros, com mesmo nível de refinamento nas malhas locais. O valor de 768 *cores* corresponde ao uso de 32 nós do SDumont, número máximo



(a) Malha grossa com 192 elementos, cada um deles de uma cor diferente e incluindo suas respectivas submalhas de 59 elementos cada, num total de 724.992 elementos.

(b) Isolinhas da aproximação numérica obtida com MHM para $x = 0{,}75$. Note que nas fronteiras do domínio o valor da aproximação para u é zero, como estabelecido pela condição de fronteira de Dirichlet homogêneo.

Figura 5. Exemplo de modelo usado nas simulações.

possível de se alocar neste *cluster* no momento dos experimentos.

Para cada uma das 15 configurações, foram feitas 3 rodadas de simulação e escolhida, para cada configuração, a rodada com o menor tempo total de execução para apresentação neste artigo. (É importante destacar que não houve variação estatística significativa entre rodadas de uma mesma configuração.) No caso da situação de disponibilidade de 24 *cores*, como a simulação ocupa somente um nó do SDumont, foi usada uma versão mais simples do simulador MHM que usa somente *multithreading*, sem nenhum envolvimento de Erlang ou MPI nas execuções. O propósito dessa simplificação foi capturar melhor o impacto da distribuição dos processos nas versões Erlang e MPI do simulador MHM.

A Tabela 1 consolida todas as medidas de escalabilidade obtidas nos experimentos. Analisando a escalabilidade forte – isto é, o quão eficiente é o simulador em ter reduzido seu tempo de execução com um aumento na quantidade de recursos disponíveis para o mesmo –, pode-se observar na tabela que a eficiência de ambos os simuladores melhora à medida que os problemas ficam maiores. Certamente são necessárias medidas de desempenho em cenários com maior quantidade de recursos disponíveis para confirmar essa tendência, mas de toda forma tratam-se de resultados promissores: à medida que os problemas crescem em tamanho, a razão entre processamento e comunicação cresce consideravelmente nos dois simuladores em função das características de baixo acoplamento entre as fases local e global dos métodos MHM. Outra observação é que a versão Erlang do simulador é menos eficiente do que a versão MPI – o que era esperado –, mas por uma margem pequena. Esse é um outro resultado interessante porque demonstra a viabilidade de se executar simulações com os métodos MHM em ambientes de nuvem e, de forma mais geral, de se utilizar Erlang como linguagem de coordenação para aplicações científicas com necessidades de tolerância a falhas e estruturas de execução semelhantes à induzida pelos métodos MHM.

Tabela 1. Medidas de escalabilidade.

		Baseline	MPI		Erlang	
Número de tetraedros		24 *cores*	192 *cores*	768 *cores*	192 *cores*	768 *cores*
$12.288 \times 59 = 724.992$						
	Tempo (seg)	$\mathbf{129{,}81}^*$	57,74	72,88	97,00	97,00
	Eficiência	100,00%	28,10%	5,57%	16,62%	4,16%
$1.536 \times 6.046 = 9.286.656$						
	Tempo (seg)	489,00	$\mathbf{82{,}57}^*$	51,72	$\mathbf{108{,}00}^*$	65,00
	Eficiência	100,00%	74,03%	29,54%	56,60%	23,51%
$12.288 \times 6.046 = 74.293.248$						
	Tempo (seg)	4.753,00	641,61	$\mathbf{220{,}20}^*$	667,00	$\mathbf{240{,}00}^*$
	Eficiência	100,00%	92,60%	67,45%	89,70%	61,89%

As entradas na Tabela 1 que estão em negrito e demarcadas com um '$*$' também podem ser usadas para uma análise das duas versões do simulador MHM no que se refere à sua escalabilidade fraca – isto é, o quão eficiente é o simulador em manter o seu tempo de execução constante à medida que o tamanho dos problemas e a capacidade computacional disponível para resolvê-los aumentam numa mesma razão. Entre a primeira e terceira configurações, usando a versão Erlang, houve um aumento no tempo de execução por um fator de $\approx 1{,}84$. Para a versão MPI, esse aumento foi de um fator de $\approx 1{,}69$. Esses fatores servem de base para medir a eficiência de escalabilidade fraca dos simuladores, tomando o número total de tetraedros em cada configuração como tamanho do problema:[4] da primeira para a terceira configuração há um aumento em $\approx 102{,}47$ vezes o número de tetraedros, contra um aumento em apenas 32 vezes o número de *cores* utilizados. Percebe-se, portanto, que as duas versões de simuladores MHM escalam de forma excepcional, com um aumento muito pequeno no tempo total de execução quando comparado a um aumento considerável no tamanho dos problemas, proporcionalmente ao aumento na quantidade de recursos computacionais envolvidos.

6. Conclusões

Este artigo apresentou uma nova abordagem para o desenvolvimento de simuladores baseados em uma família específica de métodos numéricos baseados em elementos finitos, chamada MHM. Nessa abordagem, a linguagem Erlang é usada na implementação de um algoritmo de coordenação, baseado no modelo de atores, de um conjunto de processos responsáveis pela execução distribuída do algoritmo numérico induzido pelo MHM. Os resultados experimentais apresentados neste artigo demonstram a viabilidade da solução proposta e, em particular, a possibilidade de se executar de forma eficiente métodos numéricos baseados em elementos finitos em nuvens.

O fato de Erlang não ser adequada para a computação numérica e da solução adotada neste trabalho para mitigar essa limitação envolver a integração com outras linguagens mais apropriadas para esse tipo de computação abre oportunidades para que outras famílias de métodos numéricos se beneficiem de seus atributos de qualidade. Como perspectiva de pesquisa futura, temos intenção de mostrar que outros

[4]Outras formas de medir o tamanho do problema incluem a quantidade total de variáveis a serem computadas e a qualidade da aproximação em termos de um conjunto específico de normas de erro, mas como a apresentação dessas medidas envolve uma apresentação mais detalhada dos aspectos matemáticos dos métodos MHM, essas outras formas são omitidas no presente artigo

métodos com características distintas do MHM também podem se beneficiar do modelo de atores. Em particular, estamos no momento investigando um outro método também desenvolvido no LNCC, baseado em volumes finitos e com maior nível de acoplamento [Müller et al. 2016], o que induz simuladores com estilo arquitetural distinto do *fan-out/fan-in* presente no MHM. Em seguida, pretendemos definir um ou mais *behaviors* Erlang suficientemente genéricos, desenhados na forma de um *framework* de aplicação, para atender demandas de diferentes estilos arquiteturais induzidos por diferentes métodos numéricos.

Agradecimentos

A pesquisa apresentada neste artigo foi parcialmente financiada pelo *EU H2020 Programme* e pela RNP por meio do projeto HPC4E (`http://www.hpc4e.eu`). Os experimentos descritos neste artigo empregaram os recursos computacionais providos pelo supercomputador SDumont (`http://sdumont.lncc.br`).

Os autores agradecem a Weslley Pereira e Frédéric Valentin do LNCC, e a Diego Paredes da PUC Valparaiso, Chile, pela revisão de versões anteriores do conteúdo presente neste artigo.

Referências

Acun, B., Gupta, A., Jain, N., Langer, A., Menon, H., Mikida, E., Ni, X., Robson, M., Sun, Y., Totoni, E., et al. (2014). Parallel programming with migratable objects: Charm++ in practice. In *High Performance Computing, Networking, Storage and Analysis, SC14: International Conference for*, pages 647–658. IEEE.

Agha, G. A. (1985). Actors: A model of concurrent computation in distributed systems. Technical report, DTIC Document.

Araya, R., Harder, C., Paredes, D., and Valentin, F. (2013). Multiscale hybrid-mixed method. *SIAM Journal on Numerical Analysis*, 51(6):3505–3531.

Ciarlet, P. (1978). *The finite element method for elliptic problems*. North-Holland.

Efendiev, Y., Lazarov, R., Moon, M., and Shi, K. (2015). A spectral multiscale hybridizable discontinuous Galerkin method for second order elliptic problems. *Computer Methods in Applied Mechanics and Engineering*, 292:243 – 256. Special Issue on Advances in Simulations of Subsurface Flow and Transport (Honoring Professor Mary F. Wheeler).

Harder, C., Paredes, D., and Valentin, F. (2013). A family of multiscale hybrid-mixed finite element methods for the Darcy equation with rough coefficients. *Journal of Computational Physics*, 245:107–130.

Hou, T. Y. and Wu, X.-H. (1997). A multiscale finite element method for elliptic problems in composite materials and porous media. *Journal of Computational Physics*, 134(1):169 – 189.

Huang, C., Lawlor, O., and Kalé, L. V. (2003). Adaptive MPI. In *Proceedings of the 16th International Workshop on Languages and Compilers for Parallel Computing (LCPC 2003), LNCS 2958*, pages 306–322, College Station, Texas.

Hughes, T. J. R. (1987). *The Finite Element Method: Linear Static and Dynamic Finite Element Analysis*. Prentice-Hall, Englewood Cliffs, New Jersey.

Mattsson, H., Nilsson, H., and Wikström, C. (1999). Mnesia—a distributed robust DBMS for telecommunications applications. In *Practical Aspects of Declarative Languages*, pages 152–163. Springer.

Müller, L. O., Blanco, P. J., Watanabe, S. M., and Feijóo, R. A. (2016). A high-order local time stepping finite volume solver for one-dimensional blood flow simulations: application to the ADAN model. *International Journal for Numerical Methods in Biomedical Engineering*, 32(10):e02761–n/a. e02761 cnm.2761.

Pironneau, O. (1989). *Finite Element Methods for Fluids*. John Wiley,New York.

Scalas, A., Casu, G., and Pili, P. (2008). High-performance technical computing with Erlang. In *Proceedings of the 7th ACM SIGPLAN workshop on ERLANG*, pages 49–60. ACM.

Análise distribuída em dados meteorológicos utilizando o modelo de atores

Jimmy K. M. Valverde-Sánchez, Otávio Carvalho, Eduardo Roloff, Nicolas Maillard, Philippe O. A. Navaux

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

{jkmvsanchez,omcarvalho,eroloff,nicolas,navaux}@inf.ufrgs.br

Abstract. *The GRIB scientific data format is widely used in the meteorological community to store weather and weather forecast data. These datasets are shared among the meteorological centers and used as an input to the weather models, both for weather forecasting and historical analysis. However, the current methods for processing files in this format no perform the computation in a distributed environment. This situation limits the analytical capabilities of scientists who need to perform analysis on large data sets in order to obtain information in the shortest time possible using of all available resources. In this sense, this work presents an alternative to data processing in this format, using the Actor model implemented in Akka toolkit, evaluate our proposal in the Cloud and compare it with other techniques.*

Resumo. *No âmbito da meteorologia o formato de dados científicos GRIB é amplamente utilizado para armazenar histórico de dados e previsões meteorológicas. Esses conjuntos de dados são compartilhados entre os centros meteorológicos e usados como uma entrada para os modelos de clima, tanto para a previsão do tempo e análise histórica. No entanto, as ferramentas atuais disponíveis e métodos para processar arquivos neste formato não realizam o processamento em um ambiente distribuído. Essa situação limita as capacidades de análise dos cientistas que precisam realizar uma análise sobre grandes conjuntos de dados com o objetivo de obter informação no menor tempo possível fazendo uso de todos os recursos disponíveis. Neste contexto, este trabalho apresenta uma alternativa ao processamento de dados nesse formato fazendo uso do modelo de atores implementado no Akka toolkit, avaliamos a nossa proposta na nuvem e a comparamos com outras técnicas.*

1. Introdução

Estamos vivendo na era de Big Data [Dobre and Xhafa 2014], que é resultado do massivo aumento nas quantidades de dados geradas, em intervalos de tempo cada vez menores, através de diversas fontes, tais como *smartphones*, sensores, simulações e *logs* de aplicações. O processamento dessas quantidades massivas de dados é um desafio não somente para as corporações, mas também para a comunidade científica, que gera quantidades massivas de dados em seus experimentos. Para que possamos realizar o processamento eficiente dessa nova gama de perfis de dados, necessitamos contar não somente com maiores quantidades de recursos computacionais, mas também utilizarmos melhores práticas de processamento.

Portanto, é cada vez mais necessário desenvolvermos aplicações capazes de extrair resultados válidos da informação contida nesses grandes conjuntos de dados. Na comunidade científica, podemos utilizar o conhecimento extraído desses dados com o objetivo de compreender fenômenos em áreas chaves do conhecimento, como meteorologia, sismologia e saúde.

O modelo de programação sequencial já não é suficiente para lidar com os requisitos destes grandes conjuntos de dados. Para que possamos extrair conhecimento desses perfis de dados, modelos de programação paralela e distribuída tornam-se valiosos para extrairmos informações em tempos de processamento razoáveis [Kambatla et al. 2014].

Os esforços para a análise e processamento de grandes conjuntos de dados, em ambientes paralelos e distribuídos tem sido focados principalmente em formatos de dados científicos específicos, tais como o NetCDF [Li et al. 2003], [Zhao et al. 2010], [Buck et al. 2011], [Wang et al. 2012] e o HDF5 [Wang et al. 2012], [Blanas et al. 2014]. Por outro lado, existem formatos de dados, amplamente utilizados pela comunidade científica, para os quais ainda não existem soluções explorando um ambiente paralelo e distribuído. Um exemplo disso é o formato GRIB[1], que é amplamente utilizado no âmbito da comunidade meteorológica, para armazenar e compartilhar o histórico de dados e previsões meteorológicas. Esse formato de dados, ao contrário de formatos com o NetCDF e HDF5, não suporta acesso aleatório aos dados [Fortner 1995]. Dessa forma, o processamento paralelo de larga escala, para esse tipo de arquivos, ainda é um desafio de pesquisa pouco explorado, carecendo de metodologias e ferramental específico.

O Centro de Previsão de Tempo e Estudos Climáticos (CPTEC[2]) é uma situação real onde o processamento de arquivos meteorológicos em formato GRIB é fundamental para a comunidade científica. Neste centro de pesquisa são executados periodicamente diversos modelos de previsões meteorológicas, abrangendo toda a América do Sul e os oceanos adjacentes. Os resultados dessas previsões e a condição inicial do modelo são fornecidas duas vezes por dia, utilizando o formato GRIB. Na fase de pré-processamento da execução do BRAMS[3], modelo de previsão numérica do tempo usado para simular condições atmosféricas, os arquivos GRIB devem ser convertidos em um arquivo intermediário que contém os dados de entrada para o modelo. O processamento é realizado usando um modelo sequencial, o qual leva um tempo considerável do processo global.

Neste trabalho, propomos uma maneira alternativa de processar e realizar a análise de grandes quantidades de dados no formato GRIB, realizando processamento distribuído através do modelo de atores, implementado no projeto Akka[4]. O Akka vem sido amplamente utilizada pela comunidade científica, principalmente através de projetos como Apache Spark e Flink, que usam o Akka internamente para sua comunicação distribuída. No entanto, existem limitações para conjuntos de dados que excedem largamente as capacidades de memória. Quando isso acontece no Spark, por exemplo, é necessário armazenar as abstrações de dados distribuídos (RDDs[5]) em disco, o que causa perda de desempenho ou pode acarretar até mesmo no lançamento de exceções *OutOfMemory* [Gu and Li 2013].

[1]http://www.nco.ncep.noaa.gov/pmb/docs/on388/
[2]http://www.cptec.inpe.br/
[3]http://brams.cptec.inpe.br
[4]http://akka.io/
[5]http://spark.apache.org/docs/latest/programming-guide.html

2. Motivação

Os atuais métodos para processamento de arquivos no formato GRIB (GRIB API[6], wgrib2[7]) apresentam uma carência muito relevante: A possibilidade de realizar o processamento utilizando mais de um servidor, de maneira distribuída. Dessa forma, a escalabilidade do processamento é limitada ao poder de processamento de apenas um único nó computacional.

A motivação desse trabalho é propor o uso de ferramentas que possibilitem explorar o processamento de maneira distribuída. Ao realizarmos o processamento de maneira distribuída, dois benefícios são alcançados. Primeiro, desenvolve-se a possibilidade de processamento de uma maior quantidade de arquivos de uma única vez. Segundo, através do processamento distribuído, existe a possibilidade de redução do tempo total de processamento através da divisão de tarefas entre os nós de processamento.

3. Conceitos Básicos

Nesta seção são apresentados conceitos úteis para obtermos um melhor entendimento das seções posteriores do artigo.

3.1. GRIB

O GRIB(GRIdded Binary) é um formato binário de dados científicos, criado pela Organização Meteorológica Mundial (WMO)[8], principalmente com o objetivo de transmitir e armazenar dados meteorológicos [WMO 2003]. Este formato é amplamente utilizado na comunidade meteorológica para armazenar dados históricos e de previsões meteorológicas [Markiewicz et al. 2013]. Contudo, este formato não é facilmente acessível [Candanedo et al. 2013], ao contrário de formatos similares como o NetCDF [Rew and Davis 1990] e o HDF5 [The HFD Group 2016], por não permitir acesso aleatório aos dados.

Ao longo do tempo, a versão 0 do formato tornou-se obsoleta, tendo sido substituída pela versão 1. Após certo tempo, foi proposta a versão 2 do formato, afim de permitir a representação de parâmetros não cobertos na versão 1 que eram necessários para os centros meteorológicos. A definição desses parâmetros e algumas informações dentro de uma mensagem GRIB são parte do padrão, mas não da mensagem, a qual armazena uma referência às informações definidas em uma tabela externa.

Cada arquivo pode conter uma ou várias mensagens e cada mensagem GRIB contém divisões lógicas chamadas seções, concretamente na versão 2 existem 9 seções. A primeira seção, *Indicator*, marca o início de cada mensagem e contém a Disciplina (a disciplina 0, por exemplo, corresponde a produtos meteorológicos) dos dados na mensagem, a versão e o tamanho total da mensagem em bytes. As demais seções, exceto a última, contém o tamanho da mensagem e o número da seção. A seção *Identification* define as características que se aplicam ao conteúdo da mensagem. A seção *Local use* é opcional e define itens adicionais que são próprios do centro meteorológico de origem. A seção *Grid definition* define a superfície da grade e a geometria dos valores na superfície (por

[6]https://software.ecmwf.int/wiki/display/GRIB/Home
[7]http://www.cpc.ncep.noaa.gov/products/wesley/wgrib2/index.html
[8]https://www.wmo.int/

exemplo, Latitude-longitude, Mercator). A seção *Product definition* contém a natureza dos dados. A seção *Data representation* define o método de empacotamento dos dados usado e a informação necessária para a decodificação dos mesmos (por exemplo, *Complex Packing and Spatial Differencing*). A seção *Bit-map* indica a presença ou a ausência de dados em cada ponto da grade. A seção *Data* contém os valores para cada ponto da grade e, finalmente a seção *End* indica o fim da mensagem com o código 7777.

3.2. Akka toolkit

O Akka visa facilitar o desenvolvimento de aplicações concorrentes e distribuídas, tendo sido baseado no Modelo de Atores proposto em [Hewitt et al. 1973]. Ele é baseado em entidades autônomas, denominadas Atores, que se comunicam com outros atores através de mensagens assíncronas. Cada ator possui uma caixa de correio, onde as mensagens recebidas são armazenadas até serem processadas. Os atores também possuem um comportamento que define como as mensagens recebidas devem ser processadas, os atores não compartilham estados, comunicando-se somente através de mensagens. Através dessas características, o Akka toolkit, que foi desenvolvido utilizando a linguagem de programação Scala, é capaz de produzir soluções altamente escaláveis, tanto vertical quanto horizontalmente [Roestenburg et al. 2016]. Alem disso, o Akka fornece *routers* do tipo round-robin, random entre outros, para realizar o balanceamento de carga sobre um entorno local e distribuído.

4. Metodologia

A presente proposta para processar arquivos GRIB em um ambiente paralelo e distribuído é baseada no Akka toolkit, o qual implementa o Modelo de Atores. O Akka foi utilizado principalmente para a distribuição das tarefas através dos nós membros do cluster. Para realizar este processamento, foi desenvolvido um módulo capaz de parsear e decodificar os arquivos GRIB. Esse módulo, por sua vez, também foi escrito em Scala, para obtermos melhor compatibilidade com o Akka. A Figura 1 ilustra a arquitetura proposta, onde podemos distinguir 2 tipos de máquinas: o nó Master e o nó Worker. No primeiro reside o nó Seed ❶ , o qual é padrão do Akka e responsável por formar o cluster, servindo como ponto inicial de contato para que os demais nós conectem-se ao cluster. O outro refere-se a um nó Akka com o papel de Master ❷ , onde reside o ator JobManager. No segundo residem os nós Akka com o papel de Worker ❸ , que contém aos atores do tipo JobWorker.

O conjunto de dados analisado nos experimentos foi armazenado no HDFS (com um fator de replicação de 2). Por esse motivo, na figura encontram-se um Namenode no nó Master, e Datanodes nos outros nós que atuam como Workers.

O trabalho realizado pelo JobManager e pelo ator JobWorker é baseado no modelo de comunicação **Manager-Worker** [Chandy and Taylor 1992], onde o **Worker** realiza requisições ao **Manager** das tarefas a serem processadas. O funcionamento da nossa proposta é ilustrado na Figura 2(a). No *Driver program* são lidas as configurações necessárias para a execução da aplicação, obtém-se a lista de arquivos GRIB a serem processados do HDFS, e uma primeira messagem, JobRequest, é enviada ao ator JobManager. No JobManager, um *router* do tipo *Broadcast* é criado e posteriormente utilizado para enviar uma mensagem *InitProcess* aos atores JobWorker. Consequentemente, cada JobWorker envia

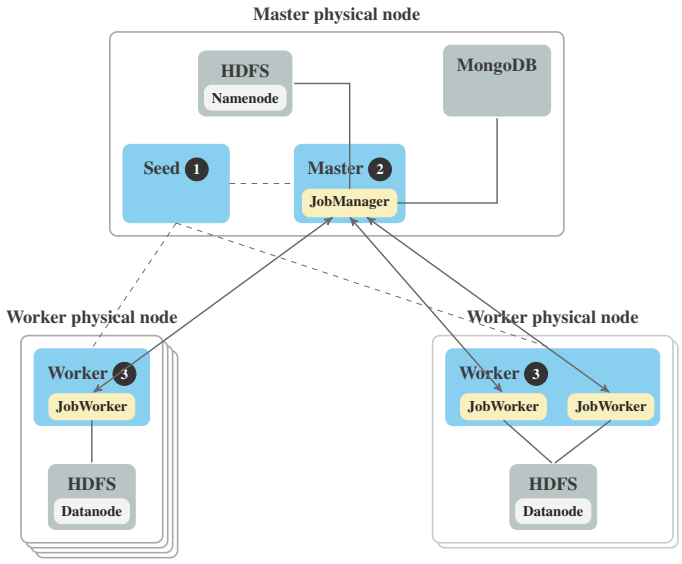


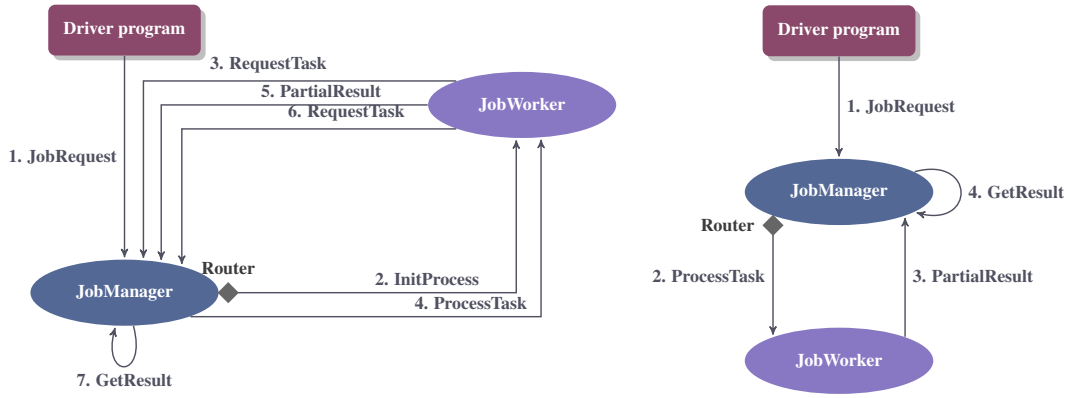
Figura 1. Arquitetura final proposta

uma mensagem *RequestTask* ao JobManager, para registrar-se como um executor de tarefas, e solicitar uma tarefa a ser processada. Todas as mensagens recebidas no JobManager são armazenadas na caixa de correio e atendidas na ordem de chegada. Desse modo, o JobManager seleciona uma tarefa e envia uma mensagem *ProcessTask* ao JobWorker com a informação da tarefa a ser realizada, porém dessa vez a mensagem é enviada diretamente, sem o uso do *router*. Nesse ponto, o ator JobWorker utiliza o módulo cliente em Scala para processar os arquivos GRIB armazenados no HDFS, e o resultado de cada mensagem GRIB é combinado para calcular um resultado parcial. Após calculado o resultado parcial, uma mensagem *PartialResult* é enviada ao ator JobManager, seguida de outra mensagem *RequestTask* solicitando uma nova tarefa a ser realizada. No JobManager os resultados parciais recebidos são persistidos em memória, e uma outra tarefa pendente é selecionada para ser enviada ao JobWorker através de uma mensagem *ProcessTask*. Esse ciclo se repete até que não hajam mais mensagens pendentes a serem processadas no JobManager e, quando isto acontece, o JobManager envia uma mensagem chamada *GetResult* a si mesmo, para calcular o resultado final a partir dos resultados parciais obtidos anteriormente.

Tabela 1. Número total de JobWorkers por cluster de VMs

Configuration	1 VM	2 VMs	4 VMs	8 VMs
4*WN* - 1*JW*	4	8	16	32
2*WN* - 2*JW*	4	8	16	32
4*WN* - 2*JW*	8	16	32	64
2*WN* - 4*JW*	8	16	32	64
4*WN* - 4*JW*	16	32	64	128
2*WN* - 8*JW*	16	32	64	128

A escolha das tarefas a serem processadas pelos atores JobWorker pode ocorrer de duas formas: Baseado em arquivo e baseado em mensagens. No primeiro, cada ator



(a) Usando o modelo Manager-Worker.

(b) Usando os routers fornecidos pelo Akka.

Figura 2. Fluxograma da nossa proposta e de uma segunda abordagem.

JobWorker processa um arquivo inteiro com todas as mensagens (175 por arquivo). Na escolha de tarefas baseada em mensagens, o JobManager particiona o arquivo em grupos de mensagens, podendo variar entre 22, 44 ou 88 mensagens, sendo essa etapa adicional realizada pelo JobManager. Esses números foram selecionados para dividir cada arquivo da forma mais equitativa possível. Dessa forma, se o número de mensagens a serem processadas por cada tarefa for 44, um arquivo GRIB será processado por quatro JobWorkers, onde 3 JobWorkers processarão 44 mensagens e 1 JobWorker processará apenas 43 mensagens.

Múltiplos atores JobWorker podem ser executados em cada nó Akka do tipo Worker. Através da Tabela 1, podemos visualizar o número de JobWorkers utilizado por cada conjunto de máquinas virtuais. Dessa forma, por exemplo, a configuração **2WN-4JW** mostra que 2 nós Akka do tipo Worker (**WN**) serão executado em cada máquina, dos quais cada um conterá 4 JobWorkers (**JW**). De modo que, para 1 VM serão criados 8 JobWorkers, para 2 VMs serão criados 16 JobWorkers, para 4 VMs serão 32 atores JobWorker e, por fim, para 8 VMs serão criados 64 JobWorkers. As demais configurações apresentadas na tabela seguem o mesmo padrão, conforme aumenta o número de máquinas virtuais.

Além disso, uma segunda abordagem é apresentada, com o objetivo de comparar a nossa proposta com relação a outras opções fornecidas pelo Akka. Os *routers* utilizados para realizar o estudo comparativo foram RoundRobin, Random e AdaptiveLoadBalancing. O AdaptiveLoadBalancing utiliza métricas do cluster para comunicar-se com os atores, podendo ser configurado para usar métricas tais como: *cpu*, para o nível de utilização da CPU; *heap*, que utiliza como métrica a pilha disponível da JVM; e a métrica *mix*, que combina as métricas de cpu, heap e load. Essas métricas foram coletadas utilizando a biblioteca Hyperic Sigar[9].

Figura 2(a) ilustra como essa outra abordagem funciona. De forma similar ao explicado anteriormente, os parâmetros necessários para a execução da aplicação são carregados no *Driver program* e uma primeira mensagem é enviada ao ator JobManager, denominada JobRequest, contendo a lista dos arquivos GRIB a serem processados. Nesse ator, um router será criado e o mesmo será encarregado de distribuir as mensagens aos

[9]https://github.com/hyperic/sigar

JobWorkers de acordo com a lógica do *router* especificado. O JobWorker, por sua vez, processará uma por uma as mensagens, *ProcessTask*, recebidas e armazenadas na caixa de correio do ator. O resultado parcial obtido será enviado ao JobManager na mensagem *PartialResult*. Esse processo continuará até não houverem mais mensagens a ser enviadas no JobManager e o JobWorkers tenham processado todas as tarefas recebidas. Quando isso acontecer, o JobManager enviará uma mensagem *GetResult* para si mesmo, e da mesma forma o resultado final será calculado a partir dos resultados parciais recebidos nas mensagens anteriores.

5. Resultados

Os experimentos foram executados usando Java 1.8.0 60, Scala 2.11.7, Akka 2.3.4 e Apache Hadoop 2.6.0, em um *cluster* no Windows Azure. Esta plataforma foi escolhida devido aos resultados em Roloff et al. [Roloff et al. 2012], onde o Windows Azure apresentou melhores resultados, com relação à eficiência de custo e desempenho, em comparação com os serviços em nuvem da Amazon e Rackspace. O cluster criado é composto por 9 instâncias do tipo A6, cada uma com 4 núcleos virtuais, 28 GB de memória e sistema operacional Ubuntu 14.04 LTS.

5.1. Descrição do conjunto de dados

O conjunto de dados usado nos experimentos contém 184 arquivos, cada um desses arquivos é composto de 175 mensagens GRIB, version 2 (GRIB2), em um conjunto total de 32200 mensagens, correspondentes a apenas três meses de dados. Esses arquivos foram obtidos através do FTP do CPTEC[10], onde encontram-se organizados em duas pastas por dia, uma correspondente às 00:00 e a outra às 12:00.

O tamanho total do conjunto de dados é de 13 GB, mas após decodificado e transformado em texto plano contendo todos os valores, o tamanho total é de 378 GB, o qual excede amplamente a quantidade de memória disponível no cluster utilizado. Este tamanho do conjunto de dados foi escolhido especificamente para acelerar a execução dos experimentos. Latitude-longitude é o tipo de grade utilizado para todas as mensagens GRIB. Além disso, cada arquivo é composto de 14 mensagens GRIB usando o **Complex packing** e 161 mensagens com o método de empacotamento **Complex packing and spatial differencing**. A maioria das mensagens contêm 1584353 pontos ou valores de dados. Da mesma forma, ao analisar todo o conjunto de dados foram encontrados 27 tipos diferentes de parâmetros. Assim, por exemplo, Temperatura, Umidade relativa e Precipitação Total tiveram no total 3864, 3496 e 184 ocorrências, respectivamente.

Para esse conjunto de dados, foram realizados experimentos abordando dois tipos de cenários. O primeiro cenário, realiza o cálculo do *average* para os parâmetros Temperatura e Umidade relativa. O segundo cenário, por sua vez, calcula o *average* para todos os parâmetros disponíveis no conjunto de dados.

Os resultados dos experimentos em 5.2 e 5.3 apresentam 6 métodos de execução. O método que utiliza a nossa proposta é denominado **pGrib**, enquanto os *routers round-robin* e *random* são denotados como **rr** e **ra**, respectivamente. Finalmente, o router *adaptive load balancing*, utilizando as métricas de CPU, Heap e mix, são denotados respectivamente como **cmc**, **cmh** e **cmm**. Assim, para cada um destes métodos, foram executados

[10] ftp://ftp1.cptec.inpe.br/modelos/io/tempo/regional/BRAMS05km/grib/

um total de 24 combinações, sendo elas: **4WN-1JW**, **2WN-2JW**, **4WN-2JW**, **2WN-4JW**, **4WN-4JW**, e **2WN-8JW**. Para cada um das configurações descritas, o tamanho do numero de mensagens processadas por cada JobWorker varia entre **22**, **44**, **88** e **175**.

5.2. Primeiro cenário

Neste cenário, são apenas considerados os parâmetros Temperatura e Umidade relativa. A Figura 3 apresenta os resultados para cada um dos métodos de execução, considerando o melhor e o pior caso obtidos. Para este cenário, os melhores casos foram obtidos usado 2WN e variando o número de JW por cada WN entre 4 e 8 na maioria dos casos.

Neste mesmo cenário, para a nossa proposta, baseando-se em arquivo ou grupo de mensagens, os melhores casos foram obtidos com a configuração 2WN-4JW. A única exceção ocorreu quando utilizamos 8 VMs e as mensagens foram agrupadas. Nesse caso, a melhor configuração foi 2WN-2JW-88, que ocupou 30.89 segundos a mais do que a 2WN-4JW-175. Dessa forma, houve uma ligeira variação de 1.6% para 1 VM, 2.55% para 2 VMs, 1.54% para 4 VMs, e 5.21% para 8 VMs entre a melhor e segunda melhor combinação para a nossa proposta.

A segunda melhor opção para processar os arquivos GRIB foi obtida utilizando-se o método rr. Os piores casos, por outro lado, foram obtidos utilizando 4WN e 1JW. O rr mostrou uma diferença significativa em relação aos outros métodos.

Quando utilizaram-se 8 VMs como *workers* o pGrib obteve um ganho significativo de 35.21%, 42.72%, 43.2%, 42.28% e 42.74% em relação aos métodos rr, ra, cmc, cmh e cmc, respectivamente.

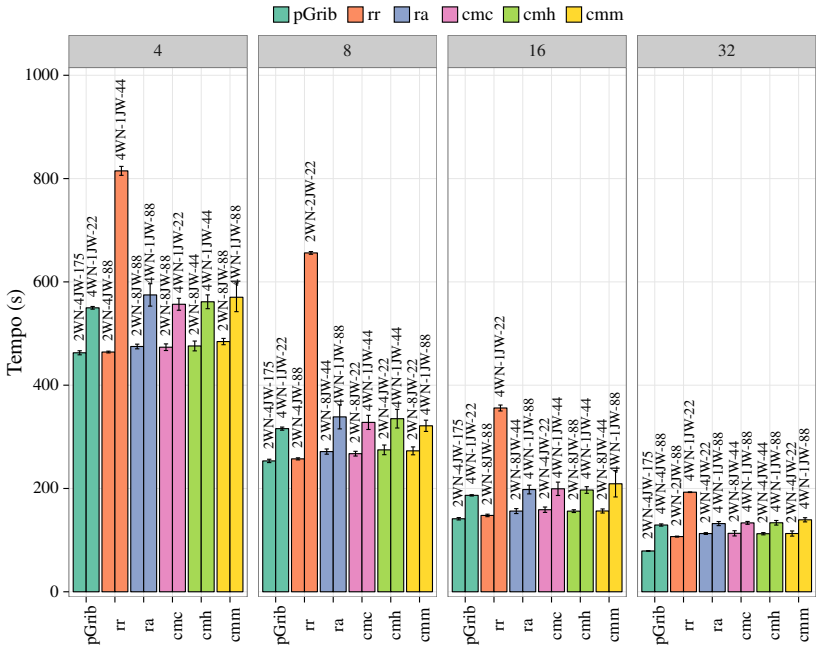


Figura 3. Tempo de execução do melhor e pior caso de cada um dos métodos de execução.

O melhor e pior speedup alcançado por todos os métodos são mostrados na Figura 4. O *baseline* utilizado foi o tempo de execução sequencial do módulo cliente, implementado para parsear os arquivos GRIB.

A Figura 4 mostra que, com o aumento do número de nós, todos os métodos obtêm boa escalabilidade, especialmente o método proposto neste trabalho. No entanto, podemos observar que para o pGrib o speedup diminui ligeiramente após 4 VMs, enquanto para os outros métodos o speedup diminui consideravelmente. Uma possível explicação para isto é que nem todas as mensagens são processadas pelo JobWorker.

Por exemplo, considerando o caso em que 1 JobWorker precise processar 44 mensagens, para as quais apenas 16 mensagens são Temperatura e Umidade relativa, 28 mensagens não seriam mais necessárias. Ainda assim, algumas seções da mensagem precisam ser analisadas, para sabermos se a mensagem GRIB precisa ser decodificada ou não. Esse processamento, por sua vez, requer leituras adicionais de dados no HDFS e processamento parcial das mensagens, acarretando as custos adicionais de processamento.

No entanto, ao implementar o modelo Manager-Worker, a nossa proposta consegue um balanceamento de carga automático, resultando em uma melhor distribuição das tarefas.

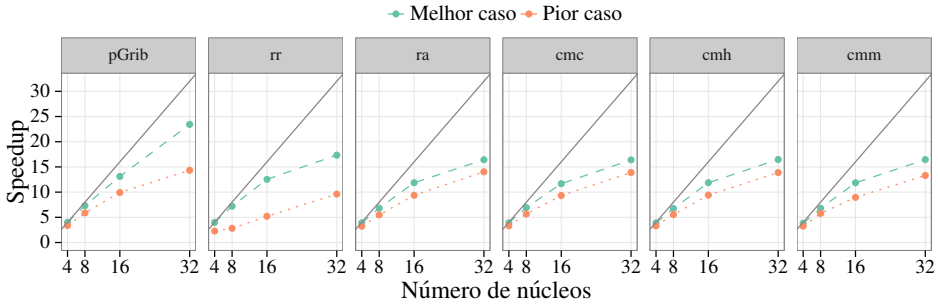


Figura 4. Speedup do melhor e pior caso de cada um dos métodos de execução.

5.3. Segundo cenário

Neste cenário, são considerados todos os parâmetros do conjunto de dados. A Figura 5 apresenta o tempo gasto por cada um dos métodos de execução, levando em conta o melhor e o pior caso. Considerando apenas uma VM, o melhor tempo foi obtido usando o rr, com um ganho de 1% em relação ao pGrib. Ao utilizarmos 2 VMs, 4 VMs e 8 VMs, o pGrib obteve uma melhor performance ao processar os arquivos GRIB quando dividimos os arquivos em grupos de 88, 44 e 22 mensagens, respectivamente. De forma semelhante ao experimento anterior, os melhores resultados para a nossa proposta foram obtidos utilizando-se a configuração 2WN-4JW. Da mesma forma, os piores resultados foram usando 4WN e apenas 1JW por cada WN.

Ao utilizarmos 8 VMs como *workers*, o pGrib obteve um ganho de 3.22%, 11.36%, 11.78%, 15.09% e 11.7%, respectivamente, em relação aos métodos rr, ra, cmc, cmh e cmc.

O speedup dos melhores e piores casos para este segundo cenário é apresentado na Figura 6, onde podemos perceber diferenças em relação ao cenário anterior. Os melhores valores de escalabilidade, para todos os métodos de execução, foram obtidos utilizando-se todos os parâmetros. Isso decorre do fato de não ocorrerem leituras no HDFS, nem seções das mensagens GRIB sendo analisadas desnecessariamente, resultando assim em um melhor desempenho.

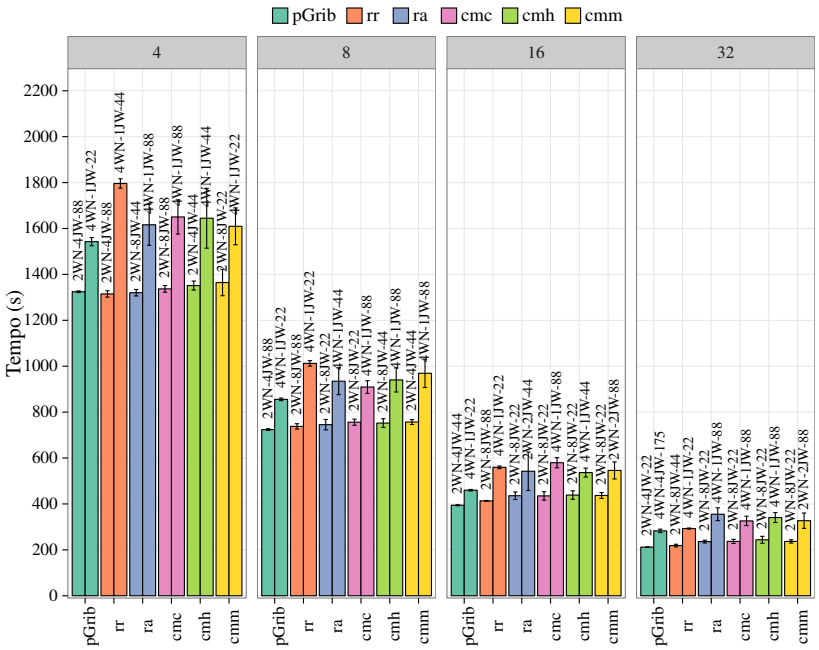


Figura 5. Tempo de execução do melhor e pior caso de cada um dos métodos de execução.

5.4. Usando Metadados

Após analisar os resultados do primeiro cenário, foi observada uma diferença entre a abordagens baseada em arquivo e por agrupamento por mensagens, o que resultou em uma perda de 28.11% na segunda abordagem, dentre as melhores configurações de cada estratégia. Esta perda é resultante do fato de que, na abordagem baseada em agrupamento de mensagens, quando precisamos escolher uma tarefa a ser enviada, o JobManager precisa realizar um pré-processamento, a fim de identificar o começo de cada mensagem GRIB dentro do arquivo, e posteriormente enviar cada grupo de mensagens ao JobWorker para ser processado. Porém, todo o pré-processamento é realizado sob demanda no JobManager, por arquivo, resultando em uma perda de desempenho. No segundo cenário, no entanto, essa diferença não ocorre. Devido ao fato de que todos os parâmetros são processados, não ocorrendo leituras desnecessárias no HDFS.

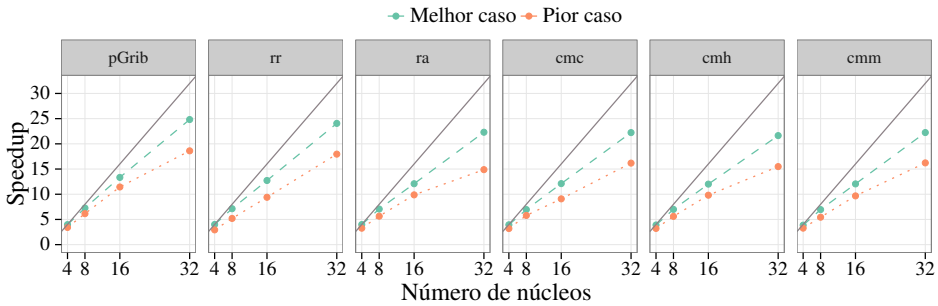


Figura 6. Speedup do melhor e pior caso de cada um dos métodos de execução.

Por esse motivo, foi realizada uma otimização sobre a abordagem baseada no agrupamento de mensagens. A otimização consiste em realizar o pré-processamento dos arquivos GRIB, para obter a ponto de início e o parâmetro que identifica cada mensagem GRIB (podendo ser salvas outras informações). O resultado desse processamento foi

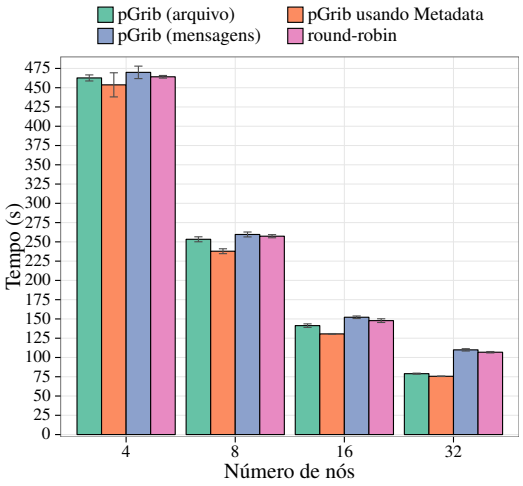


Figura 7. Tempo de execução para os melhores caso obtidos no primeiro cenário.

armazenado no banco de dados MongoDB, de forma que o JobManager possa realizar uma consulta ao MongoDB para obter os metadados relacionados a um arquivo. O pré-processamento e o posterior armazenamento em MongoDB resultou em um tempo total de 179.47 segundos, porém é uma operação realizada uma única vez, podendo ser reutilizada em todas as demais execuções.

A Figura 7 mostra os tempos de execução do pGrib com a estratégia baseada em arquivo, em mensagens, usando metadados, e round-robin. Como esperado, o uso de metadados obteve resultados satisfatórios. De forma que, levando em consideração a utilização de apenas uma VM, a otimização resultou em um ganho de 1.98% em relação ao pGrib (baseado em arquivo). Ao utilizarmos 2 e 4 VMs, a amplitude do ganho foi ainda maior, resultando em ganhos de 6.51% e 8.23% respectivamente. Ao utilizarmos 8 VMs, o ganho foi de 4.48%. Porém, com relação ao pGrib (baseado em mensagens) e round-robin, o ganho foi de 45.34% e 41.27%, respectivamente. Embora não exista um grande ganho entre os melhores casos da proposta inicial e a otimização realizada, existe um ganho considerável em relação aos resultados obtidos com a estratégia baseada em mensagem, o qual era o objetivo desta otimização.

5.5. Comparação com Apache Spark

Neste experimento, apresentamos uma comparação entre a nossa proposta e uma versão que processa os arquivos GRIB utilizando o Apache Spark (versão 1.3.1). Para este caso, a primeira abordagem contemplada foi a utilização do módulo cliente Scala para analisar e decodificar os arquivos GRIB. Porém, ao persistirmos os dados decodificados nos RDDs, ocorriam exceções do tipo *OutOfMemory*, já que os valores de dados decodificados ocupam mais memória do que o disponível. Dessa forma, optamos por armazenar nos RDDs apenas os resultados obtidos do módulo cliente Scala, da mesma forma como é realizado em nossa proposta. Assim, quando utilizada 1 VM, houve 1 *executor* e 4 *cores*, para 2, 4 e 8 VMs usou-se 2, 4 e 8 *executors* e 4 *cores* por *executor*, além de 24GB por *executor*.

A Figura 8 mostra os tempos de execução para o primeiro cenário (lado esquerdo), e para o segundo cenário (lado direito), utilizando os melhores casos obtidos com a nossa

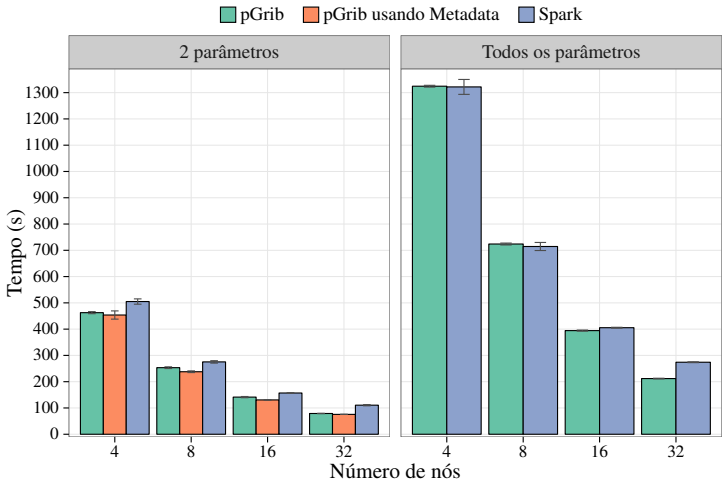


Figura 8. Tempo de execução para o melhor caso com a nossa proposta e o Apache Spark.

proposta e com o Apache Spark. De acordo com o primeiro cenário, o pGrib com o uso de metadados obteve um ganho de 11.3%, 15.74%, 20.19% e 46.31% usando 1, 2, 4 e 8 VMs, respectivamente. Para o segundo cenário, utilizando 1 e 2 VMs, o Spark foi ligeiramente melhor com um ganho de 0.21% e 1.32%, respectivamente, em relação ao pGrib com processamento baseado em mensagens. No entanto, ao utilizarmos 4 VMs, o pGrib obteve um ganho de 2.8%. Enquanto com 8 VMs o ganho foi de 29.35%. Uma explicação para a obtenção da performance superior ao utilizarmos 8 VMs, deve-se ao fato que inicialmente os arquivos GRIB foram armazenados no HDFS utilizando apenas 4 nós. De modo que, para 8 nós, uma maior transmissão de dados é necessária, causando uma perda de desempenho no Apache Spark, que em seu comportamento padrão precisa ter todos os dados disponíveis na memória antes de realizar as operações.

6. Trabalhos Relacionados

Existe um grande esforço de pesquisa sobre o processamento de grandes quantidades de dados científicos em ambientes paralelos e distribuídos. Porém, estes trabalhos são focados principalmente nos formatos NetCDF [Rew and Davis 1990] e HDF5 [The HFD Group 2016]. Li et al. [Li et al. 2003] propõem uma interface paralela para ler e escrever arquivos NetCDF, o qual está baseado em MPI-IO. Zhao et al. [Zhao et al. 2010] propõem um método para armazenar e acessar volumes massivos de dados de dados no formato NetCDF, baseado em processamento através do Apache Hadoop. No entanto, na fase inicial, os conjuntos de dados precisam ser analisados e transformados em formato texto para serem armazenados no HDFS, causando assim uma sobrecarga no volume de armazenamento de dados desnecessário.

Buck et al. [Buck et al. 2011], implementaram o SciHadoop, um *plugin* sobre o Apache Hadoop que permite o processamento de dados, em formato NetCDF, assim como a execução de consultas lógicas sobre esse modelo. Nesse trabalho, aplicam a função *mediana* sobre os dados para um determinado intervalo de tempo, e um determinado intervalo de área para analisar a pressão de ar. Além disso, neste trabalho, estendem a biblioteca NetCDF-Java para trabalhar com HDFS. O SciMate Framework, apresentado

por Wang et al. [Wang et al. 2012], é uma ferramenta que permite processar os formatos de dados científicos NetCDF e HDF5, assim como arquivos de texto puros, permitindo também a extensibilidade para outros formatos de dados científicos. Esse trabalho é baseado no sistema Mate, o qual permite processar dados usando o modelo de programação MapReduce. Blanas et al. [Blanas et al. 2014] propuseram um sistema que pode realizar a análise de dados científicos diretamente sobre conjuntos de dados armazenados no formato HDF5, executando as consultas em memória e usando a indexação através de *bitmap*, aproveitando-se das grandes quantidades de memória disponíveis nos supercomputadores.

7. Conclusões

Este trabalho apresenta uma análise de desempenho do processamento de arquivos GRIB, utilizando o modelo de comunicação Manager-Worker implementado no modelo de atores. Foi realizada uma comparação entre a nossa proposta, os *routers* suportados pelo Akka tookit que fornecem escalonadores de tarefas, e o Apache Spark. Além disso, foi realizada uma otimização sobre a proposta inicial, com o objetivo de evitar leituras do HDFS e decodificação desnecessária das mensagens GRIB. Os resultados obtidos demostraram a escalabilidade da solução, quando variados o número de máquinas virtuais e o número de atores por nó do Akka. Dado que os atores não são mapeados em uma relação 1-1 com as *threads* do sistema, e que atores são uma primitiva de alto nível implementada com base em *threads* JVM gerenciadas pelo Akka, termos apenas um ator por nó do Akka não apresentou uma boa eficiência. Por outro lado, dispararmos mais atores por nó do Akka resultou em uma melhor performance.

Como trabalho futuro, planejamos realizar uma análise sobre o comportamento dos atores JobManager e JobWorker para conseguir um melhor paralelismo, generalizar a implementação proposta para ser capaz de processar outros formatos de dados científicos.

8. Agradecimentos

Esta pesquisa recebeu financiamento do Programa H2020 da União Européia e do MCTI/RNP-Brasil no âmbito do projeto HPC4E, contrato de subvenção No. 689772. Esta pesquisa também recebeu financiamento da FAPERGS, no âmbito do projeto GreenCloud.

Referências

Blanas, S., Wu, K., Byna, S., Dong, B., and Shoshani, A. (2014). Parallel Data Analysis Directly on Scientific File Formats. In *Proceedings of the 2014 ACM SIGMOD International Conference on Management of Data*, SIGMOD '14.

Buck, J. B., Watkins, N., LeFevre, J., Ioannidou, K., Maltzahn, C., Polyzotis, N., and Brandt, S. (2011). SciHadoop: Array-based Query Processing in Hadoop. In *Proceedings of 2011 International Conference for High Performance Computing, Networking, Storage and Analysis*, SC '11.

Candanedo, J. A., Paradis, E., and Stylianou, M. (2013). Building Simulation Weather Forecast Files for Predictive Control Strategies. In *Proceedings of the Symposium on Simulation for Architecture & Urban Design*, SimAUD '13.

Chandy, K. and Taylor, S. (1992). *An introduction to Parallel Programming*. Jones and Bartlett.

Dobre, C. and Xhafa, F. (2014). Parallel Programming Paradigms and Frameworks in Big Data Era. *International Journal of Parallel Programming*, 42(5).

Fortner, B. (1995). *The Data Handbook: A Guide to Understanding the Organization and Visualization of Technical Data*. Springer-Verlag.

Gu, L. and Li, H. (2013). Memory or Time: Performance Evaluation for Iterative Operation on Hadoop and Spark. In *High Performance Computing and Communications 2013 IEEE International Conference on Embedded and Ubiquitous Computing (HPCC_EUC), 2013 IEEE 10th International Conference on*.

Hewitt, C., Bishop, P., and Steiger, R. (1973). A Universal Modular ACTOR Formalism for Artificial Intelligence. In *Proceedings of the 3rd International Joint Conference on Artificial Intelligence*, IJCAI'73.

Kambatla, K., Kollias, G., Kumar, V., and Grama, A. (2014). Trends in big data analytics. *Journal of Parallel and Distributed Computing* , 74(7):2561 – 2573. Special Issue on Perspectives on Parallel and Distributed Processing.

Li, J., Liao, W.-k., Choudhary, A., Ross, R., Thakur, R., Gropp, W., Latham, R., Siegel, A., Gallagher, B., and Zingale, M. (2003). Parallel netCDF: A High-Performance Scientific I/O Interface. In *Proceedings of the 2003 ACM/IEEE Conference on Supercomputing*, SC '03.

Markiewicz, Ł., Chybicki, A., Drypczewski, K., Bruniecki, K., and Dbrowski, J. (2013). Operational Enhancement of Numerical Weather Prediction with Data from Real-time Satellite Images. In Weintrit, A., editor, *Marine Navigation and Safety of Sea Transportation: Navigational Problems*, chapter 5, pages 153–160.

Rew, R. and Davis, G. (1990). NetCDF: an interface for scientific data access. *IEEE Computer Graphics and Applications*, 10(4):76–82.

Roestenburg, R., Bakker, R., and Williams, R. (2016). *Akka in Action*. Manning Publications Co., Greenwich, CT, USA, 1st edition.

Roloff, E., Birck, F., Diener, M., Carissimi, A., and Navaux, P. (2012). Evaluating High Performance Computing on the Windows Azure Platform. In *Cloud Computing (CLOUD), 2012 IEEE 5th International Conference on*.

The HFD Group (1987-2016). Hierarchical Data Format, version 5.

Wang, Y., Jiang, W., and Agrawal, G. (2012). SciMATE: A Novel MapReduce-Like Framework for Multiple Scientific Data Formats. In *Proceedings of the 2012 12th IEEE/ACM International Symposium on Cluster, Cloud and Grid Computing (Ccgrid 2012)*, CCGRID '12.

WMO (2003). Guide to the WMO Table Driven Code Form Used for the Representation and Exchange of Regularly Spaced Data In Binary Form: FM 92 GRIB Edition 2. Technical report, World Meteorological Organization.

Zhao, H., Ai, S., Lv, Z., and Li, B. (2010). Parallel Accessing Massive NetCDF Data Based on Mapreduce. In *Proceedings of the 2010 International Conference on Web Information Systems and Mining*, WISM'10.

Realização

Apoio Fomento

Apoio Institucional

Patrocinador Diamante

Núcleo de Informação e Coordenação do Ponto BR

Comitê Gestor da Internet no Brasil

Patrocinador Ouro

Patrocinador Bronze