Programa Interdisciplinar de Pós-Graduação em

Computação Aplicada

Mestrado Acadêmico

Vinicius Facco Rodrigues

AutoElastic: Explorando a Elasticidade de Recursos de Computação em Nuvem para a Execução de Aplicações de Alto Desempenho Iterativas

São Leopoldo, 2016

Vinicius Facco Rodrigues

AUTOELASTIC: EXPLORANDO A ELASTICIDADE DE RECURSOS DE COMPUTAÇÃO EM NUVEM PARA A EXECUÇÃO DE APLICAÇÕES DE ALTO DESEMPENHO ITERATIVAS

Dissertação apresentada como requisito parcial para a obtenção do título de Mestre pelo Programa de Pós-Graduação em Computação Aplicada da Universidade do Vale do Rio dos Sinos — UNISINOS

Orientador:
Prof. Dr. Rodrigo da Rosa Righi

São Leopoldo
2016

R696a Rodrigues, Vinicius Facco.
Autoelastic: explorando a elasticidade de recursos de computação em nuvem para a execução de aplicações de alto desempenho iterativa / Vinicius Facco Rodrigues. – 2016.
125 f. : il. color. ; 30 cm.

Dissertação (mestrado) – Universidade do Vale do Rio dos Sinos, Programa de Pós-Graduação em Computação Aplicada, 2016.
"Orientador: Prof. Dr. Rodrigo da Rosa Righi."

1. Computação em nuvem. 2. Computação de alto desempenho. 3. Sistemas operacionais distribuídos (Computadores). I. Título.

CDU 004.7

Dados Internacionais de Catalogação na Publicação (CIP)
(Bibliotecária: Carla Maria Goulart de Moraes – CRB 10/1252)

UNIVERSIDADE DO VALE DO RIO DOS SINOS
Escola Politécnica
Programa Interdisciplinar de Pós-Graduação em Computação Aplicada – PIPCA

Vinicius Facco Rodrigues

AutoElastic:
Explorando a Elasticidade de Recursos de Computação em Nuvem para a Execução de Aplicações de Alto Desempenho Iterativas

Dissertação apresentada à Universidade do Vale do Rio dos Sinos – Unisinos, como requisito parcial para obtenção do título de Mestre em Computação Aplicada.

Aprovado em 29 de fevereiro de 2016

BANCA EXAMINADORA

Prof. Dr. Rodrigo da Rosa Righi – UNISINOS

Prof. Dr. Cristiano André da Costa – UNISINOS

Prof. Dr. Lisandro Zambenedetti Granville– UFRGS

Prof. Dr. Rodrigo da Rosa Righi

Visto e permitida a impressão
São Leopoldo,

Prof. Dr. Cristiano André da Costa
Coordenador PPG em Computação Aplicada

AGRADECIMENTOS

Gostaria de agradecer minha família pelo eterno apoio e em especial minha Mãe. Ela é a responsável por tudo e sem ela este trabalho jamais teria sido realizado. Agradeço também, aos meus amigos e colegas, sempre presentes e apoiando meus passos em cada momento durante a realização deste trabalho. Em especial, agradeço ao meu professor orientador, o qual teve participação decisiva para a realização destes trabalho, conduzindo com maestria cada passo realizado. Agradeço, também, aos seguintes órgãos de pesquisa que tornaram possível a realização destes trabalho: CNPq (Conselho Nacional de Desenvolvimento Científico e Tecnológico), CAPES (Coordenação de Aperfeiçoamento de Pessoal de Nível Superior) e FAPERGS (Fundacão de Amparo à Pesquisa do Estado do Rio Grande do Sul). Obrigado a todos!

RESUMO

Elasticidade de recursos é uma das características chave da Computação em Nuvem. Através dessa funcionalidade, recursos computacionais podem ser adicionados ou removidos ao ambiente a qualquer momento, permitindo aplicações escalarem dinamicamente, evitando provisionamento excessivo ou restrito de recursos. Considerando a área de computação de alto desempenho, conhecida também como HPC (*High Performance Computing*), iniciativas baseadas em sacola-de-tarefas utilizam um balanceador de carga e instâncias de máquinas virtuais (VM) fracamente acopladas. Neste cenário, os processos desempenham papéis independentes, facilitando a adição e remoção de VM's pois o balanceador de carga se encarrega de distribuir tarefas entre os processos das VM's ativas. Entretanto, aplicações HPC iterativas se caracterizam por serem fortemente acopladas e terem dificuldade de obter vantagem da elasticidade pois, em tais aplicações, geralmente os processos são fixos durante todo o tempo de execução. Devido a isso, a simples adição de novos recursos não garante que os mesmos serão utilizados pelos processos da aplicação. Além disso, a remoção de processos pode comprometer a inteira execução da aplicação, pois cada processo desempenha um papel fundamental em seu ciclo de execução. Aplicações iterativas voltadas para HPC são comumente implementadas utilizando MPI (*Message Passing Interface*) e neste contexto, fazer o uso da elasticidade torna-se um desafio pois é necessária a reescrita do código fonte para o tratamento da reorganização de recursos. Tal estratégia muitas vezes requer um conhecimento prévio do comportamento da aplicação, sendo necessárias interrupções do fluxo de execução nos momentos de reorganização de recursos. Além disso, utilizando MPI 2.0, em que há a possibilidade da alteração da quantidade de processos em tempo de execução, existem problemas relacionados em como tirar proveito da elasticidade pois o desenvolvedor deve por si mesmo gerenciar a reorganização da topologia de comunicação. Ainda, consolidações repentinas de máquinas virtuais que executam processos da aplicação podem comprometer a sua execução. Focando nessas questões, propõe-se nessa dissertação um modelo de elasticidade baseado na camada PaaS (*Platform as a Service*) da nuvem, chamado AutoElastic. AutoElastic age como um *middleware* permitindo que aplicações HPC iterativas obtenham vantagem do provisionamento de recursos dinâmico de uma infraestrutura de nuvem sem a necessidade de modificações no código fonte. AutoElastic oferece a elasticidade de forma automática, não sendo necessária a configuração de regras por parte do usuário. O mecanismo de elasticidade conta com a utilização de *threholds* fixos além de oferecer uma nova abordagem em que eles se auto ajustam durante a execução da aplicação. Ainda, AutoElastic oferece também um novo conceito nomeado como elasticidade assíncrona, o qual oferece um arcabouço para permitir que a execução de aplicações não seja bloqueada enquanto recursos são adicionados ou removidos do ambiente. A viabilidade de AutoElastic é demonstrada através de um protótipo que executa uma aplicação de integração numérica *CPU-Bound* sobre a plataforma de nuvem OpenNebula. Resultados com tal aplicação demonstraram ganhos de desempenho de 28,4% a 59% quando comparadas diferentes execuções elásticas e não elásticas. Além disso, testes com diferentes parametrizações de *thresholds* e diferentes cargas de trabalho demonstraram que no uso de *thresholds* fixos, o valor do *threshold* superior possui maior impacto que o inferior no desempenho e consumo de recursos por parte da aplicação.

Palavras-chave: Computação em Nuvem. Elasticidade de Recursos. Computação de Alto Desempenho.

ABSTRACT

Elasticity is one of the key features of cloud computing. Using this functionality, we can increase or decrease the amount of computational resources of the cloud at any time, enabling applications to dynamically scale computing and storage resources, avoiding over- and under-provisioning. In high performance computing (HPC), initiatives like bag-of-tasks or key-value applications use a load balancer and a loosely-coupled set of virtual machine (VM) instances. In this scenario, it is easier to add or remove virtual machines because the load balancer is in charge of distribute tasks between the active processes. However, iterative HPC applications are characterized by being tightly-coupled and have difficulty to take advantage of the elasticity because in such applications the amount of processes is fixed throughout the application runtime. In fact, the simple addition of new resources does not guarantee that the processes will use them. Moreover, removing a single process can compromise the entire execution of the application because each process plays a key role in its execution cycle. Iterative applications related to HPC are commonly implemented using MPI (Message Passing Interface). In the joint-field of MPI and tightly-coupled HPC applications, it is a challenge use the elasticity feature since we need re-write the source code to address resource reorganization. Such strategy requires prior knowledge of application behaviour, requiring stop-reconfigure-and-go approaches when reorganizing resources. Besides, using MPI 2.0, in which the number of process can be changed during the application execution, there are problems related to how profit this new feature in the HPC scope, since the developer needs to handle the communication topology by himself. Moreover, sudden consolidation of a VM, together with a process, can compromise the entire execution. To address these issues, we propose a PaaS-based elasticity model, named AutoElastic. It acts as a middleware that allows iterative HPC applications to take advantage of dynamic resource provisioning of cloud infrastructures without any major modification. AutoElastic offers elasticity automatically, where the user does not need to configure any resource management policy. This elastic mechanism includes using fixed thresholds as well as offering a new approach where it self adjusts the threshold values during the application execution. AutoElastic provides a new concept denoted here as asynchronous elasticity, i.e., it provides a framework to allow applications to either increase or decrease their computing resources without blocking the current execution. The feasibility of AutoElastic is demonstrated through a prototype that runs a CPU-bound numerical integration application on top of the OpenNebula middleware. Results with a parallel iterative application showed performance gains between 28.4% and 59% when comparing different executions enabling and disabling elasticity feature. In addition, tests with different parameters showed that when using threshold-rule based techniques with fixed thresholds, the upper threshold has a greater impact in performance and resource consumption than the lower threshold.

Keywords: Cloud Computing. Resource Elasticity. High Performance Computing.

LISTA DE FIGURAS

LISTA DE TABELAS

LISTA DE SIGLAS

ABNT	Associação Brasileira de Normas Técnicas
API	Application Program Interface
ARIMA	Autoregressive Integrated Moving Average
BOINC	Berkeley Open Infrastructure for Network Computing
BoT	Bag-of-Tasks
BSP	Bulk Synchronous Parallel
CAPES	Coordenação de Aperfeiçoamento de Pessoal de Nível Superior
CLI	Command Line Interface
CNPq	Conselho Nacional de Desenvolvimento Científico e Tecnológico
COS	Cloud Operating System
CPU	Central Processing Unit
FAPERGS	Fundação de Amparo à Pesquisa do Estado do Rio Grande do Sul
GUI	Graphic User Interface
HPC	High Performance Computing
IaaS	Infrastructure as a Service
MA	Moving Average
MEC	Ministério da Educação
MPI	Message Passing Interface
MPMD	Multiple Program Multiple Data
NERSC	National Energy Research Scientific Computing Center
NFS	Network File System
NIST	National Institute of Standards and Technology
PaaS	Platform as a Service
SaaS	Software as a Service
SALSA	Simple Actor Language System and Architecture
SES	Simple Exponential Smoothing
SLA	Service Level Agreement
SLO	Service Level Objectives
SSH	Secure Shell
VM	Virtual Machine

SUMÁRIO

1 INTRODUÇÃO

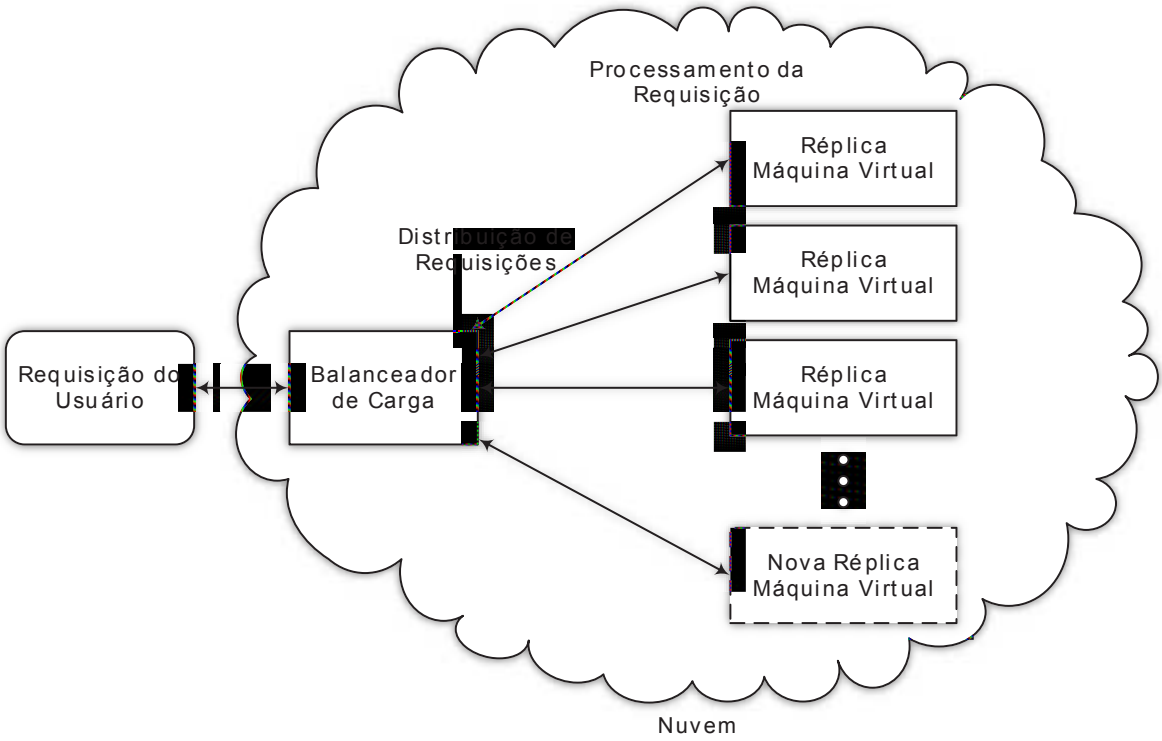
Uma das principais características relacionadas à Computação em Nuvem (MELL; GRANCE, 2011) é a elasticidade de recursos. Esta característica possibilita aos usuários alterar a capacidade da nuvem a qualquer momento aumentando ou diminuindo a quantidade de recursos de acordo com a demanda ou qualidade de serviço desejada (LORIDO-BOTRAN; MIGUEL-ALONSO; LOZANO, 2014; RAVEENDRAN; BICER; AGRAWAL, 2011). Através do princípio de provisionamento sob-demanda (YANG et al., 2015), o interesse na capacidade de elasticidade está relacionado aos benefícios que ela pode proporcionar, o que inclui melhorias no desempenho das aplicações, melhor utilização de recursos e redução de custos. Um outro fato que contribui para um melhor desempenho é a capacidade de alocação dinâmica de recursos computacionais adicionais. A possibilidade de alocar uma pequena quantidade recursos no início da aplicação, evitando provisionamento excessivo, além da possibilidade de liberá-los em períodos de carga moderada, enfatiza a justificativa para a redução de custos, o que impacta diretamente na economia de energia (SAH; JOSHI, 2014).

Considerando o cenário de Computação de Alto Desempenho, conhecida como *High Performance Computing* (HPC), e aplicações paralelas, um usuário pode querer aumentar a quantidade de recursos computacionais disponíveis para incrementar a capacidade de processamento da nuvem com o objetivo de reduzir o tempo total de execução da aplicação. Por outro lado, se a aplicação não está escalando de forma linear ou próxima a uma forma linear, e se o usuário é flexível com respeito ao tempo de execução da aplicação, a quantidade de recursos pode ser reduzida. Isso resulta em um valor menor de $recursos \times horas$, e assim reduz o custo e consumo de energia. Embora existam benefícios para sistemas HPC, a elasticidade em nuvem é mais largamente explorada em arquiteturas cliente-servidor, como vídeo sob-demanda, lojas online, aplicações BOINC (*Berkeley Open Infrastructure for Network Computing*) (ANDERSON, 2004), governança eletrônica e *Web Services* (RAVEENDRAN; BICER; AGRAWAL, 2011). Uma estratégia típica neste contexto, como ilustrado na Figura 1, é a elasticidade horizontal da infraestrutura de nuvem com a replicação de instâncias de máquinas virtuais (HAN et al., 2012; WARD; BARKER, 2014). Apesar de transparente para o usuário, este tipo de mecanismo é apropriado para aplicações fracamente acopladas em que réplicas não estabelecem comunicação entre si (GALANTE; BONA, 2012; JENNINGS; STADLER, 2014).

1.1 Motivação

Embora a solução anteriormente mencionada seja empregada com sucesso em aplicações baseadas em cliente-servidor e pertinente para aplicações *Bag-of-Tasks* (BoT) (GUTIERREZ-GARCIA; SIM, 2015), técnicas de replicação e balanceadores de carga centralizados não

Figura 1: Mecanismo de elasticidade de sistemas web: elasticidade horizontal e controlador de elasticidade agindo como balanceador de carga.



Fonte: elaborado pelo autor.

são úteis por padrão para implementar a elasticidade em aplicações científicas HPC fortemente acopladas, como aquelas modeladas como *Bulk-Synchronous Parallel* (BSP) (VALIANT, 1990), divisão-e-conquista ou pipeline (RAVEENDRAN; BICER; AGRAWAL, 2011; FRINCU; GENAUD; GOSSA, 2013). Geralmente, essas aplicações foram projetadas para usar um número fixo de recursos, e não podem explorar elasticidade sem um suporte adequado. Isso ocorre pois a adição ou remoção de instâncias causa uma reorganização de processos e também uma atualização de toda a topologia de comunicação e não somente a interação entre o balanceador de carga e as réplicas destino. Em outras palavras, a simples adição de instâncias e a utilização de balanceadores de carga não têm qualquer efeito nessas aplicações uma vez que eles não são capazes de detectar e utilizar esses recursos (COUTINHO; PAILLARD; SOUZA, 2014). Em adição, existe um problema relacionado à consolidação de máquinas virtuais, o que resulta no término repentino de um processo e sua desconexão da topologia de comunicação, consequentemente resultando no encerramento prematuro da aplicação.

Tecnicamente, nos últimos anos, muitas aplicações paralelas foram desenvolvidas utilizando o *Message Passing Interface* (MPI) 1.x, o qual não tem qualquer suporte para alterar o número de processos durante a execução (EXPóSITO et al., 2013). Desse modo, aplicações não podem explorar a elasticidade sem um suporte apropriado (WILKINSON; ALLEN, 2005). Embora esse cenário tenha mudado com a versão MPI 2.0, esse recurso

ainda não é suportado por muitas das implementações MPI disponíveis (RAVEENDRAN; BICER; AGRAWAL, 2011). A fim de tornar possível a execução dessas aplicações com o recurso de elasticidade, é necessário um esforço significativo em nível de aplicação, tanto para alterar manualmente o grupo de processos quanto para distribuir os dados entre diferentes quantidades de processos de maneira eficaz. A Figura 2 demonstra atuais abordagens para tornar isso possível. Na Figura 2 (a) o controle da elasticidade é implementado dentro do código da aplicação utilizando a API (*Application Program Interface*) suportada pela nuvem. Essa estratégia requer experiência do usuário em monitoramento da nuvem, além da seleção de pontos apropriados para a inserção das chamadas.

Por outro lado, como representado na Figura 2 (b), algumas abordagens exploram a utilização de um controlador de elasticidade externo à aplicação, o qual é normalmente oferecido como componente opcional de plataformas de nuvem, tais como Amazon e Microsoft Azure (ROLOFF et al., 2012). O monitoramento de recursos, assim como as operações de alocação e desalocação de recursos são tarefas que pertencem ao controlador, porém os usuários devem inserir chamadas de sistema em suas aplicações e tratar as reorganizações da topologia de comunicação. Uma vez que o controlador de elasticidade do lado da nuvem identifique a necessidade de adição de novos recursos, ele pode definir um parâmetro que pode ser lido pelo lado da aplicação informando que mais recursos estão disponíveis. Do lado da aplicação, ao verificar que a elasticidade foi realizada, os processos devem ser reorganizados para utilizarem os novos recursos disponíveis. A chamada do método *elasticidade()* representa uma ligação entre a aplicação e o controlador, dessa maneira o uso de um controlador sem esta ligação não tem efeito no balanceamento de carga tendo em vista que a aplicação não é capaz de detectar e se adaptar às reorganizações de recursos (COUTINHO; PAILLARD; SOUZA, 2014). Para superar essas limitações, diversos trabalhos empregam as diferentes abordagens apresentadas a seguir:

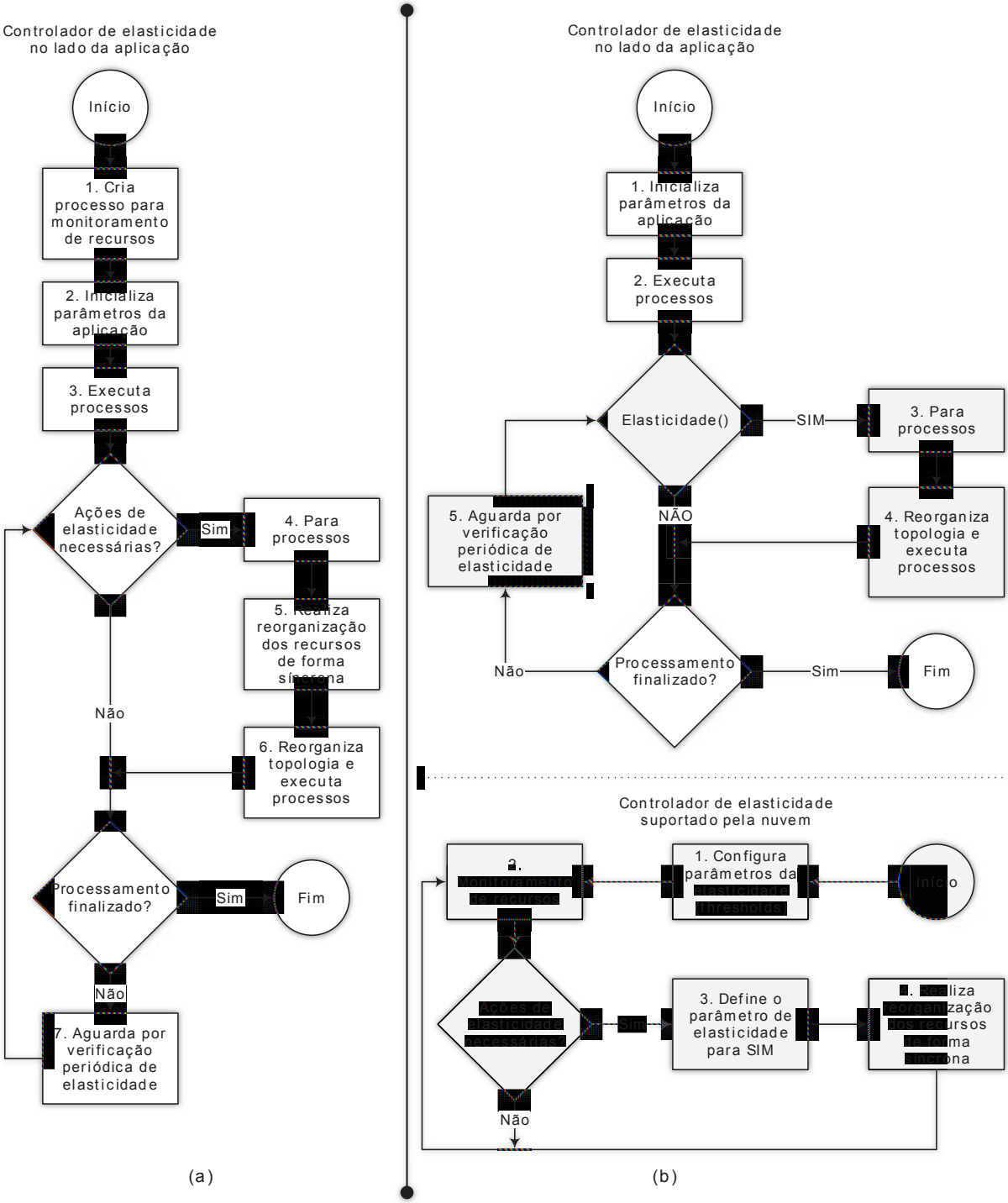
- Reescrita do código da aplicação (RAVEENDRAN; BICER; AGRAWAL, 2011; RAJAN et al., 2011);

- Configuração prévia de regras e ações de elasticidade (RAVEENDRAN; BICER; AGRAWAL, 2011; KNAUTH; FETZER, 2011; KUMAR et al., 2011; MICHON; GOSSA; GENAUD, 2012);

- Necessidade do conhecimento prévio das fases da aplicação (RAVEENDRAN; BICER; AGRAWAL, 2011; KNAUTH; FETZER, 2011; KUMAR et al., 2011; MICHON; GOSSA; GENAUD, 2012);

- Utilização de mecanismos conhecidos como *stop-reconfigure-and-go* (RAVEENDRAN; BICER; AGRAWAL, 2011).

Figura 2: Diferentes abordagens para elasticidade em nuvem: (a) ações de elasticidade são gerenciadas diretamente no código da aplicação; (b) uso de um controlador de elasticidade externo à aplicação, que pode oferecer concomitância entre ações de elasticidade e de processos da aplicação. Entretanto, essa estratégia não é transparente para os usuários, os quais precisam testar ações de elasticidade e reorganizar a topologia de comunicação por si mesmos. Estas ações estão representadas pelos blocos na cor cinza.



Fonte: elaborado pelo autor.

1.2 Questão de Pesquisa

A execução de aplicações HPC iterativas em ambientes de Computação em Nuvem enfrentam determinados problemas que podem afetar diretamente o desempenho da apli-

cação. Como principais problemas, pode-se citar:

- Pontos em que recursos estão sendo reorganizados requerem que a aplicação interrompa sua execução. Dessa maneira, durante o tempo em que a infraestrutura é reorganizada a aplicação permanece parada, impactando diretamente no seu desempenho;
- Estratégias de elasticidade atuais não lidam com eventuais picos e quedas repentinas na carga da métrica monitorada. Tais estratégias podem ocasionar ações falso-positivo e/ou falso-negativo de elasticidade, resultando em reorganizações de recursos desnecessárias.

Tendo em vista a dificuldade das abordagens atuais para oferecer elasticidade para aplicações HPC iterativas, o modelo desenvolvido nesta dissertação busca responder a seguinte questão de pesquisa:

Como seria um modelo de elasticidade em nuvem para aplicações HPC iterativas, transparente quanto a escrita da aplicação paralela, evitando custos elevados em termos de desempenho nos momentos de reconfiguração de recursos e processos?

Por natureza, aplicações HPC iterativas são estruturadas em laços iterativos (*loops*) em que o início de cada nova iteração representa um estado global consistente. Esta é uma suposição razoável para a maioria das aplicações MPI (HENDRICKSON, 2009; TAN et al., 2014), não limitando a aplicabilidade do modelo. Em termos de desempenho, sabe-se que aplicações HPC são sensíveis a quaisquer sobrecarga e que, muitas vezes, otimizações de milissegundos são decisivas para taxar uma aplicação como lenta ou rápida. A ideia é fazer com que a elasticidade seja realizada sem causar impacto proibitivo na execução da aplicação. Para tal, é necessário explorar o mecanismo da aplicação, a infraestrutura de nuvem e o mecanismo de elasticidade. Ainda, quanto a escrita da aplicação, o usuário não desenvolve sua aplicação pensando em elasticidade e, mais ainda, ele não insere nenhuma diretiva, chamada de sistema ou de elasticidade em seu código. A ideia é que o usuário tenha um código pronto e queira executá-lo com maior desempenho.

1.3 Objetivos

Buscando proporcionar elasticidade em nuvem para aplicações HPC de maneira eficiente e transparente, esta dissertação propõe o modelo de elasticidade chamado AutoElastic. AutoElastic introduz o conceito de elasticidade assíncrona: reorganização de recursos e processos na perspectiva dos usuários, não bloqueando e nem finalizando a execução da aplicação durante as operações de alocação ou desalocação de recursos. Para tal, AutoElastic fornece um *framework* com um controlador que transparentemente gerencia

operações de elasticidade horizontal, sendo assim, sem a necessidade de modificações ou adaptações na aplicação. Levando em consideração como ponto inicial a Figura 2 (b), AutoElastic oferece um *framework* para esconder do usuário todos os blocos representados na cor cinza. Embora o uso padrão de um controlador permita alocações de máquinas virtuais em paralelo com a execução da aplicação, os benefícios dos novos recursos não são transparentes para os usuários. Como discutido anteriormente, operações de adição ou remoção de máquinas virtuais também surgem como um problema para o uso padrão de um controlador, pois a remoção de uma ou mais máquinas virtuais finalizará repentinamente os processos executando nelas, o que pode implicar no final prematuro da aplicação. Tendo em vista essas questões, este trabalho possui como objetivo principal:

Desenvolver um modelo de ***elasticidade*** *automática,* ***transparente*** *e* ***não bloqueante*** *para aplicações* ***HPC*** *iterativas em ambientes de computação em nuvem sem a necessidade de intervenção do usuário.*

Neste contexto, elasticidade automática refere-se à capacidade de o ambiente de nuvem alterar a quantidade de recursos durante a execução da aplicação sem a necessidade de intervenção do usuário. Além disso, o conceito de transparência diz respeito à realização das operações de elasticidade ocorrerem sem interferência na execução da aplicação e sem a necessidade dela realizar tais operações. Ainda, o fato da elasticidade não ser bloqueante impede que a aplicação em execução seja interrompida enquanto operações de elasticidade estiverem ocorrendo. Por fim, o foco principal do modelo é suportar aplicações de HPC iterativas. Para atingir o objetivo principal, foram definidos os seguintes objetivos específicos:

(i) Através da análise do estado da arte, identificar lacunas referentes às estratégias de elasticidade em nuvem adotadas;

(ii) Desenvolver um modelo de elasticidade que contemple lacunas encontradas e desenvolver um protótipo deste modelo para a realização de testes;

(iii) Desenvolver uma aplicação de HPC iterativa para a execução e simulação em nuvem de diferentes comportamentos de carga modelados;

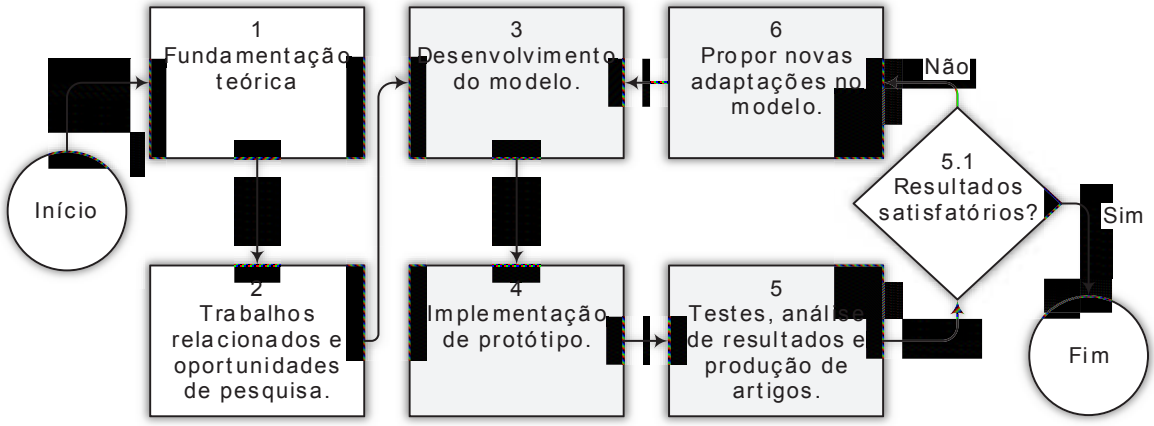
(iv) Realizar testes em laboratório com a aplicação desenvolvida, confrontando diferentes comportamentos de carga modelados com diferentes configurações de elasticidade;

(v) Analisar os resultados obtidos com a perspectiva de desempenho, consumo de energia e custo, e comparar estes resultados com as diferentes configurações de *thresholds* de elasticidade.

1.4 Plano de Pesquisa

O desenvolvimento da pesquisa foi dividido em seis etapas, das quais quatro delas formaram um ciclo que ocorreu mais de uma vez, como apresentado na Figura 3. Inicialmente, realizou-se o estudo das teorias envolvidas no tema da pesquisa para formar o referencial teórico. Após isso, iniciou-se a etapa de levantamento de trabalhos relacionados com o tema da pesquisa, mais especificamente com os objetivos apresentados. Ao final desta etapa, realizou-se uma análise com o objetivo de identificar as lacunas presentes no estado da arte. Em seguida, na terceira etapa, o modelo foi proposto e desenvolvido com foco em atender os objetivos propostos e responder a questão de pesquisa. Ainda, com as lacunas levantadas na etapa anterior, o desenvolvimento do modelo incluiu pontos fracos identificados. Em seguida, as implementações de protótipos foram realizadas na quarta etapa, na qual a parte técnica da pesquisa foi realizada. Com o desenvolvimento do modelo e dos protótipos, avançou-se para a quinta etapa, em que os testes do modelo foram realizados, incluindo a análise de resultados. A escrita de artigos baseando-se nas análises realizadas foi uma tarefa desta etapa também. Em seguida, ofereceu-se a possibilidade de finalizar ou propor novas adaptações no modelo. Dessa maneira, ao invés de finalizar a pesquisa, seguiu-se para a sexta etapa, em que melhorias e adaptações no modelo foram propostas, baseadas nas análises de resultados realizadas na etapa anterior. Depois, retornou-se para a etapa de desenvolvimento do modelo a fim de realizar modificações e adaptações propostas. Dessa maneira, as etapas 3, 4, 5 e 6 formaram um ciclo em que puderam ser desenvolvidas diferentes abordagens e produzidos diferentes trabalhos. Ao chegar novamente na quinta etapa, novos artigos foram produzidos com as novas avaliações e a pesquisa encontrou seu final, originando esta dissertação.

Figura 3: Sequência de etapas do desenvolvimento da pesquisa. As etapas representadas na cor cinza ocorreram mais de uma vez, fechando um ciclo a fim de contemplar adaptações e novas implementações.



Fonte: elaborado pelo autor.

1.5 Organização do Texto

Esta dissertação está estruturada em sete capítulos. Após a introdução apresentada no Capítulo 1, os conceitos relacionados a esta dissertação são apresentados no Capítulo 2, introduzindo tecnologias e modelos de programação voltados para aplicações HPC. Em seguida, no Capítulo 3 são discutidos os trabalhos relacionados de maneira a apresentar o estado da arte envolvendo elasticidade em nuvem e aplicações HPC.

O Capítulo 4 apresenta o modelo desenvolvido para tratamento de elasticidade em nuvem para aplicações HPC iterativas. Esse modelo é o principal foco desta dissertação e neste capítulo são apresentados os algoritmos e técnicas desenvolvidas, além de decisões de projeto. No Capítulo 5 é apresentada a metodologia para a avaliação do modelo, no qual são expostos detalhes de implementação e parâmetros de testes. Já a avaliação do modelo é apresentada no Capítulo 6 através da análise de resultados obtidos em laboratório. Por fim, no Capítulo 7 são apontadas conclusões sobre o trabalho, enfatizando as contribuições do modelo e os resultados obtidos.

2 FUNDAMENTAÇÃO TEÓRICA

Este capítulo aborda os principais conceitos relacionados à Computação em Nuvem, baseados no estado atual de conhecimento da área. Além disso, são apresentados também conceitos necessários para o entendimento das decisões de projetos realizadas no modelo desenvolvido. A seção 2.1 apresenta os conceitos relacionados à Computação em Nuvem. Além destes conceitos, também são apresentados conceitos relacionados a modelos de programação paralela na Seção 2.2. Algumas estratégias de elasticidade requerem que sejam realizas previsões de valores futuros que podem direcionar tomadas de decisão relacionadas à manutenção do ambiente de nuvem no qual uma aplicação é executada. Tendo isso em vista, na Seção 2.3 são apresentados conceitos relacionados a utilização de séries temporais para análise de desempenho. Por fim, são realizadas considerações parciais sobre os assuntos tratados neste capítulo na Seção 2.4.

2.1 Computação em Nuvem

Computação em Nuvem é o termo utilizado para definir o modelo de computação em que recursos computacionais podem ser acessados através da Internet (RAJARAMAN, 2014). Em modelos tradicionais de computação o processamento de operações, seja em computadores pessoais ou em grandes servidores, é realizado localmente utilizando os recursos dos equipamentos disponíveis. Com o surgimento da Computação em Nuvem, essas tarefas passaram a ser executadas não mais localmente, mas sim em servidores acessados através da Internet, sem a localização ser conhecida (HAYES, 2008). Este novo modelo permite que usuários tenham acesso a recursos de *hardware* e *software* através da Internet sem a necessidade de comprar equipamentos, pagando apenas pelo uso dos recursos (RAJARAMAN, 2014). Uma das principais características da Computação em Nuvem que pode ser destacada é a elasticidade de recursos, em que recursos podem ser alocados ou liberados sob demanda de acordo com as necessidades do usuário, que paga apenas pelos recursos utilizados (RAJARAMAN, 2014).

Ainda, o *National Institute of Standards and Technology* (NIST) (MELL; GRANCE, 2011) define a Computação em Nuvem como um modelo que permite, de maneira conveniente e sob demanda, o acesso através de rede de computadores a um conjunto configurável de recursos computacionais (redes, servidores, armazenamento, aplicações e serviços) que podem rapidamente ser provisionados e liberados com esforço mínimo de gerenciamento ou iteração do provedor de serviço. Segundo o NIST, este modelo de computação promove disponibilidade e é composto por (MELL; GRANCE, 2011):

- Cinco características essenciais: Auto-previsionamento sob demanda, acesso amplo a rede, agrupamento de recursos, rápida elasticidade e medição de serviço;

- Três modelos de serviço: Infraestrutura como Serviço ou *Infrastructure as a Service* (IaaS), Plataforma como Serviço ou *Platform as a Service* (PaaS) e Software como Serviço ou *Software as a Service* (SaaS);

- Quatro modelos de implantação: Nuvem Privada, Nuvem Pública, Nuvem Híbrida e Nuvem Comunitária.

2.1.1 Modelos de Serviço

Em Computação em Nuvem, os recursos oferecidos podem ser desde recursos de *hardware* como memória e unidades de processamento, amplamente chamadas de CPU (*Central Processing Unit*), quanto recursos de *software*. Os modelos de serviço oferecidos por uma nuvem variam de acordo com o tipo de recursos que são disponibilizados (PUTHAL et al., 2015). Estes modelos são classificados em três categorias: (i) Infraestrutura como Serviço (IaaS); (ii) Plataforma como Serviço (PaaS); e (iii) Software como Serviço (SaaS) (MELL; GRANCE, 2011; RAJARAMAN, 2014; PUTHAL et al., 2015).

- **IaaS**: Neste modelo, são oferecidos ao usuário recursos fundamentais de computação como CPU, memória, armazenamento e rede, permitindo ao usuário a implantação de *softwares* arbitrários. Através de um *hypervisor* é realizada a virtualização dos recursos computacionais e disponibilizados ao usuário através de máquinas virtuais. A virtualização de recursos é a característica chave deste modelo permitindo ao usuário executar seu próprio sistema operacional e gerenciar máquinas virtuais que rodam em cima da infraestrutura oferecida;

- **PaaS**: Este modelo é uma camada imediatamente superior ao modelo IaaS. Aqui, o usuário não mais tem o controle do sistema operacional e *softwares* instalados nas máquinas virtuais. Essa tarefa é de responsabilidade do provedor da nuvem, que fica encarregado de administrar, além da infraestrutura, as máquinas virtuais. Deste modo, é oferecido ao usuário um conjunto de recursos como sistema operacional, linguagens de programação, plataformas para desenvolvimento de *softwares* e ferramentas para implantação de aplicações. O usuário deste modelo tem acesso a estas plataformas e ferramentas com o objetivo do desenvolvimento e implantação de aplicações;

- **SaaS**: Por fim, no topo da pilha de modelos de serviço, este modelo é o mais comum visto hoje em dia em utilização. Neste modelo, é oferecido ao usuário o acesso a aplicações que executam sobre uma infraestrutura de nuvem. Aplicações deste modelo podem ser acessadas de diversos dispositivos diferentes através de diferentes interfaces. Outra característica é que uma aplicação pode ser acessada simultaneamente por diversos usuários. O usuário deste modelo utiliza apenas a

aplicação, sem se envolver com a infraestrutura e os serviços necessários para que a aplicação execute. A administração de toda a infraestrutura é de responsabilidade do provedor.

2.1.2 Modelos de Implantação

Uma infraestrutura de nuvem pode ser mantida de diferentes formas. Isso pode ser realizado pela própria organização ou ser contratada através de um provedor de nuvem. Ainda, uma combinação entre diferentes modos pode ser realizada. O modo como uma infraestrutura de nuvem é disponibilizada e mantida nos leva a quatro modelos de implantação diferentes (PUTHAL et al., 2015): (i) Nuvem Privada; (ii) Nuvem Pública; (iii) Nuvem Comunitária; e (iv) Nuvem Híbrida. A seguir cada um dos modelos é detalhado (MELL; GRANCE, 2011; RAJARAMAN, 2014; PUTHAL et al., 2015):

- **Nuvem Privada**: Em uma nuvem privada, toda a infraestrutura de computação é disponibilizada para uso exclusivo de uma única organização. Neste modelo, a organização pode ser dona da infraestrutura ou também terceirizá-la, porém toda a infraestrutura é alocada apenas para a organização, sem haver o compartilhamento dos recursos com outros usuários;

- **Nuvem Pública**: Neste modelo, a infraestrutura é provisionada para o uso de qualquer usuário. Toda a infraestrutura é compartilhada por diferentes usuários, podendo ser cobrada ou de graça. Neste modelo, existe a figura de um provedor que mantém a infraestrutura e oferece diferentes serviços para diferentes usuários;

- **Nuvem Comunitária**: Neste modelo, a infraestrutura é provisionada exclusivamente para um grupo específico de usuários que compartilham os mesmos requisitos e interesses. Como exemplo, podemos citar um grupo de universidades que mantém suas nuvens privadas e decidem uni-las em uma única nuvem comunitária;

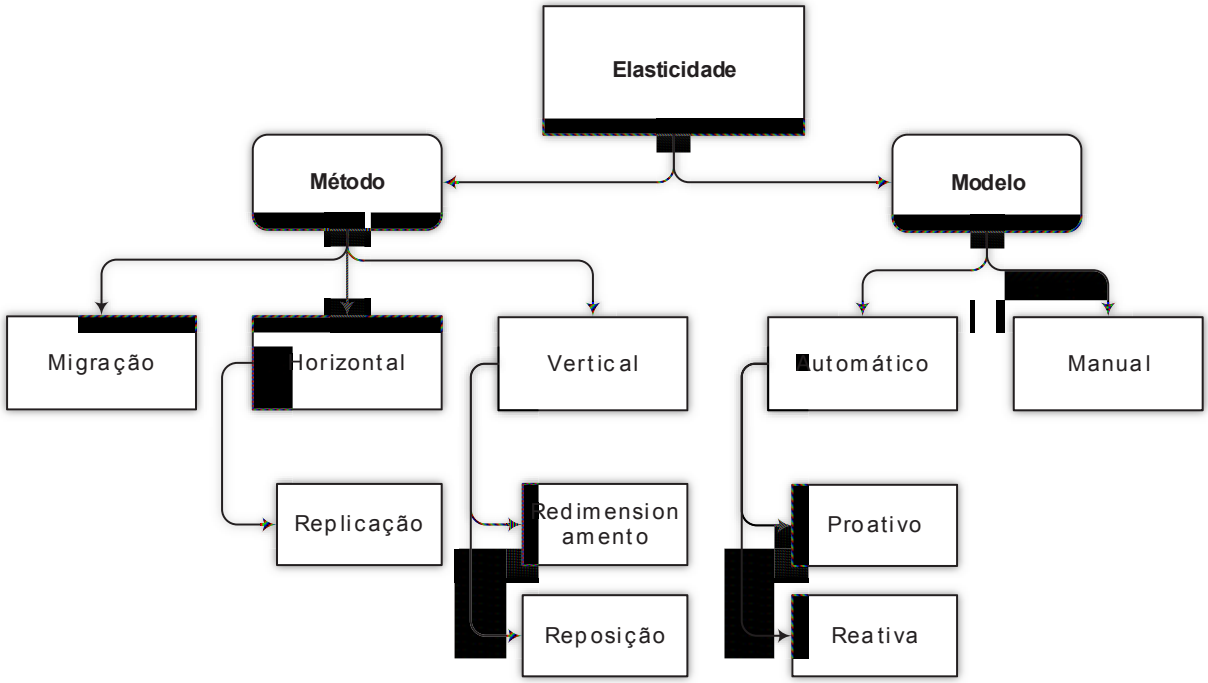
- **Nuvem Híbrida**: Por fim, uma nuvem híbrida é formada pela combinação de duas ou mais diferentes infraestruturas dos demais modelos de implantação. As infraestruturas que compõem uma nuvem híbrida continuam distintas, porém elas são unidas por protocolos específicos que permitem a portabilidade da aplicação entre uma infraestrutura e outra.

2.1.3 Elasticidade

Uma das funcionalidades mais importantes da Computação em Nuvem é a capacidade de provisionamento de recursos em tempo de execução. Esta capacidade é chamada de elasticidade de recursos, em que é possível aumentar ou diminuir a quantidade de recursos

disponíveis sem a interrupção do serviço para atender variações de demanda (COUTINHO et al., 2014; LEHRIG; EIKERLING; BECKER, 2015). Existem diferentes métodos para oferecer a elasticidade, cada um definindo diferentes operações para adição ou remoção de recursos. Além dos métodos, existem diferentes modelos utilizados, que definem a maneira como as operações são realizadas. A Figura 4 apresenta os diferentes métodos e modelos de elasticidade, bem como as operações que podem ser realizadas.

Figura 4: Métodos e modelos de elasticidade de recursos. O modelo apresenta como operações podem ser disparadas enquanto o método apresenta os tipos de operações que podem ser realizadas.



Fonte: adaptado de Galante e Bona (2012); Coutinho et al. (2014).

Considerando o método de realização da elasticidade, conforme Galante e Bona (2012), existem três métodos diferentes para oferecer a elasticidade: horizontal, vertical e migração. A seguir cada um dos métodos é detalhado (GALANTE; BONA, 2012; COUTINHO et al., 2014):

- Elasticidade **horizontal**: Consiste em adicionar ou remover instâncias de componentes do ambiente virtual do usuário. Essas instâncias podem ser máquinas virtuais, contêineres, ou módulos de aplicações que são replicados no ambiente de nuvem e disponibilizados para o usuário. O método mais utilizado atualmente para oferecer a elasticidade horizontal é através da **replicação** de máquinas virtuais;

- Elasticidade **vertical**: Consiste em adicionar ou remover recursos de processamento, memória, rede e armazenamento das instâncias virtuais em execução. Neste caso, máquinas virtuais têm seus atributos alterados podendo ter seus recursos físicos aumentados ou diminuídos. Existem duas técnicas principais de elasticidade vertical:

redimensionamento e **reposição**. A primeira, e mais comumente utilizada, consiste em alterar a quantidade de recursos físicos disponíveis para a máquina virtual. Porém, atualmente poucos sistemas operacionais suportam que quantidade de recursos seja alterada sem a necessidade de reinicialização da máquina virtual. Quanto a técnica de reposição, esta consiste em substituir um recurso físico por outro com maior ou menor poder computacional. Neste caso um servidor que comporta máquinas virtuais pode ser substituído por um mais ou menos potente dependendo da necessidade;

- **Migração**: recursos como máquinas virtuais e aplicações podem ser migrados entre os servidores que compõe a infraestrutura da nuvem. Essa técnica pode ser utilizada em substituição a técnica de reposição. Através de migrações, máquinas virtuais podem ser transferidas de uma máquina para outra com maior ou menor poder computacional. Além disso, migrações são pertinentes para a consolidação de servidores visando diminuir o consumo energético com menos equipamentos ligados.

Os diferentes métodos de elasticidade são operações que podem ser realizadas de maneiras diferentes. Galante e Bona (2012) apresentam políticas que definem dois modelos de elasticidade diferentes: **manual** e **automático**. Esses modelos são classificados conforme as interações necessárias dos usuários para a execução das operações de elasticidade (GALANTE; BONA, 2012).

No modelo manual, o usuário realiza todas as operações de gerenciamento de recursos manualmente através de ferramentas de linha de comando, interface gráfica ou API's disponibilizadas pelo provedor ou sistema de nuvem. Por outro lado, no modelo automático, a elasticidade é realizada através de ferramentas de monitoramento do sistema de nuvem ou por alguma aplicação específica. Neste caso, decisões de elasticidade são tomadas através de algoritmos de avaliação de métricas monitoradas. Apesar de ser chamado automático, o usuário ainda tem a tarefa de configurar manualmente as métricas e os algoritmos de avaliação que serão utilizados (GALANTE; BONA, 2012).

As operações realizadas no modelo automático podem ser classificadas como **reativa** ou **proativa**, dependendo do algoritmo utilizado (GALANTE; BONA, 2012; COUTINHO et al., 2014). Na forma reativa, as operações de elasticidade são tomadas baseando-se nas métricas atuais coletadas do ambiente. Comumente são utilizados limites de ocupação dos recursos (chamados *thresholds*), tanto inferior quanto superior, que servem para disparar ações em caso de os dados coletados informarem a violação de algum desses *thresholds* (GALANTE; BONA, 2012). Ainda, os dados coletados podem ser confrontados com algum SLA (*Service Level Agreement*) pra verificação se algum requisito está sendo violado (BASET, 2012). Por outro lado, na forma proativa, os dados coletados são utilizados para a realização de previsões de futuros eventos. Essas previsões são levadas em consideração para a tomada de decisão e a realização ou não de operações de elasticidade.

Técnicas de previsão de carga dos recursos são comumente utilizadas para a reorganização proativa dos recursos (GALANTE; BONA, 2012; COUTINHO et al., 2014).

2.2 Modelos de Programação Paralela

Anteriormente ao surgimento da Computação em Nuvem, em que aplicações eram executadas em apenas uma única máquina, a solução para se obter um alto grau de desempenho de determinadas aplicações era através de supercomputadores (DOWD; SEVERANCE, 1998). Com o passar do tempo, uma crescente evolução dos processadores de computadores pessoais tornou possível superar a capacidade de processamento de supercomputadores através da conexão de diversos computadores pessoais realizando a computação em conjunto (DOWD; SEVERANCE, 1998; BUYYA, 1999). Com esse avanço, surgiu o conceito de computação paralela (ALMASI; GOTTLIEB, 1989), que é atualmente a técnica utilizada para se obter a computação de alto desempenho. O avanço dos processadores permitiu que um único processador fosse capaz de comportar mais de uma CPU. Essas novas tecnologias possibilitaram que aplicações fossem desenvolvidas para executar de forma paralela em diversas CPU's realizando comunicação entre si através da passagem de mensagens (DOWD; SEVERANCE, 1998). O modo de processamento paralelo pode ocorrer dentro de um mesmo processador com diversas CPU's ou também em diversos computadores interconectados. Ainda, uma união dos dois modos pode ocorrer em um cenário em que diversos computadores interconectados dispões de processadores com mais de uma CPU. Considerando o desenvolvimento de aplicações, os programadores necessitam estar cientes da tecnologia para a qual estão programando com o objetivo de criar aplicações paralelas. Desse modo, surgiram novos modelos de processamento paralelo que, conforme seu modo de execução, foram classificados como: Mestre-Escravo, Pipeline, Divisão-e-Conquista e BSP (VALIANT, 1990; BUYYA, 1999).

Aplicações escritas no modelo Mestre-Escravo consistem em um processo mestre, responsável por dividir a tarefa a ser executada em diversas tarefas menores e distribuí-las entre diversos processos escravos (BUYYA, 1999). Por sua vez, os processos escravos são responsáveis por receber e executar as tarefas enviadas pelo processo mestre, retornando o resultado da computação. A comunicação neste paradigma ocorre apenas entre o mestre e os escravos, não havendo comunicação entre os escravos. Apesar de atingir interessantes graus de escalabilidade, este modelo possui um importante ponto fraco pois o processo que realiza a coordenação da execução é centralizado (BUYYA, 1999). Isso pode se tornar um gargalo para a aplicação em situações em que existem muitos escravos em execução, tornando custosa as tarefas de comunicação do mestre com os escravos.

Por outro lado, aplicações Pipeline utilizam uma abordagem de decomposição funcional em que as tarefas da aplicação são identificadas e cada processo executa uma tarefa específica (BUYYA, 1999). Neste modelo, o algoritmo é dividido em diversas partes me-

nores que podem ser executadas separadamente. Cada tarefa é realizada por um processo diferente. Os processos são organizados em estágios de maneira a formar um fluxo sequencial. Os dados processados no fluxo de execução passam de um estágio para o outro e desta maneira a comunicação entre processos ocorre apenas entre processos adjacentes. Devido a organização sequencial das tarefas, a eficiência deste paradigma está diretamente relacionada a habilidade de balanceamento da carga entre os estágios do Pipeline (BUYYA, 1999). Deste modo, um estágio mais lento pode comprometer o restante da execução deixando os estágios seguintes ociosos.

Sob o ponto de vista de aplicações do modelo Divisão-e-Conquista, um problema é dividido em dois ou mais subproblemas que são distribuídos entre processos diferentes (BUYYA, 1999). Cada subproblema é resolvido independentemente e os resultados são combinados, necessitando de um processamento adicional para que o resultado combinado e final seja obtido. Considerando que cada problema é independente, é possível que eles sejam computados simultaneamente por diferentes processos. Outra característica do modelo é que não há necessidade de comunicação entre os processos que estejam computando subproblemas distintos (BUYYA, 1999). As comunicações entre processos são necessárias apenas nas fases de divisão dos problemas e combinação dos resultados. Dadas as funções desempenhadas no modelo, são realizadas três operações neste paradigma: divisão, computação e combinação (BUYYA, 1999). Ainda, neste modelo, a topologia dos processos é realizada em forma de árvore. Dessa maneira, cada processo divide suas tarefas com seus processos filhos. A computação das tarefas em si é realizada pelos processos folha dessa árvore. Esse formato é semelhante ao paradigma Mestre-Escravo, o qual possui um único processo que faz a divisão das tarefas entre seus diversos filhos, realizando a combinação deles no final do processamento (BUYYA, 1999).

Por fim, o modelo BSP foi introduzido por Valiant (1990) como um modelo de programação paralela com o objetivo de realizar uma ponte entre *hardware* e *software* (VALIANT, 1990). O modelo é composto pela combinação de três atributos: computação, comunicação e barreiras. Cada componente pode realizar funções de processamento e/ou armazenamento. As mensagens entre pares de componentes são entregues pelo roteador. Neste modelo, a computação dos dados é realizadas por fases, chamadas *supersteps*, em que há uma sincronização entre os processos ao final de cada fase (VALIANT, 1990). Uma aplicação organizada no modelo BSP consiste em diversos processos distribuídos que realizam comunicação através da rede. O processamento através de fases consiste em computações realizadas pelos processos dentro de um intervalo de uma janela de computação. Cada processo realiza computações locais e ao término inicia o processo de sincronização. Todos os processos devem realizar essa sincronização ao final de suas computações, finalizando dessa maneira uma *superstep* (VALIANT, 1990). Toda a computação realizada no modelo BSP é realizada através de diversas *supersteps*, em o final de cada uma garante um estado global consistente da computação. A este momento de sincronização é dado o

nome de barreira de sincronização, em que todos os processos aguardam o início da próxima *superstep* para dar continuidade ao processamento. Para que uma *superstep* possa ser finalizada é necessário que todos os processos realizem a sincronização, garantindo o estado global consistente, e após isso darem início a próxima *superstep* (VALIANT, 1990).

2.3 Análise de Desempenho Usando Séries Temporais

Devido a adoção da Computação em Nuvem como alternativa para a execução de aplicações HPC iterativas, diversas estratégias para oferecer elasticidade são oferecidas pelas plataformas de nuvem e pesquisas acadêmicas. Destas estratégias, muitas utilizam técnicas de monitoramento que compreendem previsão de valores futuros e cálculos de suavização de valores. Análise de séries temporais oferecem diversos métodos para calcular a previsão de valores de carga de processamento baseando-se em históricos de monitoramento. Estes métodos são relevantes em cenários em que tomadas de decisões são realizadas considerando previsões futuras, o que pode impactar diretamente na qualidade de serviço e desempenho de um determinado serviço. Cada método possui diferentes características e parâmetros. Tais diferenças definem a complexidade e objetivo de cada um dos métodos. Um dos modelos mais utilizados para previsão de séries temporais é o modelo ARIMA (*Autoregressive Integrated Moving Average*) (BOX; JENKINS; REINSEL, 2008), que oferece diversos métodos de previsão baseando-se em médias móveis. A Tabela 1 apresenta as abordagens mais comuns baseadas na análise de séries temporais conforme apresentado por Herbst et al. (2013).

Tabela 1: Métodos de séries temporais baseados no modelo ARIMA

Método	Descrição	Observações
Naive Forecast $\cong$ ARIMA 100	Considera apenas o valor da observação mais recente, assumindo que esse valor possui a maior probabilidade de ocorrer na próxima observação.	1
Moving Average (MA) $\cong$ ARIMA 001	Calcula a média considerando uma janela das últimas x observações mais recentes.	≥ 2
Simple Exponential Smoothing (SES) $\cong$ ARIMA 022	Generalização de MA utilizando pesos de acordo com uma função exponencial que dá maior peso para os valores das observações mais recentes.	> 5
Cubic Smoothing Splines (CS)	Intervalos de predição são construídos através da utilização de uma abordagem de probabilidade para estimar os parâmetros de alisamento. O método pode ser mapeado para um modelo de processo estocástico ARIMA 022 com um parâmetro espaço restrito.	> 5
ARIMA 101 auto-regressive integrated moving averages	ARIMA instância do modelo de processo estocástico parametrizado com p = 1 como ordem do processo AR(p) , d = 0 como ordem de integração, q = 1 como ordem do processo MA(q). Neste caso um processo estocástico estacionário é assumido (sem integração) e não sazonalidade considerada.	> 5
ARIMA	O processo de seleção do modelo automatizado ARIMA do pacote R de previsão começa com uma estimativa complexa de um modelo ARIMA(p,d,q)(P,D,Q)m apropriado usando testes de raiz unitária e critérios de informação (como o AIC) em combinação com um procedimento passo a passo para percorrer um espaço de modelo relevante. O modelo ARIMA selecionado é então ajustado aos dados para a previsão dos pontos e intervalos de confiança.	Mínimo 3 períodos de dados

Fonte: adaptado de Herbst et al. (2013).

Dos métodos apresentados, é importante destacar a quantidade necessária de obser-

vações de valores para o cálculo. A maioria necessita de até no mínimo 5 valores, sendo alguns casos necessários apenas 1 ou 2 valores. Em particular, os métodos *Moving Average* (MA) e *Simple Exponential Smoothing* (SES) são utilizados para a suavização de valores (HERBST et al., 2013). Porém, a grande diferença está na quantidade de valores considerados para o cálculo final. Enquanto em MA apenas a quantidades de valores da janela é considerado para o cálculo, em SES todos os valores coletados são considerados (HERBST et al., 2013). Neste último, quanto mais antigo o valor coletado menor é seu peso para o cálculo final. Em adição, para esta técnica o valor mais recente recebe o peso 0,5, o segundo mais recente 0,25, e assim por diante. Diante disso, é realizada uma suavização exponencial do valor, diminuindo o impacto de ruídos na lista de valores.

2.4 Considerações Parciais

Este capítulo abordou os principais conceitos envolvidos no tema da presente dissertação. Foram apresentados conceitos relacionados à Computação em Nuvem como características e funcionalidades a fim de oferecer um maior entendimento desta tecnologia. Ainda sobre Computação em Nuvem, estes conceitos são necessários para um maior entendimento do contexto em que foi realizado o estudo do modelo desenvolvido. Além desta tecnologia, foram apresentados conceitos de modelos de programação paralela, os quais são utilizados para o desenvolvimento de aplicações HPC. Este conceitos são pertinentes para um melhor entendimento de como são estruturadas aplicações HPC com o intuito de vislumbrar tais aplicações em ambientes de Computação em Nuvem. Uma questão importante tratada nesta dissertação está relacionada à tomada de decisão para oferecer as funcionalidades do modelo baseando-se no monitoramento de aplicações HPC executadas em ambientes de nuvem. Considerando este cenário, foram apresentados conceitos de previsão de valores através de séries temporais. Estes métodos podem ser utilizados para previsões de valores futuros e suavização de ruídos em valores, os quais podem direcionar tomadas de decisão relacionadas à manutenção da nuvem na qual uma aplicação é executada. O modelo apresentado nesta dissertação está inserido dentro do contexto das tecnologias, modelos e técnicas descritas neste capítulo. No próximo capítulo será apresentada uma análise do estado da arte considerando as técnicas de elasticidade e seus objetivos empregados por provedores de nuvem e pesquisas acadêmicas.

3 TRABALHOS RELACIONADOS

Este capítulo está organizado de forma a apresentar a perspectiva da elasticidade em nuvem, tanto para provedores de nuvem pública quanto para iniciativas de pesquisa acadêmica, demonstrando a lacuna no estado da arte referente ao gerenciamento de elasticidade para aplicações HPC. A Seção 3.1 apresenta a metodologia de pesquisa utilizada para a escolha dos trabalhos relacionados, bem como os termos utilizados para a realização da pesquisa. Em seguida, são apresentados os trabalhos relacionados referentes a provedores de nuvem pública e iniciativas de pesquisa acadêmica, respectivamente nas Seções 3.2 e 3.3. Provedores de nuvem disponibilizam soluções de nuvem robustas prontas para uso dos usuários, enquanto pesquisas acadêmicas apresentam *frameworks*, *plugins* e/ou novos algoritmos para o problema de elasticidade em nuvem. Após isso, a Seção 3.4 apresenta uma análise geral dos pontos fracos dos trabalhos apresentados. Por fim, na Seção 3.5 são realizadas algumas considerações parciais referentes a este capítulo.

3.1 Metodologia de Pesquisa e Escolha dos Trabalhos Relacionados

A pesquisa por trabalhos relacionados foi realizada utilizando o Portal de Periódicos CAPES/MEC[1]. Segundo o site, o Portal de Periódicos é uma biblioteca virtual que reúne e disponibiliza a instituições de ensino e pesquisa no Brasil o melhor da produção científica internacional. Ele conta com um acervo de mais de 37 mil títulos com texto completo, 126 bases referenciais, 11 bases dedicadas exclusivamente a patentes, além de livros, enciclopédias e obras de referência, normas técnicas, estatísticas e conteúdo audiovisual.

Para a realização da pesquisa, os principais termos utilizados foram "*cloud computing high performance computing*", "*cloud computing elasticity*", "*cloud computing autoscaling*", "*elascitity high performance computing*" e "*autoscaling high performance computing*". Os resultados das pesquisas foram analisados com o objetivo de encontrar trabalhos que explorassem a elasticidade em ambientes de Computação em Nuvem, bem como a utilização de aplicações HPC em tais ambientes. Nesta dissertação, o termo elasticidade se refere a ações de adicionar ou remover recursos do ambiente de nuvem do usuário durante o tempo de execução (GALANTE; BONA, 2012). Os trabalhos considerados foram divididos em dois grupos diferentes: iniciativas comerciais e de código aberto (Seção 3.2) e iniciativas de pesquisa acadêmica (Seção 3.3). Os trabalhos referentes a cada um dos grupos foram reunidos em duas tabelas elencando as principais características consideradas. Considerando as Tabelas 2 e 3, o método define como e quais tipos de recursos serão tratados (SHARMA et al., 2011), enquanto o modelo define o tipo de elasticidade quanto ao envolvimento do usuário para a realização das operações (GALANTE; BONA, 2012). Além disso, são elencados os modelos de serviço adotados por cada abordagem,

[1]http://periodicos.capes.gov.br.ez101.periodicos.capes.gov.br

as métricas avaliadas e operações de elasticidade realizadas. Ainda, são apresentadas as interfaces disponibilizadas pelas abordagens para o gerenciamento do ambiente e elasticidade. A análise dos trabalhos consistiu na avaliação das seguintes características, conforme apresentadas nas Tabelas 2 e 3:

- **Serviço**: refere-se ao modelo de serviço adotado, considerando os tipos de recursos em nuvem oferecidos;
- **Método**: refere-se ao método de elasticidade, apresentando como novos recursos são adicionados ou removidos quando operações de elasticidade são necessárias;
- **Modelo**: refere-se ao modelo de elasticidade, considerando a maneira como são executadas as operações do método de elasticidade;
- **Métricas**: atributos e valores que são monitorados para a tomada de decisão de operações de elasticidade;
- **Operações**: define as técnicas e operações envolvidas para a realização da elasticidade relacionadas ao método de elasticidade;
- **Interface**: recursos disponibilizados para a interação do usuário com o ambiente de nuvem.

3.2 Iniciativas Comerciais e de Código Aberto

Iniciativas comercias e de código aberto são notadas por oferecerem a elasticidade de forma manual, considerando a percepção do usuário (MILOJICIC; LLORENTE; MONTERO, 2011; LONEA; POPESCU; PROSTEAN, 2012; WEN et al., 2012; CAI et al., 2012; FUJII; KIMURA, 2011; FU; GONDI, 2010), ou através de configurações prévias (elasticidade reativa) (CHIU; AGRAWAL, 2010; ROLOFF et al., 2012; MARSHALL; KEAHEY; FREEMAN, 2010; SURHONE; TENNOE; HENSSONOW, 2010). A abordagem manual é especialmente comum em nuvens privadas. Usuários podem desenvolver suas próprias aplicações para monitoramento de dados e serviços que executam em máquinas virtuais na nuvem, gerenciando as operações de elasticidade quando necessário. Porém, algumas plataformas, como Amazon AWS[2], Microsoft Azure[3] e Nimbus[4], disponibilizam sistemas configuráveis para monitoramento de serviços e gerenciamento de elasticidade. Neste caso, os usuários devem definir regras e *thresholds* de determinadas métricas para serem monitoradas, assim como condições e ações de elasticidade. Os trabalhos avaliados nesta seção foram compilados na Tabela 2, em que foram analisadas características dos

[2]https://aws.amazon.com/pt/
[3]https://azure.microsoft.com/pt-br/
[4]http://www.nimbusproject.org/

serviços de nuvem oferecidos e as técnicas de elasticidade empregadas. Além disso, a tabela apresenta as ferramentas de controle do ambiente disponibilizadas pelas plataformas.

Tabela 2: Visão geral sobre iniciativas comerciais e de código aberto focadas em elasticidade em nuvem.

Trabalho	Serviço	Método	Modelo	Métricas	Operações	Interface
OpenNebula (MILOJICIC; LLORENTE; MONTERO, 2011)	IaaS	Horizontal e vertical	Manual	Definido pelo usuário	Replicação e Migração	CLI, GUI e API
Eucalyptus (LONEA; POPESCU; PROSTEAN, 2012)	IaaS	Horizontal e vertical	Manual	Definido pelo usuário	Replicação e Migração	CLI, GUI e API
Amazon AWS (CHIU; AGRAWAL, 2010)	IaaS, PaaS	Horizontal	Automático, Reativo com pré-configuração	CPU, Rede, Memória e Disco	Replicação	Amazon AutoScale, CloudWatch e API
Microsoft Azure (ROLOFF et al., 2012)	IaaS, PaaS	Horizontal	Automático, Reativo com pré-configuração	CPU e Rede	Replicação e Migração	GUI e API
Nimbus (MARSHALL; KEAHEY; FREEMAN, 2010)	IaaS	Horizontal	Automático, Reativo com pré-configuração	CPU	Replicação	GUI e Phantom plugin
OpenStack (WEN et al., 2012)	IaaS	Horizontal	Manual	Definido pelo usuário	Replicação	CLI e API
CloudStack (CAI et al., 2012)	IaaS	Horizontal e vertical	Manual	Definido pelo usuário	Replicação e Migração	CLI, GUI e API
RightScale (SURHONE; TENNOE; HENSSONOW, 2010)	IaaS	Horizontal	Automático, Reativo com pré-configuração	CPU, fila de tarefas	Replicação	Framework especializado e GUI
Heroku (FUJII; KIMURA, 2011)	PaaS	Horizontal	Manual	CPU e Rede	Replicação e Redimensionamento	Salesforce API
GoGrid (FU; GONDI, 2010)	IaaS	Horizontal	Manual	CPU	Replicação	GUI e API

Fonte: elaborado pelo autor.

A plataforma Amazon AWS disponibiliza API's e ferramentas gráficas para a realização de tarefas de monitoramento e gerenciamento de recursos. O monitoramento dos recursos é realizado por uma ferramenta chamada CloudWatch[5], e a elasticidade é gerenciada reativamente pela ferramenta Auto Scaling[6] (CHIU; AGRAWAL, 2010). Essa ferramenta oferece de forma gráfica ou através de uma ferramenta de linha de comando a possibilidade de o usuário configurar suas políticas de monitoramento e elasticidade. Além disso, Auto Scaling oferece uma API que permite que essas tarefas sejam programadas pelo desenvolvedor.

Já a plataforma Microsoft Azure oferece uma biblioteca, chamada *Autoscaling Application Block*, em que os usuários devem adicionar o projeto da aplicação para habilitar a característica de elasticidade. Essa abordagem permite ao programador utilizar con-

[5]http://aws.amazon.com/pt/cloudwatch/
[6]http://aws.amazon.com/pt/autoscaling/

tadores de desempenho e escrever regras de elasticidade. AzureWatch[7] e CloudMonix[8] são outros caminhos para controlar a quantidade de instâncias. É uma ferramenta que utiliza coleta uso de CPU, histórico, média do tempo de resposta e outras informações das máquinas virtuais (ROLOFF et al., 2012). Outra função realizada pelo AzureWatch é empregar regras e ações específicas.

O sistema Nimbus, por sua vez, trabalha como um escalonador de tarefas para ambientes de nuvem Amazon EC2 (MARSHALL; KEAHEY; FREEMAN, 2010). Nimbus utiliza Phantom para monitorar o comportamento de recursos baseando-se em uma métrica observada. Phantom permite definir uma quantidade mínima e máxima de máquinas virtuais para a execução de um serviço, a quantidade de máquinas virtuais que serão criadas quando o *threshold* superior é atingido e a quantidade que será removida quando o *threshold* inferior for atingido.

3.3 Iniciativas de Pesquisa Acadêmica

As iniciativas de pesquisa acadêmica buscam preencher lacunas ou melhorar abordagens existentes de elasticidade. Nesse sentido, os trabalhos podem ser classificados em oito subgrupos. O subgrupo (i) é composto por trabalhos que abordam aplicações paralelas e HPC, enquanto o grupo (ii) compreende trabalhos que possuem foco em custos financeiros quando utilizando ambientes de nuvem. O grupo (iii) reúne trabalhos relacionados à computação federada. Já o grupo (iv) lista os trabalhos focados em modelos preditivos para detecção de padrões de comportamento em serviços de nuvem. O grupo (v), por sua vez, é composto por trabalhos que abordam economia de energia e sobreposição de máquinas virtuais sobre máquinas físicas. Em seguida, o grupo (vi) reúne trabalhos relacionados à computação em tempo real e prazos para conclusão de serviços. O grupo (vii) engloba soluções de *middlewares* orientados a mensagens. Por fim, o grupo (viii) abrange os trabalhos focados em carga de trabalho web e aplicações para as áreas de negócios e transações. Os trabalhos avaliados nesta seção foram compilados na Tabela 3, em que foram analisadas características dos serviços de nuvem oferecidos e as técnicas de elasticidade empregadas. Além disso, a tabela apresenta as ferramentas de controle do ambiente disponibilizadas pelas plataformas. A seguir são listados os subgrupos descritos anteriormente:

(i) Computação paralela e HPC (MAO; LI; HUMPHREY, 2010; JACKSON et al., 2010; RAVEENDRAN; BICER; AGRAWAL, 2011; RAJAN et al., 2011; KNAUTH; FETZER, 2011; LIN et al., 2011; HAN et al., 2012; IMAI; CHESTNA; VARELA, 2012; MICHON; GOSSA; GENAUD, 2012; BEERNAERT et al., 2012; MARIANI et al., 2014; SPINNER et al., 2014);

[7] https://www.paraleap.com/
[8] http://cloudmonix.com/

(ii) Trabalhos orientados a custos financeiros (MAO; LI; HUMPHREY, 2010; SHARMA et al., 2011);

(iii) Computação em nuvem federada (MARSHALL; KEAHEY; FREEMAN, 2010; COSTA et al., 2010; IMAI; CHESTNA; VARELA, 2012);

(iv) Modelos preditivos para detecção de padrões de comportamento em serviços de nuvem (WOOD et al., 2007; MAO; LI; HUMPHREY, 2010; SHARMA et al., 2011; SHEN et al., 2011; MORENO; XU, 2011; ZHANG et al., 2012);

(v) Economia de energia e sobreposição de máquinas virtuais sobre máquinas físicas (MORENO; XU, 2011; KNAUTH; FETZER, 2011);

(vi) Computação em tempo real, endereçando prazos dos serviços (MAO; LI; HUMPHREY, 2010; RAVEENDRAN; BICER; AGRAWAL, 2011; MORENO; XU, 2011; TRAN; SKHIRI; ZIMáNYI, 2011);

(vii) Soluções de *middlewares* orientados a mensagens (TRAN; SKHIRI; ZIMáNYI, 2011);

(viii) Carga de trabalho web e aplicações para as áreas de negócios e transações (WOOD et al., 2007; ZHANG et al., 2010; DAWOUD; TAKOUNA; MEINEL, 2011; MARTIN et al., 2011a; SULEIMAN, 2012).

Em particular, os trabalhos contidos no grupo (i) voltado à computação paralela e HPC possuem forte relacionamento com esta dissertação. Destes trabalhos, ElasticMPI oferece elasticidade para aplicações MPI pausando a execução da aplicação e relançando os processos com uma nova configuração de recursos (RAVEENDRAN; BICER; AGRAWAL, 2011). O sistema assume que o usuário conhece previamente o tempo esperado para a conclusão de cada fase de execução da aplicação. O sistema de monitoramento pode adicionar mais recursos se detectar que a configuração atual não é suficiente para atingir um determinado prazo da aplicação. Essa detecção é realizada através de parâmetros informados pelo usuário que definem o tempo esperado para que cada fase de processamento seja completada e a distância permitida do tempo de cada fase para esse parâmetro. Além disso, a abordagem de ElasticMPI impõe alterações no código fonte da aplicação devido a necessidade de inserção de diretivas de monitoramento.

Uma abordagem diferente é realizada por Imai, Chestna e Varela (2012), que propõem uma abordagem em nível de sistema operacional chamada COS (*Cloud Operating System*), que oferece um *framework* genérico baseado na linguagem orientada a atores SALSA (*Simple Actor Language System and Architecture*) (VARELA; AGHA, 2001). O monitoramento é realizado individualmente pra cada uma das máquinas virtuais, o qual consiste na coleta da métrica CPU em intervalos de 5 segundos. Uma máquina virtual é considerada sobrecarregada se nas últimas 5 observações pelo menos em 3 delas a utilização de CPU é maior que um determinado *threshold*, definido em 75%. Quando observado

Tabela 3: Visão geral sobre iniciativas acadêmicas focadas em elasticidade em nuvem.

Trabalho	Serviço	Método	Modelo	Métricas	Operações	Interface
Just in Time (JIT) Clouds (COSTA et al., 2010)	IaaS	Horizontal e vertical	Manual	Definido pelo usuário	Replicação	API
PRESS (SHEN et al., 2011)	IaaS	Horizontal	Automático, Reativo	SLO	Redimensionamento	Framework para gerenciamento de recursos com XEN
ElasticMPI (RAVEENDRAN; BICER; AGRAWAL, 2011)	PaaS	Vertical	Automático, Reativo	CPU	Replicação	Diretivas de monitoramento no código fonte
Work Queue (RAJAN et al., 2011)	PaaS	Horizontal	Manual	CPU	Redimensionamento	Framework para Windows Azure e Amazon AWS
COS (IMAI; CHESTNA; VARELA, 2012)	PaaS	Horizontal	Automático, Reativo	CPU e Rede	Replicação e Migração	Gerenciamento no nível de sistema operacional
Mao, Li e Humphrey (2010)	IaaS	Horizontal	Automático, proativo	Tempo de conclusão de tarefas	Replicação	Sistema que executa sobre Windows Azure
Martin et al. (2011a)	SaaS	Horizontal e vertical	Automático, baseado em agente de monitoramento	CPU	Replicação	Trabalha com Amazon EC2
Lightweight Scaling (HAN et al., 2012)	IaaS	Horizontal e vertical	Automático, Reativo	CPU e Memória	Replicação, Redimensionamento	Framework para gerenciamento de recursos
Kingsfiher (SHARMA et al., 2011)	IaaS	Horizontal	Automático, Reativo e Proativo	Custo	Replicação e Migração	Executa sobre OpenNebula
Moreno e Xu (2011)	IaaS	Horizontal e vertical	Automático, Proativo	Energia	Replicação e Migração	Framework para gerenciamento de recursos
Scattered (ZHANG et al., 2012)	IaaS	Horizontal	Automático, proativo	CPU e Rede	Migração	Framework para gerenciamento de recursos
Sepiper (WOOD et al., 2007)	IaaS	Horizontal e vertical	Automático, Reativo e Proativo	CPU, Rede e Memória	Redimensionamento e Migração	Framework que opera sobre XEN
Elastack (BEERNAERT et al., 2012)	IaaS	Horizontal	Automático, Reativo	CPU	Replicação	Plugin que opera sobre OpenStack
Knauth e Fetzer (2011)	IaaS	Horizontal	Automático, Reativo	Energia	Migração	Framework para gerenciamento de serviço
Elastic Queue Service (TRAN; SKHIRI; ZIMáNYI, 2011)	IaaS	Horizontal e vertical	Automático, Reativo	Número de requisições	Replicação	Framework que opera sobre Amazon AWS
Elastic Site (MARSHALL; KEAHEY; FREEMAN, 2010)	IaaS	Vertical	Automático, Reativo com pré-configuração	Número de requisições	Replicação, Redimensionamento	Gerenciador de recursos para Nimbus
Dawoud, Takouna e Meinel (2011)	IaaS	Vertical	Automático, Reativo com pré-configuração	Service Level Objective (SLO)	Redimensionamento	Gerenciador de recursos sobre XEN
Suleiman (SULEIMAN, 2012)	IaaS	Vertical	Automático, Reativo	CPU	Replicação	Amazon AWS e CloudWatch
Zhang et al. (2010)	IaaS	Horizontal	Automático, Reativo com pré-configuração	CPU	Replicação	Baseado em componentes Weblet
Kaleidoscope (BRYANT et al., 2011)	IaaS	Horizontal	Automático, Reativo	CPU e Memória	Replicação	Micro-elasticidade por clonagem de VMs
Lin et al. (2011)	IaaS	Horizontal	Automático, Reativo	CPU e Memória	Replicação	Simulação usando CloudSim

Fonte: elaborado pelo autor.

que todas as máquinas virtuais estão sobrecarregadas, migrações podem ser realizadas tanto de um nodo para outro quanto de uma máquina virtual para outra. Porém, as operações são realizadas apenas se o sistema permanecer sobrecarregado por um determinado período de tempo configurado em 10 segundos. Contudo, COS funciona somente com aplicações restritamente compostas por componentes migráveis SALSA.

Já a abordagem de Mao, Li e Humphrey (2010) apresenta o conceito de auto-escalabilidade, em que a quantidade de máquinas virtuais flutua de acordo com a carga de trabalho. Diversas tarefas de tipos diferentes, intensivas quanto a processamento ou intensivas quanto a entrada e saída de dados, podem ser processadas através de uma fila. Uma vez que cada tarefa tem prazos para execução de cada uma de suas fases, a proposta trabalha alocação de máquinas virtuais para atender esses limites. Dessa maneira, o mecanismo de elasticidade utiliza parâmetros de prazo de execução e tipo de tarefa que será computada relacionados a cada tarefa na fila de processamento para definir qual configuração de máquinas virtuais é mais adequada para a quantidade de tarefas na fila.

Elastack, por sua vez, é um sistema que funciona sobre OpenStack e usa Serpentine (MATOS et al., 2008) para gerenciar alterações no ambiente (BEERNAERT et al., 2012). O trabalho propõe um modelo centralizado em que um balanceador de carga realiza a distribuição de tarefas para diversas máquinas virtuais que possuem um cliente sendo executado. Máquinas virtuais são adicionadas quando a soma total da métrica CPU de todas as máquinas virtuais excede 90% to total de CPU disponível. Dessa maneira, Serpentine cria uma nova instância de máquina virtual e informa ao balanceador de carga. Já a remoção de instâncias é realizada quando a carga de uma determinada instância pode ser distribuída entre as demais sem que o *threshold* de 90% seja atingido.

Por outro lado, Michon, Gossa e Genaud (2012) trabalham com diferentes estratégias de mapeamento de tarefas para máquinas virtuais: (i) todos as tarefas para uma máquina virtual; (ii) uma tarefa por máquina virtual; e (iii) primeiro ajuste. A entrada é marcado por uma série de lotes de tarefas que devem ser mapeados para as máquinas virtuais. Os autores realizam testes com dados fixos em relação ao conjunto de tarefas, prazos de execução e tempo em filas. Além disso, eles realizaram testes apenas com aplicações sacola-de-tarefas. Nesta abordagem os recursos são alocados conforme a estratégia que está sendo utilizada. Para cálculo de custos, o modelo de cobrança da Amazon é utilizado, em que cada recurso é cobrado em unidades de tempo. A cada unidade de tempo os recursos e tarefas são avaliados para identificar a possibilidade de redução da quantidade de recursos.

Seguindo uma abordagem diferente, Lin et al. (2011) usam o simulador CloudSim (CALHEIROS et al., 2011) para habilitar seu esquema de alocação dinâmica de recursos baseado em *thresholds*. Ao invés de realizar a atividade de monitoramento em intervalos fixos, a proposta transforma o intervalo atual maleável de acordo com a regularidade da aplicação. Apesar desta vantagem, a execução do código deve esperar quando primitivas

de bloqueio para reorganização dos recursos. Além disso, não há nenhum tratamento pico quando se atinge os *thresholds*. A abordagem requer que seja informado previamente a maior taxa de ocupação que a aplicação pode atingir. Baseando-se nessa taxa, é definido um *threshold* que, combinado com a maior taxa de ocupação e com a quantidade de recursos ativos, é utilizado para calcular a variação da carga de trabalho máxima permitida. Dessa maneira, o sistema monitora as variações na carga de trabalho, que deve ser conhecida, e se essa variação ficar acima do valor calculado, são realizadas operações de reorganização de recursos. Nos testes utilizando CloudSim, foram utilizados os *thresholds* 95%, 75% e 50%.

Diferentemente dessas abordagens, Rajan et al. (2011) apresentam uma abordagem para transformar uma aplicação mestre-escravo MPI não elástica, que aplica a técnica de simulação Monte Carlo (KROESE et al., 2014), em uma aplicação elástica através do *framework* chamado Work Queue (YU et al., 2010). Através dessa abordagem, o usuário deve desenvolver um *script* utilizando as diretivas propostas pelo trabalho. Este *script* funciona como o mestre que recebe os parâmetros e a aplicação que deve ser executada. A execução é realizada de forma iterativa e ao final de cada iteração o *script* deve verificar se existem mais recursos disponíveis que possam ser utilizados. Porém, a reorganização dos recursos é realizada de forma manual, em que o usuário deve adicionar ou remover mais processos escravos ao ambiente.

Buscando uma melhor eficiência de custo, Han et al. (2012) apresentam uma abordagem para gerenciamento de recursos em nuvens IaaS focando os provedores de nuvem. Esta abordagem consiste em monitorar o tempo de resposta de uma aplicação, definindo-se *thresholds* superiores e inferiores. Caso esses *thresholds* sejam violados, são realizadas operações de elasticidade de forma vertical e horizontal, buscando-se atingir tempo de resposta dentro dos *thresholds*. Utilizando a reorganização vertical de recursos, são utilizados os *thresholds* superior e inferior 80% e 30% respectivamente. Esses *thresholds* são utilizados para identificar máquinas virtuais em uma mesma máquina física a fim de transferir unidades de recursos (CPU e Memória) de uma máquina que está abaixo do *threshold* 30% para uma máquina que está acima do *threshold* 80%. Em caso de nenhuma combinação permitir essa reorganização, é realizada a elasticidade horizontal em que mais uma máquina virtual é adicionada ao ambiente.

Por outro lado, Knauth e Fetzer (2011) propõem um estratégia de escalabilidade para aplicações não elásticas através da migração de máquinas virtuais. O foco do trabalho é em fornecer alta taxa de transferência de dados e baixa latência para processamento de fluxo de dados. Para isso, a estratégia consiste em migrar máquinas virtuais que estão em uma mesma máquina física para uma nova máquina física que não possui nenhuma máquina virtual. Os experimentos foram realizados utilizando uma aplicação previamente proposta pelos autores chamada StreamMine (MARTIN et al., 2011b), a qual implementa uma aplicação MapReduce (DEAN; GHEMAWAT, 2008). As operações de alocação de

recursos são realizadas em instantes fixos previamente configurados, sabendo-se do comportamento da taxa de chegada de requisições.

Por sua vez, Jackson et al. (2010) recriam um *cluster* virtual em uma nuvem Amazon EC2 para a execução de aplicações MPI. Assim como *clusters* tradicionais, a arquitetura é composta por uma máquina virtual que atua como *head node*, responsável por submeter as tarefas, diversos máquinas virtuais que atuam como nós trabalhadores e uma máquina virtual servindo como servidor de arquivos utilizada para criar uma área de dados compartilhada entre todas as máquinas virtuais. A abordagem realiza uma avaliação do desempenho de aplicações HPC executadas na nuvem, utilizando *benchmarks* e cargas de trabalhos provindas do centro de computação *National Energy Research Scientific Computing Center* (NERSC)[9]. Porém, a elasticidade de recursos em tempo de execução não é explorada, sendo necessárias reconfigurações manuais em sua abordagem.

Por fim, Spinner et al. (2014) confronta a abordagem típica da utilização de *thresholds* através de um algoritmo proposto que calcula em cada nova fase qual a configuração ideal de CPU's para cada uma das máquinas virtuais em execução. Dessa forma, é utilizada a elasticidade vertical em que CPU's são adicionas ou removidas das máquinas virtuais. O algoritmo utiliza um parâmetro que define a agressividade com que a elasticidade irá ocorrer, sendo utilizado o valor de 75% nos testes realizados.

3.4 Análise e Oportunidades de Pesquisa

Esta seção apresenta alguns pontos fracos em relação aos sistemas estudados nas seções anteriores. Iniciativas comerciais normalmente requerem intervenção do usuário em configurações de elasticidade. Considerando estratégias para identificar a necessidade de operações de elasticidade, estas abordagens oferecem a elasticidade de forma reativa através da configuração de regras em que uma métrica é monitorada e, em caso de seu valor atingir um determinado *threshold* por um intervalo específico de tempo, são realizadas ações de elasticidade. Como sendo uma operação que consome tempo, o momento de uma operação de adição de recursos deve ser cuidadosamente analisado para executar aplicações HPC na nuvem adequadamente. Ambos Amazon AWS (CHIU; AGRAWAL, 2010) e Microsoft Azure (ROLOFF et al., 2012) esperam uma quantidade específica de observações de carga consecutivas fora da margem de um *threshold* para lançar ações de elasticidade. Além destes parâmetros, também é oferecido um parâmetro chamado *cooldown*, que se refere a um período de tempo, após a realização de uma operação, em que novas operações de elasticidade não são permitidas (JAMSHIDI; AHMAD; PAHL, 2014a). Em resumo, analisando os trabalhos de pesquisa e iniciativas comerciais, podem ser destacados os seguintes pontos fracos:

(i) Nenhuma análise de situações de pico esporádicos quando se atinge um *threshold* (LIN

[9]https://www.nersc.gov/

et al., 2011; MARTIN et al., 2011a; BEERNAERT et al., 2012);

(ii) Necessidade de alterar o código fonte da aplicação ou desenvolver *scripts* adicionais (RAVEENDRAN; BICER; AGRAWAL, 2011; RAJAN et al., 2011; COPIL et al., 2013; MARIANI et al., 2014);

(iii) Utilização de componentes proprietários que não estão disponíveis como bibliotecas de programação para sistemas operacionais conhecidos, tais como GNU-Linux, Windows e MacOS (RAVEENDRAN; BICER; AGRAWAL, 2011; IMAI; CHESTNA; VARELA, 2012; BEERNAERT et al., 2012);

(iv) Necessidade de conhecer o comportamento da aplicação com antecedência, como o tempo de execução esperado dos componentes (RAVEENDRAN; BICER; AGRAWAL, 2011; KNAUTH; FETZER, 2011; KUMAR et al., 2011; MICHON; GOSSA; GENAUD, 2012);

(v) Reconfiguração de recursos com a abordagem *stop-reconfigure-and-go* (RAVEENDRAN; BICER; AGRAWAL, 2011);

(vi) Comunicação entre máquinas virtuais em uma taxa constante (ZHANG et al., 2012).

Considerando a área específica de aplicações paralelas e elasticidade, destacam-se algumas iniciativas: Jackson et al. (2010), ElasticMPI (RAVEENDRAN; BICER; AGRAWAL, 2011), Rajan et al. (2011), Mariani et al. (2014) e Spinner et al. (2014). Entre elas, três iniciativas executam aplicações iterativas, onde cada nova fase significa que há um novo esforço para redistribuir as tarefas aos escravos (JACKSON et al., 2010; RAVEENDRAN; BICER; AGRAWAL, 2011; RAJAN et al., 2011). A elasticidade em Rajan et al. (2011) é oferecida manualmente, em que o usuário captura dados de monitoramento, utilizando a estrutura proposta pelos autores, e adiciona ou remove recursos do ambiente. Apesar das lacunas mencionadas antes sobre ElasticMPI (RAVEENDRAN; BICER; AGRAWAL, 2011), a ideia é oferecer (com algumas limitações) elasticidade sobre programas MPI existentes (versões 1 e 2). Jackson et al. (2010) executam aplicações mestre-escravo da NERSC como *benchmark* para medir o desempenho de configurações de um *cluster* no Amazon EC2. Já Spinner et al. (2014) propõem um *middleware* de escala vertical para máquinas virtuais individuais que executam aplicações HPC. Eles argumentam que a abordagem horizontal é proibitiva no âmbito HPC porque, segundo eles, a inicialização de uma máquina virtual leva pelo menos 1 minuto para ficar pronta. Finalmente, a elasticidade no trabalho de Mariani et al. (2014) é oferecida em nível de API, em que o usuário gerencia reconfigurações de recursos por si mesmo. Esta estratégia requer experiência em nuvem pelo programador e não seria portável entre diferentes implantações de nuvem.

3.5 Considerações Parciais

Este capítulo apresentou um levantamento de trabalhos relacionados à questão de pesquisa desta dissertação. Considerando os pontos fracos e estratégias descritas, é possível notar que um usuário que deseja utilizar o recurso de elasticidade em um ambiente de Computação em Nuvem se depara com uma série de questões relacionadas a configuração e parametrização dos sistemas de monitoramento disponíveis. Devido a isso, é necessário que o usuário possua bastante conhecimento do comportamento da aplicação para configurar os parâmetros adequadamente. Em alguns casos, o próprio usuário deve desenvolver um sistema de monitoramento ou um gerenciador de elasticidade. Além disso, aplicações HPC possuem dificuldade para tirar proveito da dinamicidade de um ambiente de Computação em Nuvem, sendo necessária a alteração no código da aplicação ou a inserção de diretivas de elasticidade. Isso torna necessário que ao desenvolver a aplicação, o programador deve ser ciente da elasticidade e a plataforma de destino que executará a aplicação. Considerando estas limitações, é possível identificar oportunidades de trabalho para o desenvolvimento de um sistema de elasticidade em nuvem para aplicações HPC com o objetivo de facilitar o uso deste recurso. Estas oportunidades podem ser aprofundadas através do desenvolvimento deste ambiente contemplando algumas das lacunas descritas na Seção 3.4. No próximo capítulo, é apresentado o modelo desenvolvido considerando as lacunas identificadas neste capítulo.

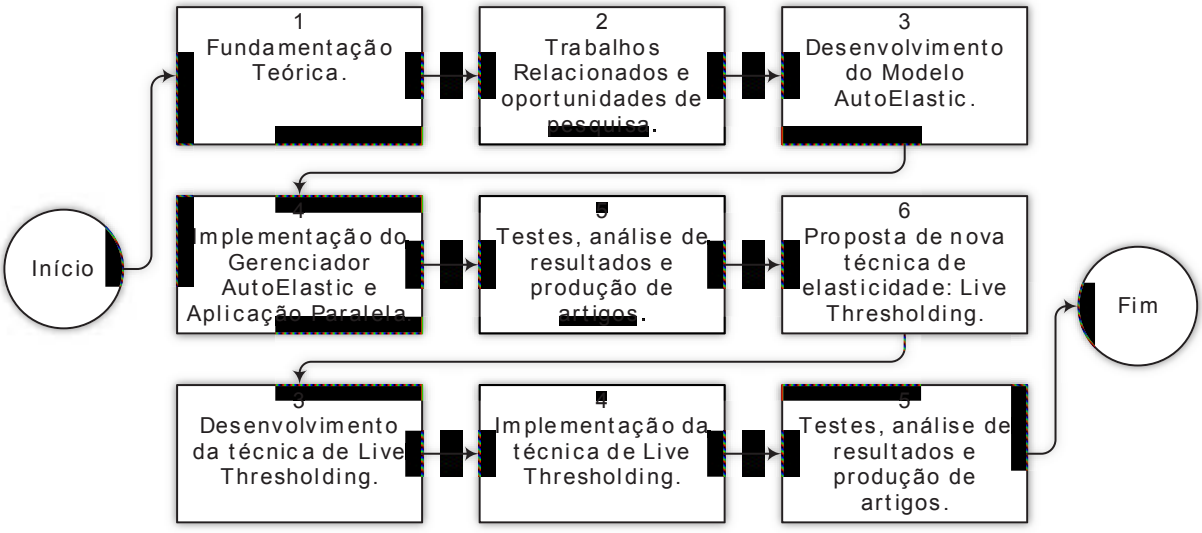
4 MODELO AUTOELASTIC

Este capítulo descreve o modelo AutoElastic, que é um modelo de elasticidade em nuvem focado para aplicações HPC iterativas. A Seção 4.1 apresenta as etapas realizadas durante o desenvolvimento do modelo. Em seguida, a Seção 4.2 apresenta as decisões de projeto para a o desenvolvimento do modelo. Na Seção 4.3 é apresentada e definida a arquitetura do modelo AutoElastic e seus componentes. Na Seções 4.4 e 4.5 são apresentados respectivamente o funcionamento do mecanismo de elasticidade e o modelo de aplicação paralela suportada por AutoElastic. Em seguida, a Seção 4.6 apresenta as métricas definidas para avaliação de desempenho de aplicações elásticas. Por fim, na Seção 4.7 são realizadas algumas considerações parciais sobre o capítulo.

4.1 Etapas de Desenvolvimento da Pesquisa

Seguindo o plano de pesquisa apresentado na Subseção 1.4, em que é previsto um ciclo em que podem ser desenvolvidas novas abordagens no modelo, a Figura 5 apresenta as etapas que foram realizadas durante a pesquisa. Após serem realizados os primeiros testes e obtidos os primeiros resultados considerando a utilização do modelo AutoElastic com *thresholds* de elasticidade fixos, foi identificada a possibilidade da utilização de uma diferente abordagem de *thresholds* em que o usuário não necessita fazer essa configuração em seu ambiente de nuvem. Dessa maneira, foi desenvolvida uma nova técnica, nomeada Live Thresholding, em que o Gerenciador AutoElastic realiza a adaptação dos *thresholds* durante a execução da aplicação, considerando o histórico de monitoramento da carga de processamento dos recursos da nuvem.

Figura 5: Etapas realizadas durante a pesquisa, conforme fluxo apresentado na Figura 3.



Fonte: elaborado pelo autor.

4.2 Decisões de Projeto

AutoElastic fornece elasticidade horizontal e reativa de forma transparente para aplicações paralelas, sem a necessidade de intervenção do usuário para definição de regras e ações ou modificações no código fonte da aplicação. Foi adotada a elasticidade horizontal devido ao fato de que, com elasticidade vertical, a quantidade de recursos fica limitada sempre aos recursos disponíveis em uma única máquina física e, além disso, a maioria dos sistemas operacionais comuns não permitem que sejam adicionados recursos em tempo de execução (DUTTA et al., 2012; LORIDO-BOTRAN; MIGUEL-ALONSO; LOZANO, 2014). Do ponto de vista do usuário, AutoElastic concentra-se em um grupo de usuários que não quer se envolver com a adaptação de sua aplicação para utilizar a elasticidade de recursos. Dessa maneira, considera-se que o usuário já tem sua aplicação desenvolvida para execução em ambientes com a quantidade de recursos fixa e quer executar essa mesma aplicação em um ambiente de nuvem dinâmico sem a necessidade de alterar seu código fonte. Essa abordagem adotada por AutoElastic caracteriza a elasticidade automática e impõe algumas limitações para grupos de usuários mais avançados. Neste sentido, o usuário não tem acesso ao funcionamento da elasticidade e devido a isso não é possível forçar manualmente a reorganização de recursos em pontos específicos da aplicação.

A Figura 6 (a) ilustra as abordagens tradicionais para fornecer elasticidade para aplicações HPC, enquanto (b) destaca a ideia de AutoElastic. A abordagem desenvolvida por AutoElastic permite usuários submeterem aplicações tradicionais para a nuvem, não cientes da elasticidade, enquanto o *framework* se encarrega da reorganização de recursos através de procedimentos automáticos de alocação e consolidação. Ainda, AutoElastic opera com a granularidade de máquinas virtuais, devendo estar ciente da sobrecarga de inicializar uma máquina virtual, levando em conta esse conhecimento para oferecer esse recurso sem custos proibitivos. O modelo busca responder às seguintes sentenças:

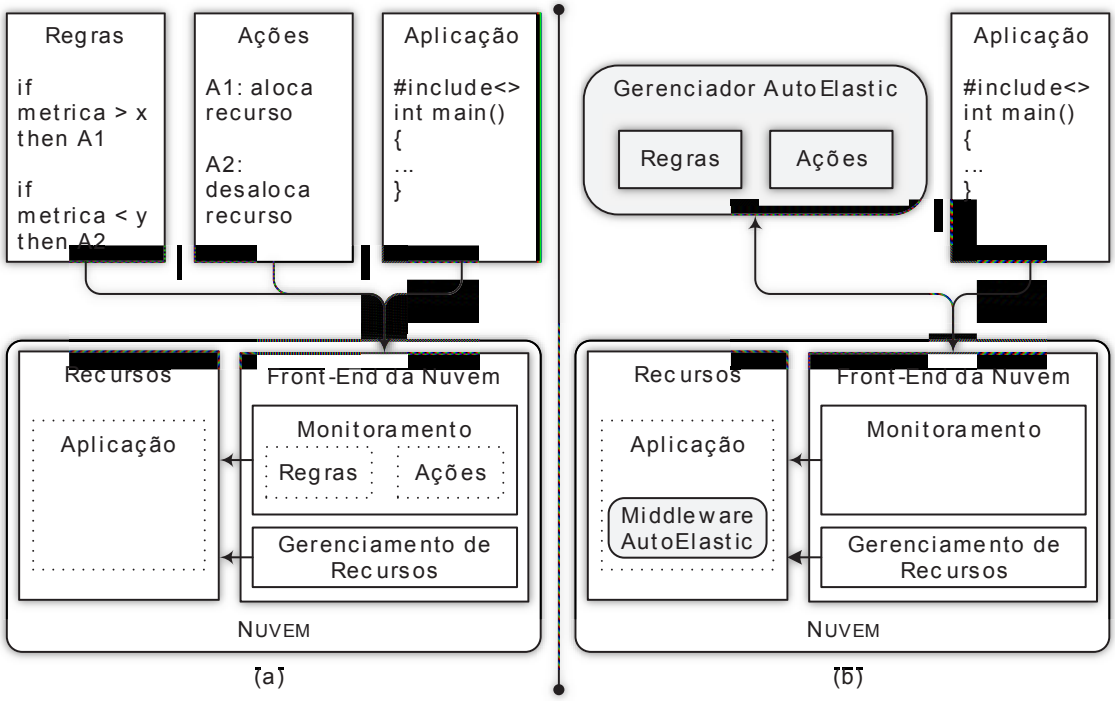
(i) *Quais mecanismos são necessários para prover elasticidade transparentemente nos níveis de usuário e aplicação, como uma capacidade viável em aplicações HPC?*

(ii) *Quais aplicações HPC podem se beneficiar da elasticidade em nuvem e sob quais restrições ela pode ser suportada?*

(iii) *Quais são as suposições mínimas para transparentemente suportar elasticidade em nuvem para aplicações HPC?*

AutoElastic adota a elasticidade reativa baseada em *thresholds* em que ações de elasticidade ocorrem se a métrica de carga monitorada violar algum dos *thresholds*, conforme apresentado na Figura 7. As regras são definidas sem a intervenção do usuário, o qual também não precisa configurar as ações necessárias para viabilizar a elasticidade. A definição de regras de elasticidade não é uma tarefa trivial por muitas razões, incluindo

Figura 6: Ideias gerais da utilização de elasticidade: (a) abordagem padrão adotada pela Amazon AWS e Windows Azure, em que o usuário deve pré-configurar um conjunto de regras e ações; (b) ideia de AutoElastic, contemplando um gerenciador que coordena as ações de elasticidade.



Fonte: elaborado pelo autor.

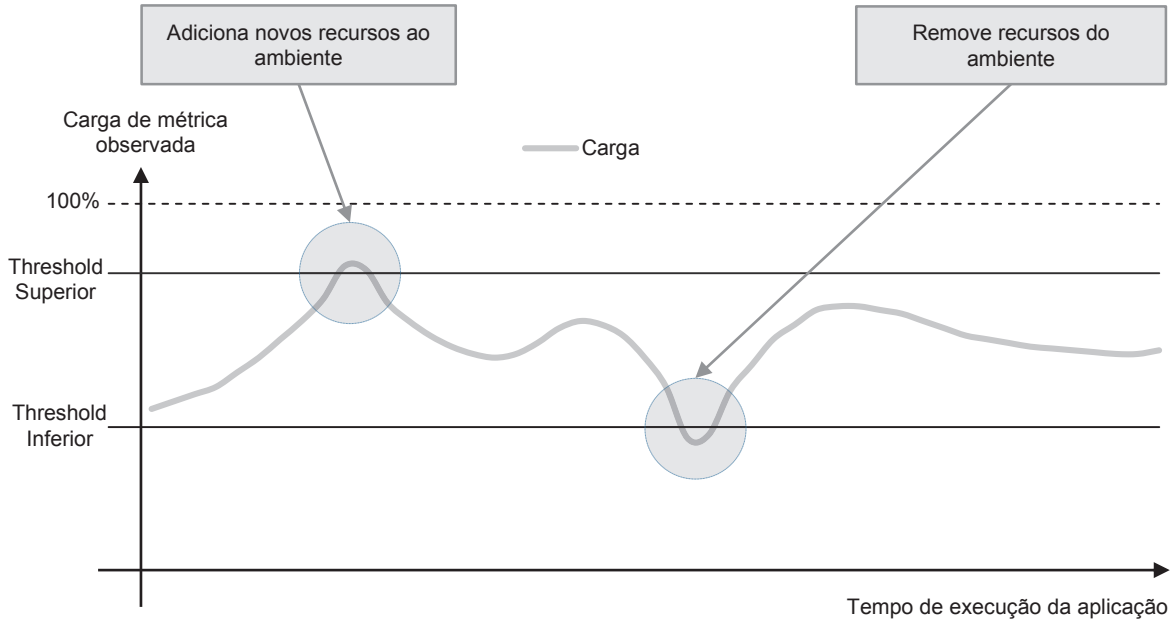
a configuração de um ou mais *thresholds*, tipos de recursos, quantidade de ocorrências, definições de picos e janelas de monitoramento. Além disso, é comum haver situações em que regras de elasticidade obtenham bons resultados para uma aplicação e resultados não tão bons para outras. Adicionalmente, a elasticidade dispara alocações de recursos que apresentam impacto direto no custo de utilização da nuvem. Assim, a relação custo benefício deve ser cuidadosamente avaliada. A abordagem de AutoElastic provê elasticidade escondendo do programador todas as ações de reconfiguração de recursos, executando sem que o programador realize nenhuma modificação no código fonte da aplicação. Em particular, AutoElastic foca aplicações que não utilizam prazos específicos para concluir subpartes. Suas contribuições cobrem dois tópicos, os quais nenhum foi encontrado nos trabalhos relacionados:

(i) Controle eficiente do lançamento e consolidação de máquinas virtuais totalmente transparente para o usuário;

(ii) Mecanismo para executar aplicações HPC na nuvem de maneira não proibitiva.

A Figura 6 (b) demonstra a principal ideia de AutoElastic. Atuando no nível PaaS de uma nuvem, AutoElastic apresenta um *middleware* que pode ser visto como uma biblioteca de comunicação utilizada para compilar a aplicação. Além disso, dispõe de um gerenciador

Figura 7: Modelo de elasticiade reativa baseada em *thresholds* adotada pelo modelo AutoElastic.



Fonte: elaborado pelo autor.

de elasticidade que controla as reconfigurações de recursos. Para o desenvolvimento do modelo, foram definidas as seguintes decisões de projeto:

(i) Usuários não precisam configurar o mecanismo de elasticidade, porém há a possibilidade de informar o SLA com a quantidade mínima e máxima de máquinas virtuais para executar a aplicação;

(ii) Desenvolvedores não precisam reescrever o código da aplicação a fim de tirar proveito da elasticidade;

(iii) O cenário de investigação consiste na execução de uma aplicação simples em uma nuvem privada, composta por recursos homogêneos não compartilhados entre usuários;

(iv) A estratégia de elasticidade adotada será de forma reativa, automática e horizontal, utilizando a replicação de máquinas virtuais;

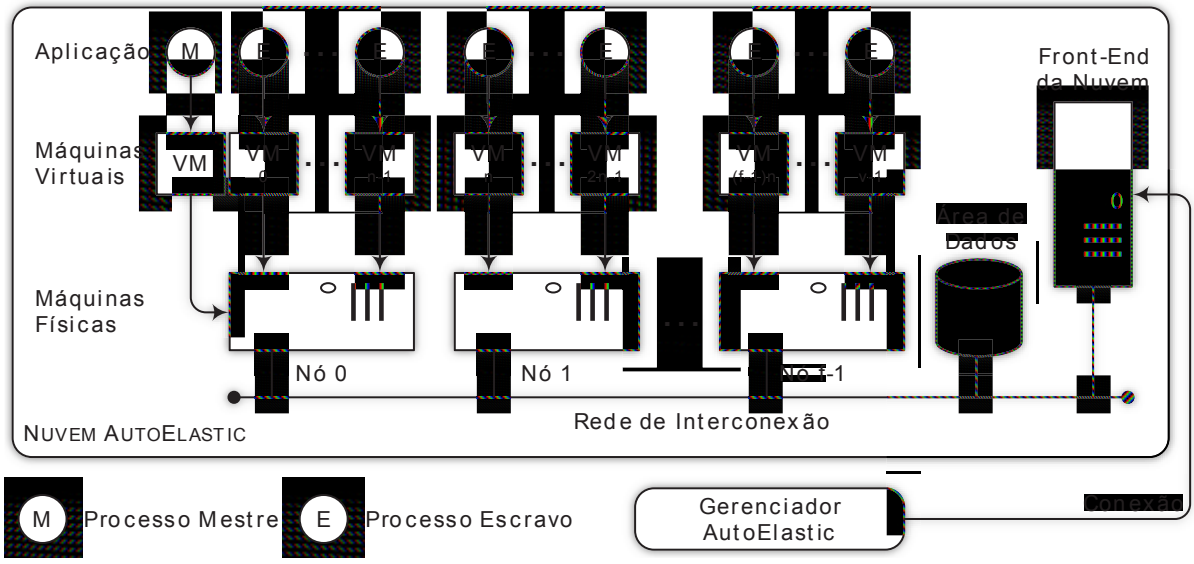
(v) Devido ao fato de ser baseado no nível de PaaS, AutoElastic deve supor o uso de ferramentas que automaticamente transforma a aplicação paralela em uma aplicação elástica de forma transparente para os usuários;

(vi) AutoElastic deve analisar picos e quedas repentinas na carga de trabalho a fim de evitar operações desnecessárias, evitando o fenômeno conhecido como *thrashing* (BERSANI et al., 2014).

4.3 Arquitetura

AutoElastic é um *middleware* que opera no nível de PaaS permitindo aplicações paralelas iterativas não elásticas obterem vantagem da elasticidade em nuvem sem a necessidade de alterações no código fonte por parte do usuário. Para fornecer elasticidade, o modelo atua com operações de alocação e consolidação de instâncias de máquinas virtuais e máquinas físicas. A Figura 8 representa a arquitetura do modelo AutoElastic, apresentando os componentes do *framework* e o mapeamento de máquinas físicas e processos em um ambiente de nuvem composto por máquinas físicas homogêneas. O *framework* inclui um Gerenciador, que não possui dependência de localização para sua execução. Dessa maneira, o Gerenciador pode ser executado tanto dentro do ambiente de nuvem como fora em uma máquina totalmente independente do ambiente. Isso é possível através do uso da API fornecida pela plataforma de nuvem utilizada no ambiente, a qual necessita apenas uma conexão de rede. Além disso, a arquitetura contempla uma área de dados compartilhada entre todos os componentes do ambiente, incluindo o Gerenciador, que acessa essa área através do Front-End da nuvem. Essa área de dados é um componente importante, possibilitando a implementação de políticas de comunicação entre os componentes e processos do ambiente. Apesar de a área de dados e o Front-End da nuvem serem representados de forma centralizada, a sua implementação pode ser realizada de forma distribuída evitando problemas de dependência.

Figura 8: Arquitetura de AutoElastic, em que a quantidade de máquinas físicas é f e a quantidade de núcleos de processamento em uma máquina física é identificada por n. A quantidade de máquinas virtuais (VM) executando processos escravos é v, que pode ser computada por $n \times f$.



Fonte: elaborado pelo autor.

Na Figura 8, o *middleware* AutoElastic não é representado pois é um componente integrado com cada processo da aplicação. Na Tabela 4 são descritos os parâmetros utili-

Tabela 4: Descrição dos parâmetros utilizados na definição da arquitetura do modelo apresentada na Figura 8.

Parâmetro	Descrição
f	Quantidade de máquinas físicas no ambiente de nuvem.
n	Quantidade de núcleos de processamento dentro de cada máquina física.
v	Quantidade de máquinas virtuais executando processos escravo.

Fonte: elaborado pelo autor.

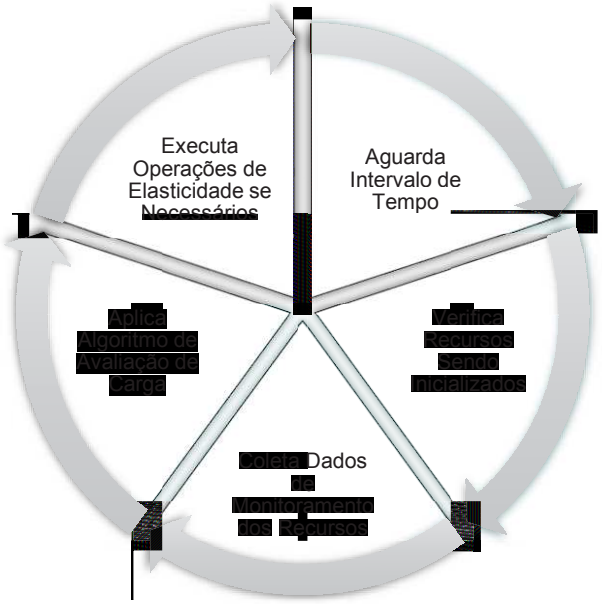
zados na definição do modelo. O ambiente AutoElastic é composto por f máquinas físicas homogêneas em que, a qualquer momento da execução da aplicação, pelo menos uma está ativa. Como aplicações HPC em geral são intensivas quanto a CPU (AZMANDIAN et al., 2011), são distribuídos um processo por máquina virtual e n máquinas virtuais por máquina física, em que n representa a quantidade de núcleos de processamento dentro da máquina física destino. Essa abordagem é baseada no trabalho de Lee et al. (2011), em que os autores procuram explorar uma melhor eficiência em aplicações paralelas. O grão de elasticidade no ambiente para cada operação de adição ou remoção de recursos é sempre uma máquina física, e consequentemente, suas máquinas virtuais e processos. Por fim, a quantidade de máquinas virtuais em operação no ambiente é definida por $v + 1$, sendo v a quantidade de máquinas virtuais executando processos escravos, o qual pode ser obtido por $v = n \times f$. Considerando o escopo de nuvem, uma nuvem AutoElastic pode ser definida como segue:

> **Definição 1 - Nuvem AutoElastic**: uma nuvem modelada com f recursos computacionais homogêneos e distribuídos, em que no mínimo um deles (Nó 0) está sempre ativo. Essa máquina física é encarregada de executar uma máquina virtual com o processo mestre e outras n máquinas virtuais com um processo escravo cada, em que n representa o número de núcleos de processamento dentro de um nó em particular. O grão de elasticidade para cada ação se refere a uma simples máquina física. Por fim, a qualquer momento, o número de máquinas virtuais executando processos escravos é igual a $v = n \times f$.

As principais funções de monitoramento e gerenciamento da elasticidade são realizadas pelo Gerenciador AutoElastic. Ele é o responsável por realizar a coleta de dados dos recursos do ambiente, aplicar a avaliação e gerenciar as operações de elasticidade. A Figura 9 apresenta o seu ciclo de monitoramento, demonstrando as principais operações realizadas. A cada intervalo de tempo, o gerenciador verifica se existem recursos inicializados anteriormente que podem estar disponíveis para a utilização. Isso ocorre pois, quando novos recursos são adicionados ao ambiente, eles estarão disponíveis apenas após um determinado tempo. Após essa verificação, o Gerenciador coleta os dados de monitoramento dos recursos aos quais aplica um algoritmo de avaliação buscando obter uma

única métrica que define a carga do ambiente. O resultado desse cálculo é utilizado para comparação com os *thresholds* buscando identificar situações que possam ocasionar operações de elasticidade. Após a realização ou não dessas operações, o gerenciador aguarda novamente um intervalo de tempo para iniciar o próximo ciclo de monitoramento.

Figura 9: Principais operações realizadas pelo Gerenciador AutoElastic em cada ciclo de monitoramento.



Fonte: elaborado pelo autor.

O Gerenciador AutoElastic considera dados de carga de trabalho das máquinas físicas que compõem a infraestrutura de nuvem como entrada para análise de necessidades de reconfigurações de recursos para aplicações paralelas iterativas do modelo mestre-escravo. Embora trivial, este tipo de construção é utilizado em muitas áreas, tais como algoritmos genéticos, técnicas de Monte Carlo, transformações geométricas em computação gráfica, algoritmos de criptografia e aplicações que seguem o modelo *Embarrassingly Parallel* (RAVEENDRAN; BICER; AGRAWAL, 2011). Em resumo, a ideia é oferecer a elasticidade para o usuário de uma maneira sem esforço considerando a perspectiva de desempenho, o qual é obrigatório em cenários de HPC. Embora a estratégia de AutoElastic empregue a replicação de máquinas virtuais para aplicações no estilo mestre-escravo, a qual é fortemente utilizada para aplicações web com balanceadores de carga, a arquitetura de AutoElastic permite que aplicações HPC iterativas possam tirar proveito desta abordagem. AutoElastic implementa estratégias que possibilitam uma aplicação paralela, após ser automaticamente transformada em uma aplicação elástica, estar ciente da adição ou remoção de recursos sem a necessidade de envolvimento do usuário. Essas estratégias utilizam diretivas de comunicação comum em aplicações paralelas, tornando a arquitetura extensível a outros modelos de programação paralela.

No que diz respeito ao monitoramento de recursos, o Gerenciador AutoElastic mo-

nitora a utilização de recursos das máquinas virtuais, realizando ações de elasticidade quando consideradas necessárias para a atual configuração do ambiente e comportamento da carga de trabalho aplicação, a qual é refletida no consumo de CPU. Este recurso foi modelado especialmente para trabalhar com aplicações paralelas mestre-escravo iterativas, em que o Gerenciador AutoElastic controla as operações simultaneamente com a execução da aplicação. O usuário pode informar um SLA contendo a quantidade mínima e máxima de máquinas virtuais para a execução da aplicação na nuvem. Se o SLA não é informado, essas restrições são definidas baseando-se na quantidade de máquinas virtuais no ponto inicial de execução da aplicação. O fato do Gerenciador AutoElastic, e não a aplicação por si, realizar as operações de reconfiguração de recursos oferece o benefício da elasticidade assíncrona. Apesar de que Spinner et al. (2014) afirmam que apenas elasticidade vertical é adequada para cenários HPC devido a sobrecarga nas operações de reorganização de recursos, com a elasticidade assíncrona, torna-se viável a utilização da elasticidade horizontal sem impactar na execução da aplicação. Dessa maneira, a execução da aplicação e as ações de elasticidade ocorrem simultaneamente, não penalizando a aplicação com a sobrecarga da reconfiguração de recursos. A seguir, o conceito de elasticidade assíncrona é definido (R. RIGHI et al., 2016):

> **Definição 2 - Elasticidade Assíncrona**: é uma maneira de assincronamente notificar uma aplicação que está executando em nuvem sobre mudanças na disponibilidade de recursos do ambiente, tais como o número de instâncias de máquinas virtuais. Por exemplo, a aplicação é notificada assim que uma nova instância de uma máquina virtual está disponível no ambiente, sem prejudicar seu fluxo de execução normal.

No entanto, esta operação não-bloqueante implica na seguinte pergunta: *Como podemos notificar a aplicação sobre a reconfiguração de recursos?* Para isso, AutoElastic oferece a comunicação entre as máquinas virtuais e o Gerenciador AutoElastic utilizando uma área de dados compartilhada. Essa é uma área privada para as máquinas virtuais e as máquinas físicas dentro do ambiente da nuvem AutoElastic. A comunicação entre os processos da aplicação e o Gerenciador AutoElastic através dessa área pode ser viabilizada, por exemplo, via NFS (*Network File System*), *middleware* orientado a mensagens (tais como JMS ou AMQP) ou espaço de tuplas (como JavaSpaces). O uso de uma área compartilhada para interação de dados entre máquinas virtuais é uma abordagem comum em nuvens privadas (CAI et al., 2012; MILOJICIC; LLORENTE; MONTERO, 2011; WEN et al., 2012). Através dessa área de dados é possível realizar a troca de mensagens entre o Gerenciador AutoElastic e os processos da aplicação, os quais possuem o *middleware* AutoElastic integrado viabilizando essa comunicação. Dada a natureza iterativa da aplicação, a cada novo laço ela realiza verificações na área de dados a fim de identificar se novos recursos estão disponíveis ou se recursos serão removidos. Através dessas notifi-

cações é possível que a aplicação reorganize seus processos realizando novas conexões ou desconectando processos que serão removidos.

Através da área de dados compartilhada, a troca de mensagens pode ser realizada através de três tipos de notificações para viabilizar a elasticidade assíncrona, conforme demonstra a Tabela 5. A Notificação 1 é realizada de forma assíncrona pelo Gerenciador AutoElastic anunciando que novos recursos foram alocados e estão disponíveis para a aplicação. Ainda, a Notificação 2 também é realizada pelo Gerenciador AutoElastic e sua necessidade se dá por duas razões: (i) evitar a finalização abrupta de processos enquanto procedimentos de comunicação ou computação ainda estiverem ocorrendo, o que pode causar perda de dados; (ii) garantir que a aplicação não será finalizada devido a interrupções súbitas de seus processos. Em particular, a segunda razão é pertinente para aplicações MPI que executam sobre redes TCP/IP pois são usualmente interrompidas com o término prematuro de algum de seus processos. Por fim, a Notificação 3 é realizada pelo processo mestre da aplicação após tratar uma Notificação 2 recebida anteriormente garantindo que a aplicação está em um estado global consistente em que processos podem ser desconectados apropriadamente. Em outras palavras, uma vez recebida uma Notificação 2, o processo mestre finaliza sua comunicação com os processos que serão consolidados não enviando mais tarefas para eles. Após isso, o processo mestre envia a Notificação 3 permitindo que o Gerenciador AutoElastic consolide os recursos. A área de dados compartilhada desempenha um papel chave neste processo, pois através dela todos os processos são mantidos informados sobre reconfigurações de recursos, permitindo uma adaptação segura para a nova topologia de rede. A Figura 10 demonstra como a elasticidade assíncrona é viabilizada através da Notificação 1.

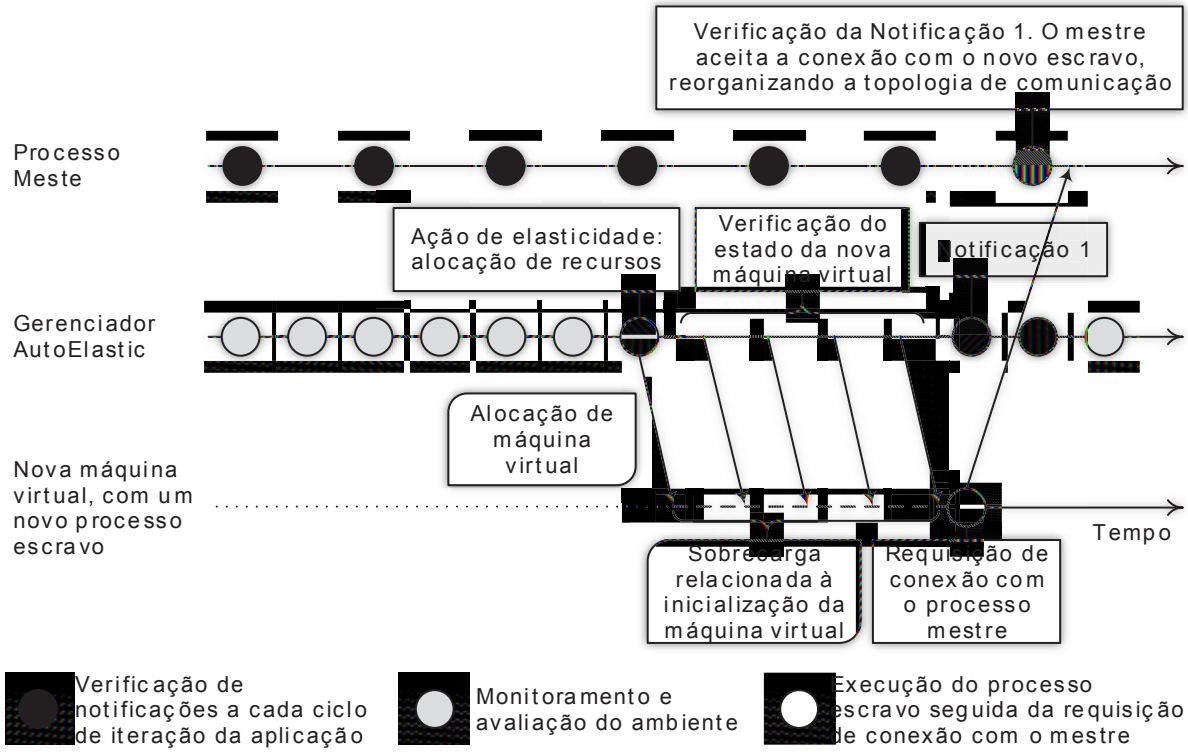
Tabela 5: Notificações fornecidas através da área de dados compartilhada a fim de viabilizar a comunicação entre o Gerenciador AutoElastic e o processo mestre da aplicação.

Notificação	Direção	Descrição
Notificação 1	Gerenciador AutoElastic → Processo Mestre	Disponível n novos recursos.
Notificação 2	Gerenciador AutoElastic → Processo Mestre	Solicitação de permissão para consolidação de recursos específicos.
Notificação 3	Processo Mestre → Gerenciador AutoElastic	Resposta à Notificação 2 permitindo a consolidação dos recursos.

Fonte: elaborado pelo autor.

Considerando o método de elasticidade, AutoElastic utiliza a replicação horizontal de máquinas virtuais (KOUKI et al., 2014). As operações de elasticidade consistem sempre na adição de uma máquina física com n máquinas virtuais ou a remoção de uma máquina física e suas máquinas virtuais. A inicialização de uma máquina virtual é um processo que demanda tempo (*boot* do sistema operacional) que é finalizada com a execução de um processo escravo. Esse processo automaticamente requisita a conexão com o processo mestre, completando o ciclo da elasticidade assíncrona. A Figura 11 demonstra o diagrama de sequência de uma operação de inicialização de uma máquina virtual. As ações elencadas

Figura 10: Funcionamento do processo mestre, um novo processo escravo e o Gerenciador AutoElastic para habilitar a Elasticidade Assíncrona.



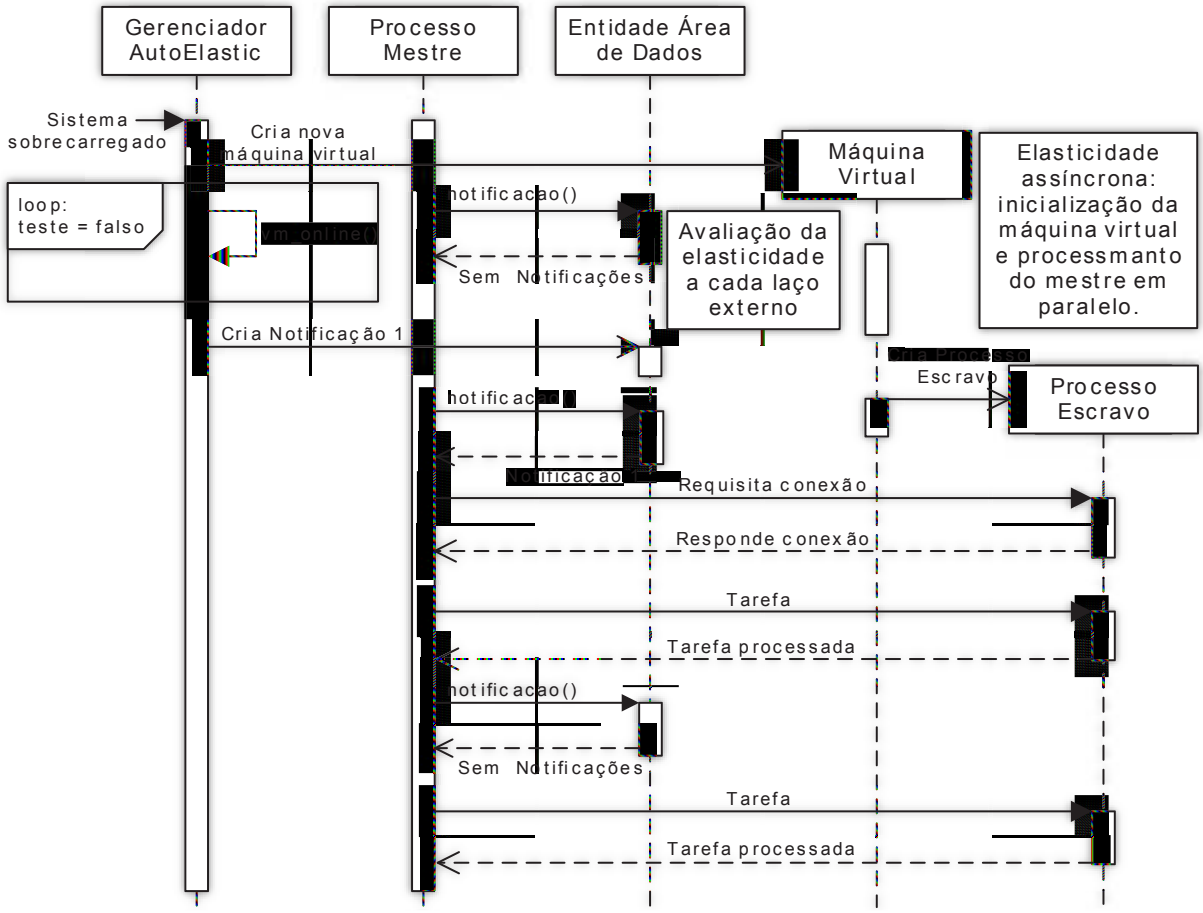
Fonte: elaborado pelo autor.

anteriormente e a replicação viabilizam a elasticidade assíncrona, uma vez que a execução da aplicação ocorre concomitantemente com a reconfiguração de recursos. A inicialização de novos recursos é realizada pelo Gerenciador AutoElastic que, somente após as novas máquinas virtuais terminarem seu processo de inicialização, realiza o processo de notificação de novos recursos. Quanto à consolidação de recursos, o grão de trabalho é sempre uma máquina física e não uma máquina virtual. Isso se deve ao fato do uso eficiente dos recursos (não usar parcialmente os núcleos disponíveis) e melhor gerenciamento no consumo de energia elétrica. Em especial, Baliga et al. (2011) afirmam que o número de máquinas virtuais em uma máquina física não é um fator tão influente para consumo energético, mas sim se a máquina física está ligado ou não.

4.4 Modelo de Elasticidade

Esta Seção apresenta as técnicas de monitoramento e lançamento da elasticidade utilizadas pelo modelo AutoElastic. A Subseção 4.4.1 apresenta como é definida a carga do ambiente e como esse valor é utilizado para realizar o lançamento de operações de elasticidade. Na Subseção 4.4.2 é apresentada a abordagem Live Thresholding desenvolvida.

Figura 11: Diagrama de sequência quando detectado uma carga do sistema acima do *threshold* superior, desencadeando a adição de uma nova máquina virtual no ambiente de nuvem.



Fonte: elaborado pelo autor.

4.4.1 Definição de Carga e Lançamento da Elasticidade

Assim como Imai, Chestna e Varela (2012) e Chiu e Agrawal (2010), a atividade de monitoramento do Gerenciador AutoElastic é realizada periodicamente. Considerando que as aplicações de HPC são conhecidas como sendo de computação intensiva, AutoElastic está se concentrando apenas na métrica de CPU, particularmente, na carga de CPU de cada máquina virtual. O Gerenciador captura periodicamente valores de carga de CPU de cada máquina virtual executando processos escravos e aplica um cálculo de séries temporais, considerando também valores coletados anteriormente, para obter a carga geral, aqui chamada de carga de processamento do sistema. Cada ciclo de monitoramento consiste em uma observação, em que os valores são coletados e armazenados (permitindo o uso de informações temporais) para serem utilizados posteriormente. Do mesmo modo como plataformas de nuvem comerciais como Amazon AWS e Microsoft Azure, AutoElastic implementa a técnica de elasticidade utilizando regras baseadas em *thresholds* (LORIDO-BOTRAN; MIGUEL-ALONSO; LOZANO, 2014). O valor resultante do cálculo da carga de processamento do sistema é utilizado em comparação com os *thresholds* superior e

inferior para a tomada de decisões de elasticidade (JAMSHIDI; AHMAD; PAHL, 2014b). A esses *thresholds*, podem ser atribuídos valores entre 0 e 1, representando níveis de carga de 0% a 100% respectivamente.

A carga de processamento do sistema é calculada através da Equação 4.1, em que a função chamada $CPS(o)$ calcula este valor na observação o. Esta função consiste em uma média aritmética da carga de CPU de cada uma das máquinas virtuais, em que v é a quantidade de máquinas virtuais executando um processo escravo. A carga de CPU de cada máquina virtual é calculada através da função $SES(e, o)$, representada na Equação 4.2. Esta equação aplica o método SES do modelo ARIMA, realizando a suavização exponencial simples da carga de CPU, representada por $CPU(e, o)$, de uma determinada máquina virtual e, considerando todo o histórico de observações, sendo o a observação atual. Por fim, o valor de CPS obtido é utilizado em comparação com os *thresholds* superior T_s e inferior T_i disparando ações de elasticidade se ocorrerem violações. A Figura 12 apresenta o modelo de elasticidade reativa empregado por AutoElastic utilizando a carga de processamento do sistema. Embora o modelo contemple três notificações, essa figura apresenta somente as notificações que são disparadas pelo Gerenciador AutoElastic. Em relação aos parâmetros utilizados para o cálculo das Equações 4.1 e 4.2, a Tabela 6 sumariza todos os parâmetros e funções apresentadas.

Figura 12: Modelo de elasticidade reativa AutoElastic baseado em *thresholds* superior (T_s) e inferior (T_i).

REGRA1: se CONDIÇÃO1 então AÇÃO1
REGRA2: se CONDIÇÃO2 então AÇÃO2

CONDIÇÃO1: CPS(o) $> T_s$, em que o é a última observação de monitoramento.
CONDIÇÃO2: CPS(o) $< T_i$, em que o é a última observação de monitoramento.

AÇÃO1: Aloca um novo nó e aloca n máquinas virtuais nele, em que n é a quantidade de núcleos de processamento do nó.
AÇÃO2: Finaliza as máquinas virtuais que estão executando dentro do nó e remove este nó após isso.

Fonte: elaborado pelo autor.

$$CPS(o) = \frac{\sum_{e=0}^{v-1} SES(e, o)}{v} \tag{4.1}$$

$$SES(e, o) = \begin{cases} \frac{CPU(e,o)}{2} & se\ o = 0 \\ \frac{SES(e,o-1)}{2} + \frac{CPU(e,o)}{2} & se\ o \neq 0 \end{cases} \tag{4.2}$$

em que $CPU(e, o)$ se refere a carga de CPU da máquina virtual e na observação de monitoramento de índice o.

SES opera com o conceito de *Aging* (TANENBAUM, 2003), implementando o método

Tabela 6: Descrição dos parâmetros e funções utilizados nas equações de definição do mecanismo de elasticidade.

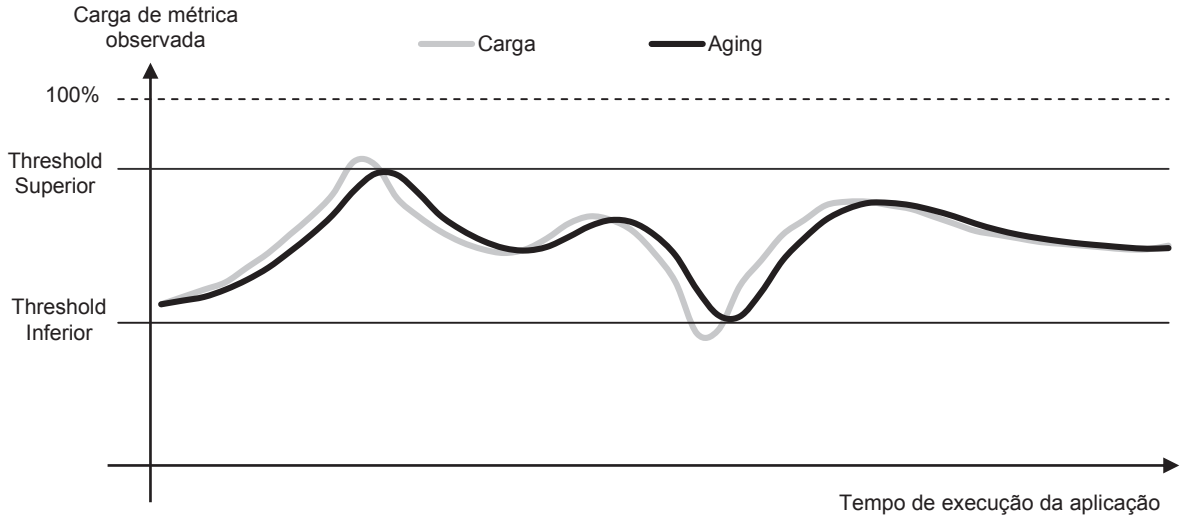
Parâmetro/Função	Descrição
e	Índice de uma máquina virtual executando um processo escravo.
o	Observação de carga mais atual do Gerenciador AutoElastic.
v	Quantidade de máquinas virtuais executando processos escravos.
T_i	Valor do *threshold* inferior.
T_s	Valor do *threshold* superior.
$CPU(e, o)$	Carga de CPU da máquina virtual e na observação o.
$SES(e, o)$	Carga de processamento da máquina virtual e na observação o.
$CPS(o)$	Carga de processamento do sistema na observação o.

Fonte: elaborado pelo autor.

SES apresentado na seção 2.3, empregando uma suavização exponencial simples em que a última medida tem a maior influência sobre o índice de carga. A ideia de *Aging* é empregada no modelo AutoElastic buscando tratar situações de picos de carga, especialmente para evitar ações de elasticidade ocasionadas por falsos-negativos ou falsos-positivos. Por exemplo, assumindo um *threshold* superior de 80%, valores de monitoramento de CPU para uma única máquina virtual tais como 82, 78, 81, 80, 77 e 93 (valor mais recente), tem-se $SES(e, o) = \frac{1}{2}.93 + \frac{1}{4}.77 + \frac{1}{8}.80 + \frac{1}{16}.81 + \frac{1}{32}.78 + \frac{1}{64}.82 = 84.53$. Esse valor permite a reconfiguração de recursos. Em contraste com AutoElastic, a abordagem de Imai, Chestna e Varela (2012) espera por x observações consecutivas fora da margem do *threshold*. Neste caso, o uso de x igual a 2, 3 ou 4 não resultaria em ações de elasticidade. Em adição a esse cenário de falso-negativo, um falso-positivo ocorre quando uma aplicação apresenta um pico ou queda de forma inesperada, desencadeando ações de elasticidade desnecessárias. Assim, a maneira de AutoElastic evitar *thrashing* contempla a utilização de séries temporais e da técnica suavização exponencial simples para suavizar possíveis ruídos nas observações de carga. A Figura 13 demonstra uma situação em que ações de elasticidade ocorrem quando é considerado apenas a carga real da métrica monitorada. Entretanto, com a utilização do conceito de *Aging* os picos e quedas da métrica monitorada são amortizados levando-se em conta os valores coletados nas observações passadas, evitando operações desnecessárias.

Do ponto de vista da abordagem de *thresholds*, AutoElastic contempla a abordagem tradicional de *thresholds* fixos e propõe uma nova abordagem, chamada de Live Thresholding, em que seus valores são recalculados em tempo de execução. Utilizando a abordagem de *thresholds* fixos, devem ser informados previamente quais serão seus valores superior e inferior antes do início do monitoramento, não podendo ser realizadas alterações nesses valores durante a execução da aplicação. Esses valores são mantidos fixos durante todo o período de monitoramento. Essa abordagem é comum em plataformas de nuvens comerciais como Amazon AWS e Microsoft Azure em que o usuário deve previamente definir os *thresholds* para que o sistema de monitoramento execute as operações de elas-

Figura 13: Cenário em que a alocação e desalocação de recursos são evitadas com o uso da técnica de *Aging* em momentos de pico e queda dos valores da métrica monitorada.



Fonte: elaborado pelo autor.

ticidade (LORIDO-BOTRAN; MIGUEL-ALONSO; LOZANO, 2014). A seguir a técnica de Live Thresholding é apresentada em mais detalhes.

4.4.2 Live Thresholding

Visando uma diferente abordagem dos *thresholds* fixos, em que há necessidade de parametrização do sistema pelo usuário, foi desenvolvida uma nova abordagem chamada de Live Thresholding, em que o próprio Gerenciador AutoElastic automaticamente calcula os *thresholds* a serem utilizados em cada observação. O Algoritmo 1 apresenta os principais procedimentos e testes realizados em cada observação de monitoramento. O procedimento na linha 3 é responsável pela coleta de dados de todos os recursos ativos na nuvem. Assim, esses dados são utilizados para calcular a carga do sistema (CPS) detalhada na Subseção 4.4.1. Ao invés da utilização de *thresholds* fixos previamente parametrizados para a tomada de decisões de elasticidade, é realizada a adaptação dinâmica dos *thresholds* inferior e superior, os quais são inicializados com os valores 0 e 1, respectivamente. Live Thresholding possui dois procedimentos específicos: adapta_thresholds() e recalcula_thresholds(). A primeira é computada a cada observação de monitoramento, enquanto que a segunda é chamada apenas quando uma ação de elasticidade é realizada. Ambas foram modeladas para aumentar a reatividade da elasticidade do ambiente, buscando oferecer valores competitivos de tempo de execução da aplicação (*tempo*) e custo ($tempo \times recursos$) quando comparados com os cenários da tradicional técnica de *thresholds* fixos e a não utilização de elasticidade.

Algoritmo 1: Ciclo de monitoramento considerando procedimentos para realizar adaptação e recálculo dos *thresholds*.

```
   Entrada: observação = 0; T_s= 1; T_i= 0;
 1 início
 2     enquanto (aplicação paralela executando) faça
 3         coleta_dados_monitoramento();
 4         carga = CPS(observação);
 5         adapta_thresholds(T_i, T_s, carga);
 6         se (carga > T_s) então
 7             se (SLA permite adicionar recursos) então
 8                 adiciona_recursos();
 9                 recalcula_thresholds(T_i, T_s, carga);
10             fim
11         fim
12         senão se (carga < T_i) então
13             se (SLA permite remover recursos) então
14                 remove_recursos();
15                 recalcula_thresholds(T_i, T_s, carga);
16             fim
17         fim
18         observação = observação + 1;
19         aguarda_tempo();
20     fim
21 fim
```

4.4.2.1 adapta_thresholds(): Redefinindo os *Thresholds* a cada Observação de Monitoramento

Este procedimento possui três parâmetros: T_i, T_s (ambos entrada/saída) e *carga* (somente entrada). Primeiramente, é computada a variação da carga do sistema considerando a carga atual ($CPS(o)$) e a carga da observação anterior ($CPS(o-1)$), atribuindo esse valor para ΔCPS (Equação 4.3). ΔCPS define qual *threshold* será atualizado: (i) se ΔCPS é negativo, a carga do sistema está diminuindo, então T_i é recalculado para lidar com essa situação rapidamente; (ii) se ΔCPS é positivo, a carga do sistema está aumentando, então T_s é atualizado para resolver essa situação; (iii) se ΔCPS é igual a 0, adaptações de *thresholds* não ocorrem devido à carga do sistema ter se mantido estável. A Equações 4.4 e 4.5 demonstram como os novos valores dos *thresholds* são computados, em que Min e Max retornam respectivamente o menor e maior valor passados por parâmetro. As igualdades nessas equações se referem a atribuições comumente utilizadas em abstrações de linguagens de programação, em que a variável do lado direito da sentença é utilizada para atualizar a parte da esquerda. Contemplando que T_s diminui quando atualizado, esse *threshold* possui um limite inferior igual a 0. Da mesma maneira, um limite superior de 1 é utilizado ao computar o novo valor de T_i, o qual aumenta quando atualizado.

$$\Delta CPS = CPS(o) - CPS(o-1) \tag{4.3}$$

$$T_i = Min(T_l + |\Delta CPS|, 1) \tag{4.4}$$

$$T_s = Max(T_u - \Delta CPS, 0) \tag{4.5}$$

4.4.2.2 recalcula_thresholds(): Redefinindo os Thresholds Quando um Deles é Violado

Como pode ser visto no Algoritmo 1, os valores 0 e 1 servem para inicializar os *thresholds*. Como estratégia inicial para definir o recalcula_thresholds(), o qual possui os mesmos parâmetros apresentados em adapta_thresholds(), é reatribuir aos *thresholds* seus valores padrões a cada ação de elasticidade. Essa estratégia equivale a reiniciar o Gerenciador AutoElastic, mantendo apenas os dados históricos da carga do sistema. Essa pode não ser a melhor estratégia para a reatividade da elasticidade pois os dados históricos de operações realizadas é perdido, não podendo ser utilizados para influenciar decisões futuras por parte do Gerenciador. Isso resulta nas seguintes questões:

(i) Como modelar outras estratégias levando em consideração o comportamento da aplicação?

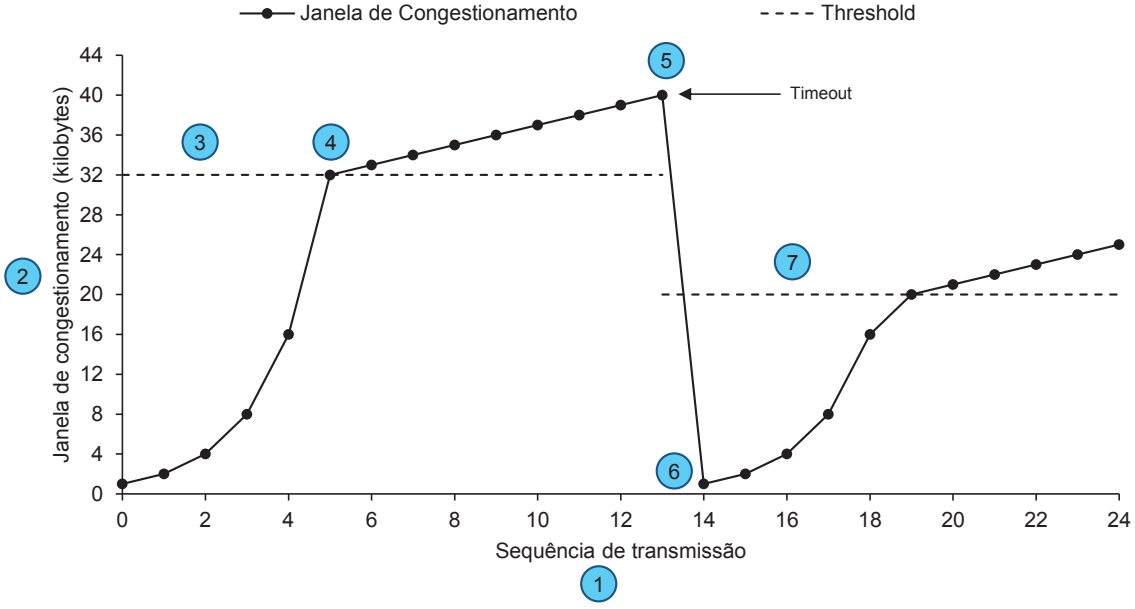
(ii) Existem benefícios em utilizar o valor da carga do sistema imediatamente anterior e posterior a uma reorganização de recursos para definir novos valores para os *thresholds*?

Buscando propor novas formas de recalcular os *thresholds*, foi analisado o algoritmo de congestionamento TCP o qual todas as implementações do TCP devem suportar (BING; YING-LAN; BAI, 2009; CHANGQING; QINGHUI; GUANGXING, 2005; TANENBAUM, 2003). O protocolo TCP utiliza uma janela de congestionamento que define a quantidade de bytes que o transmissor pode transmitir em um momento específico. Quando uma conexão é estabelecida, o segmento máximo na conexão é atribuído a essa janela. A cada envio de dados em que o *ack* é recebido antes de estourar o *timer*, o algoritmo TCP dobra o valor do tamanho da janela de congestionamento. A janela de congestionamento continua crescendo exponencialmente até atingir um *threshold* de congestionamento. Após superar esse *threshold*, o novo valor da janela é incrementado linearmente pelo segmento máximo em cada ciclo de envio. Então, em cada estouro no tempo de envio, esse *threshold* é ajustado para a metade do valor da janela de congestionamento atual, e a janela de congestionamento é redefinida para o segmento máximo.

A Figura 14 ilustra um exemplo do algoritmo de congestionamento TCP, demonstrando pontos de analogia com o conceito de elasticidade em nuvem baseada em *thresholds*. Considerando o procedimento recalcula_thresholds(), os pontos 5, 6 e 7 dessa figura são os mais importantes. Dessa maneira, o é o índice da observação de monitoramento após uma ação de elasticidade (ponto 6) e, $CPS(o)$ e $CPS(o-1)$ a carga do sistema nas

observações o e $o-1$ (ponto 5). Assim, foram investigadas 6 abordagens diferentes A_z ($A_z|z \in \{a,b,c,d,e,f\}$) para tratar a adaptatividade dos *thresholds* após uma ação de elasticidade (ponto 7): ao violar T_i podem ser aplicados A_a, A_b ou A_c para computar o novo valor para T_i, enquanto T_s é redefinido para 1; ao violar T_s podem ser aplicados A_d, A_e ou A_f, enquanto T_i é reiniciado para 0. O Gerenciador AutoElastic sempre utiliza uma combinação fixa de uma abordagem quando violando T_i e outra para T_s. Essa combinação resulta em uma notação nomeada LT_{xy}, em que x (x é A_a, A_b ou A_c) e y (y é A_d, A_e ou A_f) se referem a uma possibilidade particular para os *thresholds* inferior e superior, respectivamente[1]. Em resumo, recalcula_thresholds() depende em qual *threshold* foi violado: se T_i, seu novo valor é computado pela Equação 4.6 e T_s nessa situação volta para 1; se T_s, seu novo valor é calculado pela Equação 4.7 e T_i é reiniciado para 0.

Figura 14: Exemplo do algoritmo de congestionamento TCP. Baseado em seu comportamento, criou-se uma analogia para ser usada na discussão de elasticidade em nuvem, destacando 7 pontos: (1) e (2) referem-se a observação de monitoramento e carga do sistema, respectivamente; (3) *threshold* de elasticidade; (4) momento em que o Gerenciador AutoElastic detecta a necessidade de alocação de recursos, enquanto (5) representa o momento em que um novo recurso é disponibilizado para a aplicação; (6) esse ponto revela o impacto no balanceamento de carga, com a queda da carga do sistema; (7) após uma ação de elasticidade, o novo valor do *threshold* deve ser computado.



Fonte: adaptado de Tanenbaum (2003).

$$T_i = \begin{cases} 0 & para\ A_a \\ \frac{CPS(o)}{2} & para\ A_b \\ CPS(o-1) - \left|\frac{CPS(o-1)-CPS(o)}{2}\right| & para\ A_c \end{cases} \quad (4.6)$$

[1]Por simplicidade, em LT_{xy}, x e y serão representados apenas pelas letras minúsculas para identificar a abordagem adotada: A_a até A_f

$$T_s = \begin{cases} 1 & para\ A_d \\ CPS(o) + \frac{1-CPS(o)}{2} & para\ A_e \\ CPS(o-1) + \left|\frac{CPS(o-1)-CPS(o)}{2}\right| & para\ A_f \end{cases} \quad (4.7)$$

Considerando as diferentes abordagens, A_a e A_d simplesmente reiniciam os *thresholds* para os mesmos valores que eles foram inicializados no início do Algoritmo 1. A_b e A_e usam a carga do sistema após uma ação de elasticidade para redefinir os *thresholds*, enquanto A_c e A_f fazem isso considerando a carga do sistema antes e depois de uma entrega ou remoção de recursos. Como o principal objetivo da utilização de Lite Thresholding é a não necessidade de parametrização prévia do sistema, foram conduzidos experimentos com todas as possibilidades de LT_{xy} considerando quatro comportamentos de carga de trabalho da aplicação, apresentados na metodologia de avaliação. A avaliação revelou que os melhores resultados foram obtidos pela combinação LT_{ce}, sendo essa combinação selecionada para ser a abordagem definitiva do Live Thresholding. Detalhes referentes a esta avaliação podem ser vistos na Seção 6.1.

4.5 Modelo de Aplicação Paralela

AutoElastic explora paralelismo de dados em aplicações paralelas iterativas de passagem de mensagens. Em particular, a atual versão do modelo ainda tem a restrição de operar com aplicações do estilo de programação mestre-escravo. Embora trivial, esse estilo de programação é utilizado em diversas áreas, tais como algoritmos genéticos, técnica de Monte Carlo, transformações geométricas na computação gráfica, algoritmos de criptografia e aplicações no estilo SETI-at-home (RAVEENDRAN; BICER; AGRAWAL, 2011). Contudo, vale ressaltar que o *framework* permite aos processos existentes da aplicação HPC conheçam o identificador de novos processos criados, ou seja, permitindo também uma topologia de comunicação todos-para-todos. Em outras palavras, isso significa que AutoElastic suporta também aplicações como BSP e Divisão-e-Conquista.

Para desenvolvimento do *framework* de comunicação, foram investigadas as semânticas e sintaxe de ambas versões 1.0 e 2.0 do MPI. Enquanto o primeiro cria todos seus processos estaticamente no ponto inicial de execução, o segundo suporta criação dinâmica de processos e a reorganização da topologia de comunicação durante a execução da aplicação. Isso significa que MPI 2.0 é compatível para ambientes elásticos. Dessa maneira, AutoElastic possui como requisito que a aplicação paralela siga a semântica de primitivas que MPI 2.0 oferece. É importante frisar que o modelo não é feito focado em MPI 2.0, mas sim nas suas diretivas de comunicação. Assim, é possível implementar uma aplicação com qualquer biblioteca, desde que ela tenha as diretivas com a semântica daquelas

encontradas no MPI 2.0.

As aplicações paralelas de AutoElastic são projetadas segundo o modelo MPMD (*Multiple Program Multiple Data*) (WILKINSON; ALLEN, 2005), no qual o mestre tem um executável e os escravos outro. A ideia é dissociar processos da aplicação com diferentes objetivos, permitindo flexibilidade e tornando mais fácil a implementação da elasticidade. O Algoritmo 2 apresenta um pseudocódigo de uma aplicação iterativa suportada por AutoElastic. O processo mestre executa uma série de tarefas, capturando cada uma sequencialmente e paralelizando uma por uma para ser processada pelos processos escravos. Este comportamento pode ser observado no laço externo (linha 3). Atualmente, AutoElastic opera com as seguintes diretivas de comunicação, baseadas em MPI 2.0:

(i) publicação de porta de conexão;

(ii) procura por servidor, tomando como ponto de partida a porta de conexão;

(iii) requisição de conexão;

(iv) permissão de conexão;

(v) requisição de desconexão.

Algoritmo 2: Pseudo-linguagem do processo mestre.

```
   Entrada: tarefas e quantidade de processos inicial
   Saída: tarefas processadas
 1 início
 2     tamanho = mapeamento_inicial(portas);
 3     para (j=0; j < total_tarefas; j++) faça
 4         publica_portas(portas, tamanho);
 5         para (i=0; i < tamanho; i++) faça
 6             aceita_conexão(escravos[i], portas[i]);
 7         fim
 8         calcula_carga(tamanho, trabalho[j], intervalos);
 9         para (i=0; i < tamanho; i++) faça
10             tarefa = cria_tarefa(trabalho[j], intervalos[i]);
11             envia_assincrono(escravos[i], tarefa);
12         fim
13         para (i=0; i < tamanho; i++) faça
14             recebe_sincrono(escravos[i], resultados[i]);
15         fim
16         guarda_resultados(escravos[j], resultados);
17         para (i=0; i < tamanho; i++) faça
18             desconecta(escravos[i]);
19         fim
20         despublica_portas(portas);
21     fim
22 fim
```

Diferentemente da abordagem em que o processo mestre lança processo utilizando uma diretiva conhecida como *spawn()*, AutoElastic atua de acordo a abordagem de MPI

2.0 para suportar criação dinâmica de processos: comunicação ponto-a-ponto baseada em *Sockets* (LUSK, 1997). A execução de uma nova máquina virtual ocasiona a execução de um processo escravo, o qual requisita conexão com o processo mestre automaticamente, como apresentado no Algoritmo 3.

Algoritmo 3: Pseudo-linguagem do processo escravo.

```
   Entrada: tarefa
   Saída: tarefa processada
 1 início
 2 |   mestre = procura(endereço_mestre, nome);
 3 |   porta = cria_porta(endereço_IP, VM_id);
 4 |   enquanto (verdadeiro) faça
 5 |   |   requisita_conexão(mestre, porta);
 6 |   |   recebe_sincrono(mestre, tarefa);
 7 |   |   resultado = computa(tarefa);
 8 |   |   envia_assincrono(mestre, resultado);
 9 |   |   desconecta(mestre);
10 |   fim
11 fim
```

O método `mapeamento_inicial` (linha 2 do Algoritmo 2) é utilizada pelo processo mestre para verificar a configuração da execução, que define a configuração inicial de máquinas virtuais, um identificador e o endereço IP de cada processo. Levando em consideração essa informação, o mestre sabe a quantidade de escravos e cria portas para receber a conexão dos processos escravos. A comunicação ocorre de forma assíncrona, em que o mestre envia dados para os escravos de maneira não bloqueante mas recebe dados de retorno deles de maneira síncrona. De fato, aplicações baseadas em laços são convenientes para a implementação de elasticidade em nuvem devido ao fato da fácil reconfiguração da quantidade de recursos no início de cada iteração sem alterar a semântica da aplicação (SPINNER et al., 2014). Além disso, o laço de distribuição garante o estado de consistência global da aplicação. A transformação de uma aplicação não elástica em uma elástica através do *middleware* AutoElastic pode ser modelada no nível de PaaS por uma das seguintes estratégias:

(i) Polimorfismo pode ser utilizado para sobrescrever um método para fornecer elasticidade para aplicações orientadas ao objeto;

(ii) Tradutor código-para-código pode ser utilizado para inserir código de elasticidade entre as linhas 2 e 3;

(iii) Utilização de um *wrapper* para o método na linha 4 pode ser desenvolvido para linguagens procedurais, possibilitando a chamada de outro método que implemente o código de elasticidade.

É importante salientar que essa transformação ocorre em tempo de compilação quando o *middleware* AutoElastic é integrado à aplicação. Independente da estratégia, o código

requerido para a elasticidade é simples, como mostrado no Algoritmo 4. Primeiro, é necessário verificar se existe alguma notificação do Gerenciador AutoElastic na área de dados compartilhada. Se a Notificação 1 é recebida, o processo mestre lê as informações referentes aos novos escravos e sabe que deve aguardar a conexão deles. Em caso de receber uma Notificação 2, o processo mestre remove do seu grupo de processos aqueles pertencentes à máquina física que será consolidada. Após isso, o processo gera uma Notificação 3, liberando a remoção de recursos para o Gerenciador AutoElastic.

Algoritmo 4: Pseudo-código para gerenciar a elasticidade no processo mestre.

```
   Entrada: recursos
   Saída: reorganização de recursos
 1 início
 2 |   alterações = 0;
 3 |   se (Ação == 1) então
 4 |   |   alterações += adiciona_VMs();
 5 |   fim
 6 |   senão se (Ação == 2) então
 7 |   |   alterações -= remove_VMs();
 8 |   |   permissão_consolidação(); //Noficação 3
 9 |   fim
10 |   se (Ação == 1 ou Ação == 2) então
11 |   |   reorganiza_portas(portas);
12 |   fim
13 |   tamanho += alterações;
14 fim
```

Embora o foco inicial de AutoElastic seja aplicações mestre-escravo, a modelagem iterativa e o uso de diretivas de MPI 2.0 facilitam a inclusão de processos e o restabelecimento das conexões para uma nova topologia totalmente arbitrária. Em nível de implementação, é possível otimizar conexões e desconexões caso o processo persistir na lista de processos ativos. Essa atitude é pertinente principalmente sobre conexões TCP/IP, que usa um protocolo de três vias que sabidamente pode acarretar sobrecarga em aplicações HPC.

4.6 Métricas de Avaliação

A análise do estado da arte não apresenta *benchmarks* e metodologias específicas para avaliar sistemas com elasticidade. Neste sentido, foi visto um campo de trabalho para a proposição de novas métricas como uma contribuição para a área. Tendo isso em vista, para a avaliação dos resultados, foram definidas métricas para avaliação de questões de desempenho da aplicação bem como eficiência na utilização dos recursos. Ainda, foram definidas métricas para medir o consumo de recursos em nuvem para a execução da aplicação paralela e também um cálculo de custo baseando-se no consumo de recursos. As próximas subseções exploram cada uma das métricas definidas.

4.6.1 *Speedup* Elástico e Eficiência Elástica

Speedup (S) é uma conhecida medida de desempenho utilizada para calcular ganho de tempo entre diferentes execuções de uma determinada aplicação (HENNESSY; PATTERSON, 2011). Geralmente, este cálculo é utilizado para realizar a comparação entre o tempo gasto para executar uma aplicação em dois cenários: (i) com apenas um processo, representando uma execução sequencial, e (ii) com vários processos, representando uma execução paralela. Dessa maneira, a Equação 4.8 apresenta o *speedup* (S) como uma função do número de processos p, tais que $t(1)$ e $t(p)$ expressam os tempos de execução sequencial e paralelo. É importante notar que o *speedup* é uma medida livre de unidades. Além dessa métrica, também é possível medir a eficiência da execução de uma aplicação paralela (KUMAR, 2002). A Equação 4.9 denota a eficiência de um sistema paralelo, descrevendo a fração de tempo que é utilizada pelos processadores p para uma dada computação. Eficiência, algumas vezes é mais pertinente que o *speedup* devido a representar uma maneira fácil de observar o quão bem a paralelização está trabalhando.

$$S(p) = \frac{t(1)}{t(p)} \tag{4.8}$$

$$E(p) = \frac{S(p)}{p} \tag{4.9}$$

Com o objetivo de observar o ganho da elasticidade em nuvem para aplicações HPC, foram desenvolvidas duas métricas como extensão dos conceitos de *speedup* e eficiência: *Speedup* Elástico (SE) e Eficiência Elástica (EE). Ambas métricas SE e EE são exploradas de acordo com a elasticidade horizontal, em que instâncias de máquinas virtuais podem ser adicionadas ou removidas durante o processamento da aplicação, alterando a quantidade de processos (CPUs) disponíveis. Além disso, é considerado um ambiente homogêneo em que cada máquina virtual executa em 100% de um núcleo de processamento da CPU, não importando o modelo de implantação da nuvem (IaaS, PaaS ou SaaS). SE é calculado pela função $SE(n, i, s)$ de acordo com a Equação 4.10, em que n representa a quantidade inicial de máquinas virtuais enquanto s e i representam, respectivamente, os limites superior e inferior para a quantidade de máquinas virtuais definidos pelo SLA. t_e e t_{ne} se referem aos tempos de execução da aplicação executada em cenários com elasticidade e sem elasticidade, respectivamente. Aqui, t_{ne} é obtido com o menor número de máquinas virtuais possíveis (definido por i), fornecendo uma analogia com a execução sequencial do *speedup* padrão.

$$SE(n, i, s) = \frac{t_{ne}(i)}{t_e(n, i, s)} \tag{4.10}$$

Por sua vez, a função $EE(n, i, s)$ na representada pela Equação 4.11 calcula a eficiência elástica. Os parâmetros desta função são os mesmos da função SE. A eficiência representa o quão eficaz é o uso dos recursos, sendo este posicionado como denominador da fórmula. Diferentemente da Equação 4.9, os recursos aqui são maleáveis e por isso foi criado um mecanismo para alcançar um único valor de forma coerente. Para alcançar isso, EE assume a execução de um sistema de monitoramento que captura o tempo gasto em cada configuração de máquinas virtuais. A Equação 4.12 apresenta a métrica $Recursos$[2] utilizada no cálculo de EE indicando o uso de recursos da aplicação, em que $pt_e(j)$ é a fatia de tempo em que a aplicação foi executada com j máquinas virtuais. Logicamente, se operações de elasticidade não ocorrem, $pt_e(j)$ e $t_e(n, i, s)$ resultarão no mesmo valor para $j = n$. A Equação 4.11 apresenta um parâmetro n no numerador, que está multiplicando o *speedup* elástico. Ao contrário de $E(p)$ em que o menor número de recursos é 1 (execução sequencial) servindo como base de comparação, aqui esse valor é igual a n. Mais precisamente, isso ocorre pois $SE(n, n, n)$ é igual a 1 e $Recursos(n, n, n)$ é igual a n, então n no numerador retorna uma eficiência elástica de 100%.

$$EE(n, i, s) = \frac{ES(n, i, s) \times n}{Recursos(n, i, s)} \tag{4.11}$$

em que

$$Recursos(n, i, s) = \sum_{j=i}^{s} (j \times \frac{pt_e(j)}{t_e(n, i, s)}) \tag{4.12}$$

4.6.2 Modelo de Energia

Através da Eficiência Elástica não é possível inferir quanto ao consumo de recursos precisamente. A fim de fornecer uma maneira mais fácil de medir os recursos consumidos durante a execução da aplicação, foi definida uma métrica de avaliação calculada empiricamente chamada *Energia*. Essa métrica é baseada na relação próxima entre consumo energético e consumo de recursos apresentado por Orgerie, Assuncao e Lefevre (2014). A Equação 4.13 calcula um índice da utilização de recursos, com i e s com o mesmo significado descrito para a Equação 4.10, representando respectivamente a menor e a maior quantidade de máquinas virtuais permitida.

Aqui, são consideradas as mesmas ideias do modelo de custo empregado pela Amazon e Microsoft: eles consideram a quantidade de máquinas virtuais em cada unidade de tempo, que normalmente é definido em 1 hora. $pt_e(j)$ representa a fatia de tempo que a aplicação foi executada com j máquinas virtuais. Então, a unidade de tempo depende da medida de pt_e (minutos, segundos ou milissegundos, e assim por diante) em que a ideia final é somar a quantidade de máquinas virtuais utilizadas em cada unidade de tempo. Por

[2]Por questões de facilidade, *Recursos* será chamado de RE na avaliação apresentada na Seção 6.2.

exemplo, considerando uma unidade de tempo em minutos e um tempo de execução total da aplicação de 7 minutos, o seguinte histograma pode ser obtido: 1 minutos (2 máquinas virtuais), 1 minuto (2 máquinas virtuais), 1 minuto (4 máquinas virtuais), 1 minuto (4 máquinas virtuais), 1 minuto (2 máquinas virtuais), 1 minuto (2 máquinas virtuais) e 1 minuto (2 máquinas virtuais). Por fim, 2 máquinas virtuais foram utilizadas durante 5 minutos (recurso parcial igual a 10), e 4 máquinas virtuais em 2 minutos (recurso parcial igual a 8), totalizando o número 18 para a Equação 4.13. Assim, a *Energia* analisa o uso de i até s máquinas virtuais, levando em consideração o tempo de execução parcial em cada organização da infraestrutura. O índice de energia é pertinente para comparações entre diferentes execuções elásticas variando i e s.

$$Energia(i, s) = \sum_{j=i}^{s}(j \times pt_e(j)) \tag{4.13}$$

4.6.3 Modelo de Custo

Para estimar a viabilidade da elasticidade em diferentes situações, foi definida uma métrica que calcula o custo da execução através da multiplicação do tempo total de execução da aplicação pela energia utilizada para a execução, conforme Equação 4.14. A noção de custo significa uma adaptação do custo de uma computação paralela (WILKINSON; ALLEN, 2005), agora com suporte a ambientes elásticos. A ideia final consiste em obter um custo melhor quando utilizando a elasticidade em comparação com uma execução com a quantidade de recursos fixa. Em outras palavras, uma configuração pode ser classificada como ruim, se é capaz de reduzir o tempo de execução pela metade com elasticidade, mas gasta quatro vezes mais energia, aumentando os custos. Portanto, considerando os valores da função de custo em ambientes elásticos e não elásticos, o objetivo é preservar a verdade da Desigualdade 4.15.

$$Custo = Tempo\_Aplicação \times Energia \tag{4.14}$$

então, o objetivo é obter

$$Custo_\alpha \leq Custo_\beta \tag{4.15}$$

em que α e β se referem a execução de da aplicação com a elasticidade habilitada e desabilitada, respectivamente.

4.7 Considerações Parciais

Este capítulo apresentou o modelo AutoElastic definindo decisões de projeto para seu desenvolvimento, sua arquitetura e o modelo de aplicação suportada. Além disso foram apresentados o modelo de elasticidade e métricas definidas para avaliação de aplicações elásticas. Considerando o diferencial do modelo, podem ser citadas as seguintes contribuições:

- Arcabouço para execução de aplicações HPC iterativas de maneira elástica;
- Elasticidade Assíncrona;
- Técnica de *Aging* no tratamento de picos na avaliação da carga;
- Métricas de avaliação de desempenho e eficiência para aplicações elásticas;
- Técnica de Live Thresholding.

AutoElastic define uma arquitetura de nuvem homogênea composta por uma área de dados compartilhada que permite a comunicação entre todos os componentes. Além disso, AutoElastic apresenta um Gerenciador e uma política de notificações que permitem que a elasticidade seja executada de forma assíncrona e transparente para a aplicação. Este gerenciador realiza o monitoramento dos recursos aplicando a técnica de *Aging* para a suavização da carga de computação de todo ambiente, evitando disparar ações em momentos de picos repentinos. Além disso, foram definidas novas métricas de avaliação de desempenho e eficiência voltadas para aplicações elásticas, estendendo os conceitos das conhecidas métricas de *speedup* e eficiência. Por fim, através de uma análise foi possível identificar e desenvolver melhorias na técnica de *thresholds* fixos, comumente usada. Isso resultou no desenvolvimento de uma nova técnica, aqui chamada de Live Thresholding, que permite que o próprio mecanismo de elasticidade calcule o valor dos *thresholds* em tempo de execução, sem que haja a necessidade de configuração de parâmetros. A próxima seção apresenta a metodologia para a avaliação do modelo, apresentando protótipos de desenvolvidos e detalhes do ambiente de nuvem utilizado.

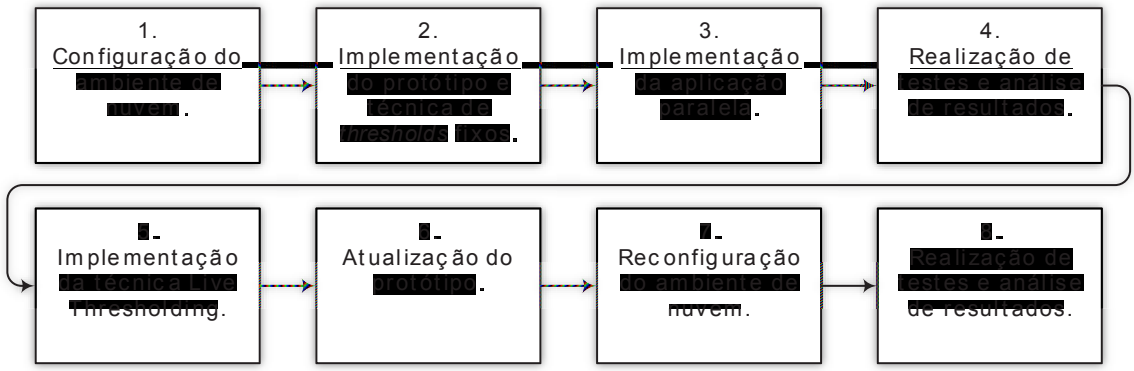
5 METODOLOGIA DE AVALIAÇÃO

Este capítulo apresenta a metodologia utilizada para avaliar o modelo AutoElastic, bem como os protótipos desenvolvidos e a infraestrutura utilizada. Para a avaliação do modelo, foi desenvolvida uma aplicação iterativa para executar no ambiente com diferentes configurações de comportamentos de carga e *thresholds*. Além do tempo de execução da aplicação, a ideia consiste em analisar a reatividade da elasticidade e os custos em termos de infraestrutura para atingir um tempo de execução em particular. A Seção 5.1 descreve as etapas de desenvolvimento realizadas. Em seguida, a implementação dos protótipos e a infraestrutura utilizada são detalhadas na Seção 5.2. Na Seção 5.3 a aplicação paralela utilizada nos testes é apresentada em detalhes, bem como os comportamentos de carga modelados. Já na Seção 5.4 são definidos os parâmetros e cenários avaliados. Por fim, as considerações parciais do capítulo são realizadas na Seção 5.5.

5.1 Etapas de Desenvolvimento

As etapas específicas relacionadas ao desenvolvimento da pesquisa são apresentadas na Figura 15, demonstrando a evolução do trabalho ao longo da realização da pesquisa. Em particular, destaca-se o desenvolvimento da nova técnica de Live Thresholding e adaptações do modelo realizadas nas etapas 5 e 6. Estas adaptações foram possíveis através de melhorias para o modelo de elasticidade, identificadas na análise de resultados preliminares realizadas na etapa 4.

Figura 15: Sequência de etapas do desenvolvimento da pesquisa, baseadas no ciclo de etapas da Figura 3 da Seção 1.4.



Fonte: elaborado pelo autor.

Seguindo todas as etapas, inicialmente foi instalado, configurado e implantado um ambiente de nuvem com as novas ideias de AutoElastic. Em seguida, foi realizado o desenvolvimento do protótipo do Gerenciador AutoElastic, no qual foi implementada a técnica de elasticidade baseada em *thresholds* fixos. Além disso, foi desenvolvida uma aplicação paralela para ser executada no ambiente de nuvem com diferentes configurações

de *thresholds*. Com os protótipos desenvolvidos e o ambiente configurado foram realizadas avaliações preliminares. Após essas avaliações e análise de resultados, notou-se uma oportunidade de melhoria na estratégia de elasticidade. Dessa maneira, foi modelada a nova técnica de elasticidade Live Thresholding. Tendo isso em vista, realizaram-se novas implementações e atualizações do protótipo a fim de contemplar esta nova técnica. Por fim, foi realizada uma atualização no ambiente de nuvem para uma versão mais recente e, em seguida, foram realizados novas avaliações e analisados os novos resultados.

5.2 Implementação e Infraestrutura

Para a realização de experimentos no ambiente de nuvem, foi desenvolvido um protótipo em Java do Gerenciador AutoElastic. Esse protótipo contemplou as técnicas de *thresholds* fixos e Live Thresholding, além de implementar a técnica de *Aging* e Elasticidade Assíncrona. Ainda, para conexão com o Front-End da nuvem, foi utilizada a API fornecida, permitindo o gerenciamento da nuvem, bem como do próprio servidor.

5.2.1 Protótipo Desenvolvido

Para a realização da avaliação do modelo AutoElastic, foi implementado um protótipo do Gerenciador AutoElastic utilizando a linguagem Java. Este protótipo consiste em um gerenciador que realiza as operações de monitoramento e elasticidade dos recursos de uma nuvem OpenNebula. O controle e monitoramento do ambiente de nuvem é realizado através da API Java fornecida pelo próprio *middleware* OpenNebula, que permite que sejam obtidas informações sobre os recursos do ambiente, além de permitir que esses recursos sejam gerenciados possibilitando que máquinas virtuais sejam adicionadas ou removidas do ambiente. Utilizando a interface gráfica fornecida pelo *middleware* de nuvem, foram configurados dois modelos de máquinas virtuais para diferenciar a função que o processo da aplicação irá empenhar no início de sua execução. Dessa maneira, foi criado um modelo para o processo mestre e um outro modelo para os processos escravos. As identificações destes modelos são passadas para o Gerenciador AutoElastic, que utiliza os modelos para o lançamento de novas máquinas virtuais. Além destes parâmetros, é possível informar ao gerenciador qual técnica de elasticidade e quais os *thresholds* serão utilizados antes do início do monitoramento, bem como o intervalo de monitoramento. Além dos parâmetros iniciais, o Gerenciador AutoElastic possibilita que seja informado o SLA através de um arquivo XML que respeite o padrão WS-Agreement[1]. Esse arquivo é passado como parâmetro ao gerenciador contendo a quantidade máxima e mínima de recursos para executar a aplicação. Caso esse arquivo não seja informado, o gerenciador considera a quantidade inicial de nós como a quantidade mínima, enquanto que para a

[1]https://www.ogf.org/documents/GFD.192.pdf

quantidade máxima é considerado o dobro da quantidade mínima.

Dentro do ambiente OpenNebula, foi configurada uma área de dados compartilhada utilizando NFS, a qual todos os recursos (máquinas físicas e virtuais) possuem acesso. Através desta área são compartilhados os arquivos fonte da aplicação paralela para que a aplicação seja acessada e executada pelas máquinas virtuais após sua inicialização. Utilizando esta área de dados compartilhada, as notificações do modelo AutoElastic foram implementadas utilizando arquivos no formato texto. O Gerenciador AutoElastic e o processo mestre escrevem e leem arquivos de texto, contendo as informações das notificações. O acesso a esta área pelo Gerenciador AutoElastic é realizado através de SSH (*Secure Shell*), possibilitando que o gerenciador possa ser executado fora do ambiente de nuvem.

5.2.2 Infraestrutura de Nuvem

O ambiente selecionado para a realização dos testes é o laboratório C01 413 localizado no Programa de Pós-Graduação em Computação Aplicada (PIPCA) da Universidade do Vale do Rio dos Sinos. Este laboratório possui 18 equipamentos homogêneos com processadores de dois núcleos de 2.9 GHz e 4 GB de memória, interconectados através de uma rede 100 Mbps. Desses equipamentos, foram selecionados 11 para a configuração da plataforma de nuvem OpenNebula versão 4.12.1, dos quais em um deles foi instalado o servidor OpenNebula para servir como o Front-End da nuvem. Já os 10 equipamentos restantes foram configurados como nós OpenNebula para receberem e executarem máquinas virtuais. Ainda, o servidor SSH e a área de dados compartilhada foram configurados no Front-End, o qual também serve como repositório de arquivos e imagens de máquinas virtuais.

5.3 Aplicação Paralela

A aplicação utilizada nos testes calcula a aproximação para a integral do polinômio $f(x)$ num intervalo fechado $[a, b]$. Para tal, foi implementado o método de Newton-Cotes para intervalos fechados conhecido como Regra do Trapézio Repetida (COMANESCU, 2012). A fórmula de Newton-Cotes pode ser útil se o valor do integrando é dada em pontos igualmente espaçados. Considere a partição do intervalo $[a, b]$ em s subintervalos iguais, cada qual de comprimento h ($[x_i, x_{i+1}]$, para $i = 0, 1, 2, ..., s - 1$). Assim, $x_{i+1} - x_i = h = \frac{b-a}{s}$. Dessa forma, pode-se escrever a integral de $f(x)$ como sendo a soma das áreas dos s trapézios contidos dentro do intervalo $[a, b]$ como apresentado na Equação 5.1. A Equação 5.2 demonstra o desenvolvimento de uma integração numérica de acordo com fórmula de Newton-Cotes. Os valores x_0 e x_s na Equação 5.2 são iguais a a e b, respectivamente. Nesse contexto, s representa a quantidade de subintervalos. Sendo assim, existem $s + 1$ equações $f(x)$ simples para se obter o resultado final da integral

numérica. O processo mestre deve distribuir essas $s + 1$ equações entre os processos escravos. Logicamente, alguns escravos podem receber mais trabalho do que outros quando $s + 1$ não é inteiramente divisível pela quantidade de processos escravos. Como s define a quantidade de subintervalos, e consequentemente a quantidade de equações para computar a integral, quanto maior for esse parâmetro, maior é a carga computacional para atingir o resultado final para uma equação em particular. A carga de trabalho recebida pelo processo mestre consiste em uma lista de equações e seus parâmetros, incluindo o intervalo $[a, b]$ para cada uma, enquanto o retorno é a mesma quantidade de valores de integração numérica.

$$\int_a^b f(x)\,dx \approx A_0 + A_1 + A_2 + A_3 + ... + A_{s-1} \tag{5.1}$$

em que A_i = area do trapezoide i, com $i = 0, 1, 2, 3, ..., s - 1$.

$$\int_a^b f(x)\,dx \approx \frac{h}{2}[f(x_0) + f(x_s) + 2.\sum_{i=1}^{x-1} f(x_i)] \tag{5.2}$$

A ideia de usar diferentes comportamentos para a mesma aplicação HPC é amplamente explorada na literatura para observar como a carga de entrada pode impactar em pontos de saturação, gargalos, e adição e remoção de recursos (ZHANG; SUN; INOGUCHI, 2008; MAO; HUMPHREY, 2011; ISLAM et al., 2012). Buscando analisar diferentes comportamentos, o parâmetro s foi utilizado para modelar quatro comportamentos de carga de computação: Constante, Crescente, Decrescente e Onda. Nesse sentido, para cada conjunto de equações enviadas para processamento para o processo mestre, em cada iteração (uma equação é selecionada para cálculo) o parâmetro s é recalculado individualmente, modelando um determinado comportamento de carga. Por exemplo, s pode ser igual a $1x$ para a primeira equação, $2x$ para a segunda, $3x$ para a terceira e assim por diante para gerar o comportamento Crescente. As Tabelas 7 e 8 apresentam respectivamente as funções para o cálculo do valor de s para a geração de cada comportamento de carga e o modelo de entrada de dados utilizado nos testes, em que $s = carga(x)$. O parâmetro $\$iteracoes$ na Tabela 8 significa a quantidade de equações que serão geradas para o processamento, resultando no mesmo número de integrações numéricas. Ainda, o polinômio para cada uma dessas equações é o mesmo $(5x^5 + x^2 + x)$ devido ao fato de não haver importância no polinômio selecionado, pois o foco real são as variações do processamento e não o resultado final da integração numérica.

A Figura 16 apresenta graficamente uma representação de cada comportamento de carga. O eixo x expressa a iteração (cada iteração representa uma equação que será calculada, dividida e distribuída pelo processo mestre), enquanto o eixo y representa a respectiva carga de processamento para aquela iteração (valor de s). Novamente, a carga é definida pela quantidade de subintervalos s entre os limites a e b, que nos experimentos são 1 e 10, respectivamente. Quanto maior a quantidade de subintervalos, maior é a

Tabela 7: Funções para expressar os diferentes comportamentos de carga. Em *carga(x)*, x é o índice da equação que será processada.

Carga	Função de Carga	Parâmetros			
		v	w	t	z
Constante	$carga(x) = w$	-	500000	-	-
Crescente	$carga(x) = x * t * z$	-	-	0,2	500
Decrescente	$carga(x) = w - (x * t * z)$	-	1000000	0,2	500
Onda	$carga(x) = v * z * sen(t * x) + v * z + w$	1	500	0,00125	500000

Fonte: elaborado pelo autor.

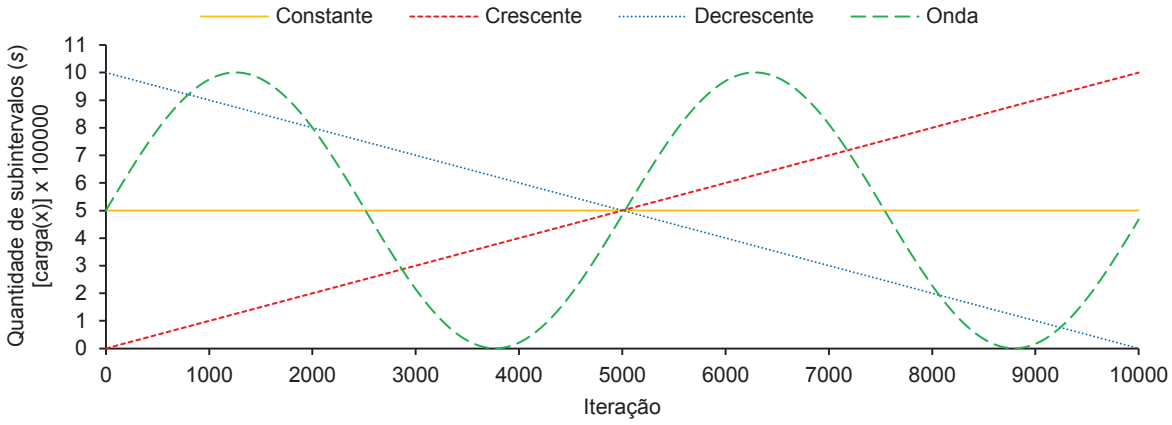
Tabela 8: Modelo de entrada de dados para os comportamentos de carga.

Modelo	Constante	Crescente	Decrescente	Onda
Polinômio	+;5;x^ 5;+;x^ 2;+;x^ 1	+;5;x^ 5;+;x^ 2;+;x^ 1	+;5;x^ 5;+;x^ 2;+;x^ 1	+;5;x^ 5;+;x^ 2;+;x^ 1
$a, $b	1,10	1,10	1,10	1,10
carga	CONSTANTE	CRESCENTE	DECRESCENTE	ONDA
$iteracoes	10000	10000	10000	10000
$v; $w; $t; $z	0;500000;0;0	0;0;0,2;500	0;1000000;0,2;500	1;500;0,00125;500000

Fonte: elaborado pelo autor.

quantidade de equações a serem calculadas pelos processos escravos, e consequentemente maior é a carga de processamento.

Figura 16: Visão gráfica dos comportamentos de carga. Cada iteração corresponde a uma equação a ser calculada e a quantidade de subdivisões entre o intervalo $[a, b]$ que será utilizada para o cálculo.



Fonte: elaborado pelo autor.

5.4 Parâmetros, Thresholds e Métricas

Como discutido na Seção 3.4, a elasticidade é oferecida através de diferentes estratégias. No geral, as abordagens oferecem a elasticidade de maneira automática, porém necessitando que seja pré-configurado o mecanismo de gerenciamento de recursos. A maioria dos trabalhos emprega a elasticidade de forma horizontal e reativa, em que a principal métrica avaliada é a carga de CPU. As técnicas mais utilizadas são baseadas

em *thresholds*, em que devem ser configurados valores que influenciam os algoritmos de tomada de decisão. Destas técnicas, podem destacar-se os seguintes *thresholds* utilizados: 50%, 70%, 75%, 80% e 90%. Essas técnicas e configurações foram levadas em consideração no modelo para a definição do algoritmo de elasticidade. Buscando avaliar o impacto de diferentes configurações de *thresholds* no desempenho e consumo de recursos da aplicação paralela, foram definidos três cenários diferentes para a execução da aplicação:

(i) C1: Execução da aplicação com a quantidade mínima de recursos do ambiente sem habilitar a elasticidade. Dessa forma, a aplicação deve ser executada com apenas 1 máquina física durante todo o tempo de execução;

(ii) C2: Execução da aplicação iniciando com apenas 1 máquina física com a elasticidade habilitada utilizando a técnica de *thresholds* fixos. Dessa maneira, diversas configurações de *thresholds* superior e inferior devem ser utilizadas provocando diferentes execuções nesse cenário;

(iii) C3: Execução da aplicação iniciando com apenas 1 máquina física com a elasticidade habilitada utilizando a técnica de Live Thresholding. Tendo em vista as diferentes abordagens desenvolvidas, diferentes execuções devem ser realizadas com cada abordagem, definindo-se a abordagem final após a avaliação.

Cada comportamento de carga deve ser executado nos três cenários podendo-se obter as métricas da execução da aplicação com o objetivo de comparar uma execução elástica contra uma não elástica. Considerando o tempo de intervalo entre as observações de monitoramento do Gerenciador AutoElastic, foi adotado o intervalo de 15 segundos. Esse valor leva em consideração o tempo total que o *middleware* OpenNebula leva para atualizar as métricas de todos os nós que pode chegar a 1 segundo por nó, considerando a quantidade máxima de 10 nós.

Assumindo as escolhas de diferentes trabalhos, nos quais podem ser encontrados *thresholds* superiores como 50% (AL-HAIDARI; SQALLI; SALAH, 2013), 70% (DAWOUD; TAKOUNA; MEINEL, 2011; AL-HAIDARI; SQALLI; SALAH, 2013), 75% (IMAI; CHESTNA; VARELA, 2012), 80% (SULEIMAN, 2012; AL-HAIDARI; SQALLI; SALAH, 2013) e 90% (BEERNAERT et al., 2012; AL-HAIDARI; SQALLI; SALAH, 2013), foram adotados 70%, 75%, 80%, 85% e 90% como *thresholds* superiores a serem testados, enquanto que para os *thresholds* inferiores foram adotados 30%, 35%, 40%, 45% e 50%. Particularmente, o intervalo para os *thresholds* inferiores foi baseado no trabalho de Al-Haidari, Sqalli e Salah (2013), que propõe uma análise teórica com teoria de filas para observar o desempenho de elasticidade em nuvem. No caso em que a elasticidade é habilitada com *thresholds* fixos, cada comportamento de carga deve ser executado com uma configuração de *thresholds*, totalizando 25 execuções, em que esse é o número total de combinações possíveis entre os *thresholds* selecionados conforme representado na Tabela 9.

Tabela 9: Apresentação das 25 combinações possíveis de *thresholds* fixos, representadas por T_s/T_i. Cada combinação será utilizada em uma execução com cada comportamento de carga no cenário de *thresholds* fixos.

Thresholds (%)		Superior (T_s)				
		70	75	80	85	90
Inferior (T_i)	30	70/30	75/30	80/30	85/30	90/30
	35	70/35	75/35	80/35	85/35	90/35
	40	70/40	75/40	80/40	85/40	90/40
	45	70/45	75/45	80/45	85/45	90/45
	50	70/50	75/50	80/50	85/50	90/50

Fonte: elaborado pelo autor.

Considerando a técnica Live Thresholding, sua principal característica é o fato de não serem necessárias parametrizações dos algoritmos. A Seção 4.4.2 apresentou diferentes abordagens para os algoritmos de recálculos de *thresholds* quando operações de elasticidade ocorrem. Na avaliação do modelo foram realizados experimentos com estas abordagens, sendo selecionados os algoritmos definitivos para a definição da técnica.

5.5 Considerações Parciais

Este capítulo apresentou a metodologia utilizada para a avaliação do modelo. Foram desenvolvidos protótipos tanto da aplicação paralela quanto do Gerenciador AutoElastic utilizando a linguagem Java. O ambiente de testes em que os protótipos foram testados consistem em um ambiente de nuvem OpenNebula. Esta ferramenta oferece uma API Java utilizada pelo Gerenciador para realizar as tarefas de monitoramento e reorganização de recursos. A aplicação paralela utilizada no ambiente realiza o processamento de uma lista de funções integrais em intervalos determinados. Buscando avaliar o modelo desenvolvido, foram modeladas diferentes comportamentos de carga de trabalho para serem enviadas para a aplicação computar através de diferentes parametrizações para cada uma das funções a serem calculadas. No que diz respeito à avaliação, foram definidos diferentes cenários em que a aplicação será submetida para processamento. A fim de realizar uma comparação direta dos benefícios que a elasticidade pode proporcionar, esses cenário compreendem:

- C1: A execução da aplicação sem o recurso de elasticidade;
- C2: A execução da aplicação com a elasticidade utilizando a técnica de *thresholds* fixos;
- C3: A execução da aplicação com a elasticidade utilizando Live Thresholding.

Além disso, foram definidas diferentes configurações de *thresholds* levando em conta as diferentes opções adotadas por diferentes trabalhos. Com a variação de parâmetros é possível que sejam estudadas as melhores configurações para cada um dos cenários e comportamentos de cargas definidos. Os resultados obtidos das avaliações são apresentados no próximo capítulo.

6 AVALIAÇÃO

Neste capítulo é apresentada a avaliação do modelo AutoElastic considerando a metodologia definida no capítulo anterior. Durante o período de pesquisa foram realizados diversos experimentos com diferentes parâmetros e cenários para a produção científica de artigos. Aqui, são apresentados os últimos experimentos os quais consideram todas as configurações já utilizadas, como definido na metodologia de avaliação.

Este capítulo está dividido em sete seções. A Seção 6.1 apresenta a avaliação realizada com os diferentes algoritmos da técnica Live Thresholding, em que são definidos os algoritmos definitivos. Em seguida, na Seção 6.2 é realizada a avaliação das métricas definidas no modelo. Considerando os resultados da Seção 6.2 que se destacaram, na Seção 6.3 é demonstrado e avaliado em detalhes os dados de monitoramento ao longo da execução da aplicação. Ainda considerando estes resultados, na Seção 6.4 é realizada uma análise do consumo de recursos. Na Seção 6.5 é realizada uma análise do impacto da elasticidade assíncrona na execução da aplicação, enquanto que a técnica de *Aging* é avaliada na Seção 6.6. Por fim, a Seção 6.7 apresenta algumas considerações parciais do capítulo.

6.1 Análise da Abordagem Final para Live Thresholding

A técnica Live Thresholding é guiada pelo desenvolvimento de dois procedimentos: adapta_thresholds() e recalcula_thresholds(). Este último apresenta três abordagens diferentes para definir um novo valor para T_i(A_a, A_b e A_c) e outras três para T_s(A_d, A_e e A_f). A Tabela 10 apresenta os resultados considerando as nove combinações possíveis das abordagens citadas com a execução da aplicação de integração numérica e os quatro comportamentos de carga desenvolvidos. Com o objetivo de gerar apenas uma abordagem definitiva para guiar o funcionamento da técnica Live Thresholding, foi realizada uma avaliação dos resultados utilizando a técnica chamada Modelo de Soma Ponderada (ou *Weighted Sum Model*) (TRIANTAPHYLLOU, 2000). Primeiramente, foram reordenados cada comportamento de carga de forma crescente de acordo com o Custo. Após isso, em cada comportamento de carga, foi atribuído um peso para cada posição na lista: considerando as 9 combinações, a primeira posição na lista recebe o peso 1, a segunda recebe peso 0,9 e assim por diante com decrementos de 0,1. Os resultados apresentados na Tabela 11 revelaram LT_{ce} como tendo o maior peso somando-se os pesos obtidos para cada comportamento de carga, sendo esta combinação a adotada como abordagem final para a técnica Live Thresholding. LT_{ce} obteve uma soma de 3,1 composta por: 1,0 (Crescente), 1,0 (Constante), 0,9 (Decrescente) e 0,2 (Onda). Essa solução é promissora na maioria dos tipos de cargas pois é composta pela combinação de quatro comportamentos de cargas diferentes. Além da perspectiva de Custo, LT_{ce} também obteve o melhor índice do ponto de vista do tempo de execução da aplicação. Foi observado também que a abordagem A_d

obteve os piores resultados em termos de desempenho (tempo) pois o *threshold* superior é reiniciado para 1 a cada ação de elasticidade e, dessa maneira, adiando o momento em que a curva de carga do sistema vai atingir novamente o *threshold*.

Tabela 10: Resultados incluindo todas as abordagens para adaptar T_i(A_a, A_b e A_c) e T_s(A_d, A_e e A_f). Os valores em verde e vermelho representam respectivamente o melhor e o pior resultado para cada métrica.

Abordagens		Constante			Crescente			Decrescente			Onda		
T_i	T_s	Tempo	Energia	Custo	Tempo	Energia	Custo	Tempo	Energia	Custo	Tempo	Energia	Custo
	A_d	2380	10836	25790	2084	10924	22766	2397	11100	26607	2334	11700	27308
A_a	A_e	2267	11366	25767	1987	10824	21507	2243	12288	27562	2376	11602	27566
	A_f	2266	11176	25325	1893	11804	22345	2251	12252	27579	2274	11724	26660
	A_d	2413	10866	26220	2130	10836	23081	2305	12160	28029	2674	10514	28114
A_b	A_e	2382	11058	26340	1931	11548	22299	2138	13662	29209	2221	12634	28060
	A_f	2338	11032	25793	1930	11384	21971	2271	12310	27956	2609	11036	28793
	A_d	2502	10572	26451	2138	10590	22641	2274	10920	24832	2660	10472	27856
A_c	A_e	1932	12828	24784	1769	12064	21341	2000	13088	26176	2165	13408	29028
	A_f	2292	11404	26138	1879	11686	21958	2245	12036	27021	2603	10426	27139

Fonte: elaborado pelo autor.

Tabela 11: Utilizando a métrica de Custo para definir a solução final para a técnica Live Thresholding: LT_{ce} foi selecionado como a melhor abordagem considerando os diferentes tipos de comportamento de carga.

Abordagens	Peso				
	Constante	Crescente	Decrescente	Onda	Total
LT_{ad}	0,7	0,3	0,8	0,8	2,6
LT_{ae}	0,8	0,9	0,6	0,7	3,0
LT_{af}	0,9	0,5	0,5	1,0	2,9
LT_{bd}	0,4	0,2	0,3	0,4	1,3
LT_{be}	0,3	0,6	0,2	0,5	1,6
LT_{bf}	0,6	0,7	0,4	0,3	2,0
LT_{cd}	0,2	0,4	1,0	0,6	2,2
LT_{ce}	1,0	1,0	0,9	0,2	3,1
LT_{cf}	0,5	0,8	0,7	0,9	2,9

Fonte: elaborado pelo autor.

6.2 Métricas de Tempo, Energia, Custo, *Speedup* Elástico e Eficiência Elástica

Devido à quantidade de resultados e a proximidade entre os valores analisados, optou-se pela apresentação destes resultados através de tabelas ao invés de gráficos. A Tabela 12 apresenta uma análise das métricas Tempo, Energia e Custo considerando os quatro comportamentos de carga em 3 diferentes cenários: (C1) execução da aplicação com a quantidade mínima de recursos com a elasticidade desabilitada; (C2) execução da aplicação utilizando elasticidade com *thresholds* fixos; e (C3) execução da aplicação utilizando elasticidade com a nova técnica Live Thresholding. No segundo cenário, nota-se

que as maiores variações da métrica Tempo ocorrem quando são usados *thresholds* superiores diferentes. Isso demonstra que a alocação de recursos causa um maior impacto no tempo final da aplicação pois mais processos estão disponíveis para receber uma parte da carga de trabalho. Assim, os melhores e piores resultados dessa métrica foram obtidos respectivamente com o *threshold* superior 70% e 90%. Logo, quanto menor é o *threshold* superior, mais rapidamente recursos são alocados e mais processos são disponibilizados para contribuir na computação da carga de trabalho, enquanto que quanto maior o *threshold* superior mais demoradas são as alocações de recursos e mais tempo a aplicação fica sobrecarregada. Em contrapartida, a métrica de Energia possui um comportamento inverso pois os melhores e piores resultados para essa métrica foram obtidos respectivamente com o *threshold* superior 90% e 70% para todos os comportamentos de carga. Isso ocorre pois o fato de alocar mais recursos para diminuir a sobrecarga da aplicação está diretamente ligado à utilização de mais recursos. Apesar de o tempo da aplicação diminuir com a alocação de mais recursos, o tempo ganho não é o suficiente para manter um mesmo consumo de Energia.

É possível observar no cenário C2 que o maior *threshold* superior é responsável por obter os melhores valores para a métrica de Energia. Enquanto que para o comportamento de carga Crescente isso acontece com T_s igual a 90%, o valor de 50% para T_i dita os melhores índices de energia para o comportamento de carga Decrescente. Isso ocorre pois um valor perto de 100% para o comportamento de carga Crescente impede novas alocações de recursos, enquanto desalocações de recursos (melhor métrica de energia) são facilitadas com o maior valor para T_i. À medida que a inclusão de novos recursos é crítica para atingir melhor desempenho para o comportamento de carga Crescente, a desalocação de recursos não tem um impacto crucial no tempo de execução do comportamento de carga Decrescente. Isso ocorre devido ao fato de que a carga de processamento diminui com o tempo e menos recursos implicam em uma melhor redistribuição de tarefas entre os recursos disponíveis, ou seja, melhor uso dos recursos. Enquanto Tempo e Energia são divergentes algumas vezes, a métrica Custo aparece para analisar a viabilidade de uma ação de elasticidade.

Diferentemente das métricas Tempo e Energia que possuem comportamento inverso, a métrica Custo, por sua vez, não segue um padrão de qual a melhor configuração de *thresholds*. A configuração de *thresholds* que obteve o melhor Custo para cada carga possui um desempenho mais próximo da configuração que obteve o melhor índice de tempo. Para o comportamento de carga Constante, a mesma configuração de *thresholds* obteve o melhor tempo e o melhor Custo. Já para o comportamento de carga Crescente, o mesmo *threshold* superior 70% obteve o melhor Custo e melhor índice de Tempo, em que o *threshold* inferior 40% (melhor Custo) atingiu um Tempo 1,2% maior que o *threshold* inferior 30% (melhor Tempo). Por outro lado, para os comportamentos de carga Decrescente e Onda, o melhor Custo foi obtido por execuções com *thresholds* superiores diferentes dos *thresholds*

Tabela 12: Analisando as métricas de Tempo, Energia e Custo para os quatro comportamentos de carga considerando os seguintes cenários: (C1) sem elasticidade; (C2) utilizando *thresholds* fixos; e (C3) utilizando a técnica Live Thresholding. Os valores em verde e vermelho representam, respectivamente, o melhor e o pior resultado com *thresholds* fixos.

Cenários	Parâmetros		Constante			Crescente			Decrescente			Onda		
	T_s	T_i	Tempo	Energia	Custo	Tempo	Energia	Custo	Tempo	Energia	Custo	Tempo	Energia	Custo
C1		-	4283	8542	36585	4319	8618	37221	4410	8798	38799	4363	8700	37958
C2	70	30	1888	12382	23377	1818	11936	21700	1891	14056	26580	2286	12496	28566
		35	1903	12444	23681	1873	11824	22146	1906	13882	26459	2304	12256	28238
		40	1863	12294	22904	1840	11718	21561	1892	13196	24967	2288	12010	27479
		45	1898	12264	23277	1833	12058	22102	1889	13092	24731	2301	11814	27184
		50	1913	12546	24000	1825	11874	21670	1880	12746	23962	2296	11784	27056
	75	30	1890	12388	23413	2011	11350	22825	1986	13020	25858	2327	11982	27882
		35	2112	11280	23823	1959	11258	22054	1976	12540	24779	2300	12212	28088
		40	2109	11280	23790	1954	11294	22068	1967	12314	24222	2323	12036	27960
		45	2123	11370	24139	2007	11040	22157	1984	12194	24193	2340	11638	27233
		50	2119	11268	23877	2041	11302	23067	1958	11916	23332	2352	11662	27429
	80	30	2101	11238	23611	2201	10512	23137	1998	12888	25750	2325	11642	27068
		35	2110	11292	23826	2234	10488	23430	1971	12570	24775	2372	11362	26951
		40	2105	11284	23753	2201	10632	23401	1965	12294	24158	2355	11442	26946
		45	2103	11228	23612	2271	10470	23777	1971	12164	23975	2322	11222	26057
		50	2118	11348	24035	2211	10642	23529	1972	11982	23629	2383	11046	26323
	85	30	2652	10066	26695	2521	10026	25276	2170	11370	24673	2469	10808	26685
		35	2129	11450	24377	2463	10080	24827	2195	11400	25023	2540	10680	27127
		40	2603	9900	25770	2582	10080	26027	2161	11226	24259	2460	10590	26051
		45	2603	9900	25770	2610	10040	26204	2165	11160	24161	2503	10852	27163
		50	2620	9960	26095	2604	10070	26222	2191	11134	24395	2559	10568	27044
	90	30	2625	9886	25951	3091	9450	29210	2667	10110	26963	2911	9750	28382
		35	2610	9840	25682	2923	9470	27681	2667	10092	26915	2946	9842	28995
		40	2649	9900	26225	3015	9494	28624	2663	10036	26726	2922	9724	28414
		45	2626	9834	25824	2945	9480	27919	2643	9900	26166	2914	9750	28412
		50	2653	9954	26408	3000	9540	28620	2638	9840	25958	2904	9600	27878
C3		-	1932	12828	24781	1769	12064	21345	2000	13088	26176	2165	13408	29028

Fonte: elaborado pelo autor.

superiores das execuções que obtiveram melhor índice de Tempo.

Comparando os diferente cenários de execução, apesar dos comportamentos de carga do cenário C1 atingirem os melhores índices de Energia, todas as execuções com a elasticidade habilitada (cenários C2 e C3) obtiveram melhores índices de Tempo e Custo. Esse comportamento se deve ao fato de que ao executar a aplicação sem elasticidade, os recursos permanecem por muito tempo sobrecarregados, aumentando drasticamente o tempo de execução da aplicação e consequentemente o Custo, apesar de um baixo consumo de Energia. Isso demonstra a viabilidade da elasticidade para melhorar o tempo de execução da aplicação e o Custo necessário para sua execução. Ainda, comparando os resultados dos cenários C2 e C3, os comportamentos de carga Constante e Decrescente do cenário C3 obtiveram respectivamente um tempo 3,7% e 6,4% maiores em comparação aos melhores resultados dessa mesma métrica do cenário C2. Por outro lado, os comportamentos de

carga Crescente e Onda do cenário C3, em comparação com os melhores resultados da métrica tempo do cenário C2, obtiveram respectivamente um tempo 2,7% e 5,3% menor. Isso demonstra que a técnica de Live Thresholding obteve melhores resultados em comparação com a execução sem elasticidade e resultados próximos à configuração de *thresholds* fixos que obteve os melhores resultados de Tempo.

A Tabela 13 apresenta os resultados obtidos para as métricas *Speedup* Elástico (ES) e Eficiência Elástica (EE). Além dessas duas métricas, a tabela também apresenta o valor de Recursos (RE) utilizado para o cálculo da métrica EE. Este valor representa um índice da quantidade de máquinas virtuais utilizadas durante a execução da aplicação e possui um comportamento semelhante à métrica Energia. Analisando o cenário C2, e as configurações de *thresholds* que obtiveram os melhores e piores resultados para as métricas ES e EE, é possível identificar sua relação direta com as métricas Tempo e Energia apresentadas na Tabela 12. As configurações que obtiveram os melhores e piores resultados para a métrica ES são as mesmas configurações que obtiveram os melhores e piores resultados para a métrica Tempo. O mesmo ocorre com a métrica EE e a métrica Energia, demonstrando a relação direta estre essas métricas. Porém, em alguns casos para as métricas SE e EE mais de uma configuração obteve o mesmo resultado, como pode ser notado nos resultados da métrica EE dos comportamentos de carga Constante e Crescente. No geral, os resultados demonstram que um melhor desempenho equivale a uma eficiência menor e vice-versa. Isso ocorre pois, como a aplicação utilizada é intensiva quanto à CPU, na medida em que são adicionados recursos eles vão ficando cada vez menos utilizados pois o carga é espalhada entre uma maior quantidade de recursos. Entretanto, mesmo com a subutilização de recursos ganha-se desempenho pois os processos não concorrem por uma única unidade de processamento. Ainda, considerando a métrica RE, é possível notar que quanto menor o *threshold* superior maior é a quantidade de recursos, maior é o desempenho e menor é a eficiência.

6.3 Histórico de Comportamento de Carga e Alocação de Recursos

Na seção anterior, a Tabela 12 demonstrou todos os resultados obtidos por todas as execuções em diferentes cenários. Das execuções realizadas no cenário em que foi utilizada a técnica de *thresholds* fixos (C2), para cada comportamento de carga, foram selecionadas as duas execuções que obtiveram o melhor e o pior resultado considerando a métrica Tempo. Estas execuções serão chamadas respectivamente por C2.m e C2.p. Nas Figuras 17, 18, 19 e 20, são realizadas as comparações do histórico de alocação e utilização de recursos entre essas duas execuções selecionadas, a execução sem elasticidade e a execução com Live Thresholding. Todas as figuras foram geradas utilizando os eixos na mesma escala, possibilitando a comparação vertical entre cada uma das execuções. É possível notar que todas as execuções com elasticidade foram executadas de maneira mais

Tabela 13: Analisando as métricas de Recursos (RE), *Speedup* Elástico (ES) e Eficiência Elástica (EE) para os quatro comportamentos de carga considerando os seguintes cenários: (C1) sem elasticidade; (C2) utilizando *thresholds* fixos; e (C3) utilizando a técnica Live Thresholding. Os valores em verde e vermelho representam, respectivamente, o melhor e o pior resultado com *thresholds* fixos.

Cenários	Parâmetros		Constante			Crescente			Decrescente			Onda		
	T_s	T_i	RE	SE	EE	RE	SE	EE	RE	SE	EE	RE	SE	EE
C1		-	2,00	1,00	1,00	2,00	1,00	1,00	2,00	1,00	1,00	2,00	1,00	1,00
		30	6,56	2,27	0,69	6,57	2,36	0,72	7,43	2,26	0,61	5,47	1,87	0,68
		35	6,54	2,25	0,69	6,31	2,29	0,73	7,28	2,25	0,62	5,32	1,86	0,70
	70	40	6,60	2,30	0,70	6,37	2,33	0,73	6,97	2,26	0,65	5,25	1,87	0,71
		45	6,46	2,26	0,70	6,58	2,34	0,71	6,93	2,27	0,66	5,13	1,86	0,73
		50	6,56	2,24	0,68	6,51	2,35	0,72	6,78	2,28	0,67	5,13	1,87	0,73
		30	6,55	2,27	0,69	5,64	2,13	0,76	6,56	2,16	0,66	5,15	1,84	0,71
		35	5,34	2,03	0,76	5,75	2,19	0,76	6,35	2,17	0,68	5,31	1,86	0,70
	75	40	5,35	2,03	0,76	5,78	2,19	0,76	6,26	2,18	0,70	5,18	1,84	0,71
		45	5,36	2,02	0,75	5,50	2,13	0,77	6,15	2,16	0,70	4,97	1,83	0,74
		50	5,32	2,02	0,76	5,54	2,10	0,76	6,09	2,19	0,72	4,96	1,82	0,73
		30	5,35	2,04	0,76	4,78	1,95	0,82	6,45	2,14	0,66	5,01	1,84	0,73
		35	5,35	2,03	0,76	4,69	1,92	0,82	6,38	2,17	0,68	4,79	1,81	0,76
C2	80	40	5,36	2,03	0,76	4,83	1,95	0,81	6,26	2,18	0,70	4,86	1,82	0,75
		45	5,34	2,04	0,76	4,61	1,89	0,82	6,17	2,17	0,70	4,83	1,84	0,76
		50	5,36	2,02	0,75	4,81	1,94	0,81	6,08	2,17	0,71	4,64	1,80	0,78
		30	3,80	1,62	0,85	3,98	1,70	0,85	5,24	1,97	0,75	4,38	1,73	0,79
		35	5,38	2,01	0,75	4,09	1,74	0,85	5,19	1,95	0,75	4,20	1,69	0,80
	85	40	3,80	1,65	0,87	3,90	1,66	0,85	5,19	1,98	0,76	4,30	1,74	0,81
		45	3,80	1,65	0,87	3,85	1,64	0,85	5,15	1,98	0,77	4,34	1,71	0,79
		50	3,80	1,63	0,86	3,87	1,64	0,85	5,08	1,95	0,77	4,13	1,67	0,81
		30	3,77	1,63	0,86	3,06	1,39	0,91	3,79	1,61	0,85	3,35	1,47	0,88
		35	3,77	1,64	0,87	3,24	1,47	0,91	3,78	1,61	0,85	3,34	1,45	0,87
	90	40	3,74	1,62	0,87	3,15	1,42	0,90	3,77	1,61	0,85	3,33	1,47	0,88
		45	3,74	1,63	0,87	3,22	1,45	0,90	3,75	1,62	0,86	3,35	1,47	0,88
		50	3,75	1,61	0,86	3,18	1,43	0,90	3,73	1,62	0,87	3,31	1,47	0,89
C3		-	6,64	2,22	0,67	6,82	2,44	0,72	6,54	2,21	0,67	6,19	2,02	0,65

Fonte: elaborado pelo autor.

rápida do que a simples execução da aplicação sem elasticidade.

Um ponto importante a ser destacado é que em todas as figuras que exibem execuções em que foi utilizado a técnica de Live Thresholding, nota-se que, apesar de comportamentos de carga definidos, os dois *thresholds* sofrem variações. Isso ocorre pois as pequenas variações entre duas simples observações ocasionam variações nos *thresholds* dependendo se é maior ou menor que 0. Como essas pequenas variações são aplicadas de imediato ao *threshold* correspondente, elas acabam se acumulando, porém, devido a cada um dos comportamentos de carga possuírem uma tendência ao longo do tempo, isso resulta em maiores variações no *threshold* afetado pela tendência da aplicação. A cada nova operação de elasticidade, nota-se que o *threshold* inverso do qual foi violado retorna para seu valor inicial e as pequenas variações que se acumularam até esse ponto são descartadas.

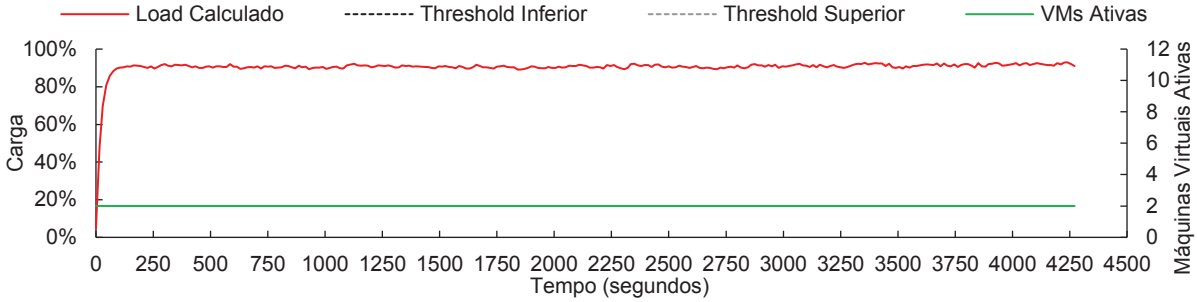
Esse comportamento pode ser melhor observado nas Figuras 18 (d) e 19 (d) em que o comportamento de carga de cada uma possui apenas um único sentido crescente ou decrescente. Apesar de pequenas variações se acumularem no valor do *threshold* contrário à tendência, para o comportamento de carga Decrescente isso é o suficiente para o valor do *threshold* superior diminuir no início da aplicação e provocar operações de elasticidade. Por outro lado, para o comportamento de carga Crescente, como a carga cresce de maneira acentuada no início e a execução é iniciada com a quantidade mínima de recursos, o novos valores que o *threshold* inferior assume não causam nenhum impacto. A seguir, cada comportamento de carga é analisado separadamente.

6.3.1 Comportamento de Carga Constante

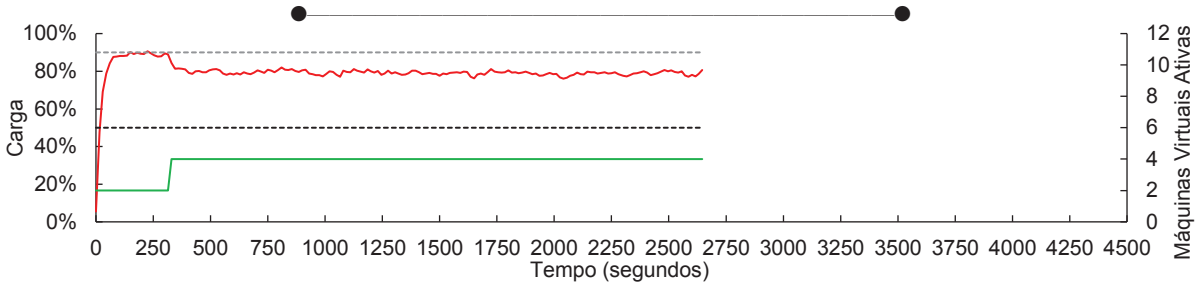
Considerando a Figura 17, um ponto importante se refere ao fato de que para todas as execuções, o *threshold* inferior não possui nenhuma influência na alocação de recursos. Este comportamento ocorre pois o comportamento de carga Constante permanece praticamente no mesmo índice de utilização de recursos e *thresholds* são violados apenas quando esse índice é um valor acima ou abaixo de um determinado valor. Durante todo o tempo de execução do comportamento de carga Constante sem elasticidade, apresentada na Figura 17 (a), a carga de processamento do sistema calculada se manteve em média em 90%, representando um maior uso dos recursos disponíveis. Já na execução desse mesmo comportamento de carga utilizando *thresholds* fixos que obteve o pior Tempo (Figura 17 (b)), apenas uma operação de alocação de recursos foi realizada logo no início da execução da aplicação no tempo 315s. Após a aplicação receber esses novos recursos, dobrando sua capacidade em relação a configuração inicial, a média da carga de processamento do sistema durante todo o restante do tempo de execução baixou para 79%. Esses novos recursos foram o suficiente para garantir um tempo de execução 38,1% melhor em comparação à execução sem elasticidade.

Já a execução do comportamento de carga Constante com *thresholds* fixos que obteve o melhor Tempo (Figura 17 (c)) e a execução utilizando Live Thresholding (Figura 17 (d)) atingiram tempos de execução muito próximos. Porém, foram alocadas diferentes quantidades de recursos em momentos diferentes de cada uma das execuções. Através da Figura 17 (c) nota-se que todas as operações de elasticidade foram realizadas durante a parte inicial da execução da aplicação. Isso ocorre pois após as primeiras duas alocações, a carga de processamento do sistema permanece acima do *threshold* superior 70% mesmo após a entrega do novo recurso. Apenas na terceira operação, esse valor fica abaixo do *threshold* e não são mais necessárias alocações de recursos. Enquanto que na execução com *thresholds* fixos foram necessárias três operações de elasticidade, entregando novos recursos nos tempos 240s, 420s e 609s, na execução com Live Thresholding foi realizada uma operação a mais. Porém, nessa execução, inicialmente foram realizadas duas operações

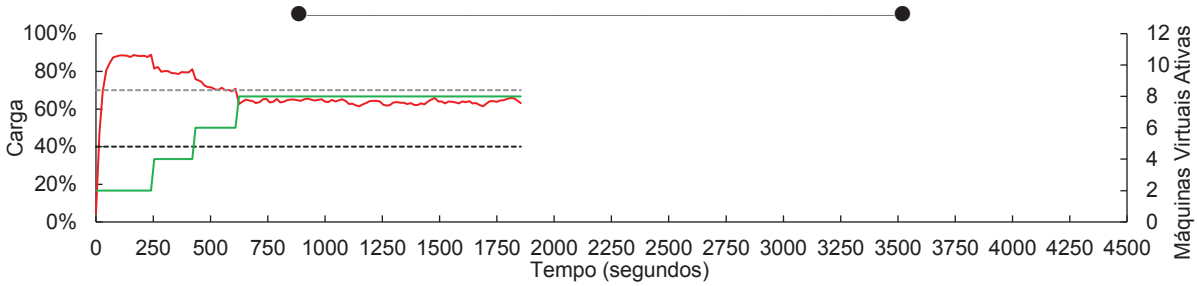
Figura 17: Carga de processamento do sistema e disponibilidade de recursos nas execuções do comportamento de carga Constante. (b) e (c) apresentam respectivamente os resultados com *thresholds* fixos que obtiveram pior e melhor Tempo de acordo com a Tabela 12.



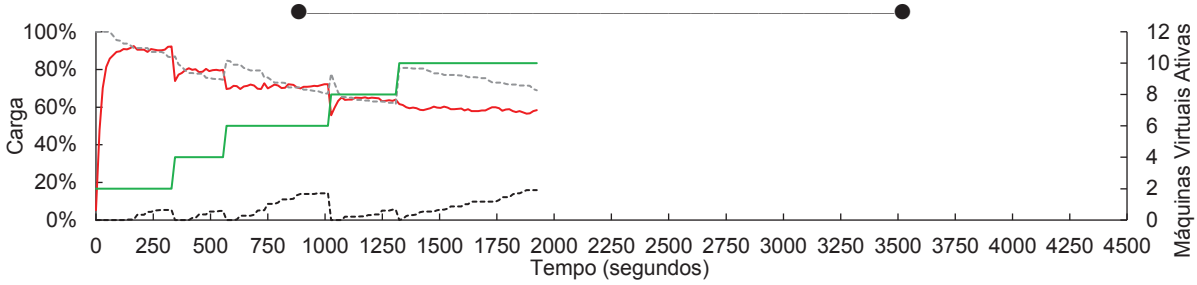
(a) C1 - Sem Elasticidade



(b) C2.p - Thresholds Fixos - Pior Resultado



(c) C2.m - Thresholds Fixos - Melhor Resultado



(d) C3 - Live Thresholding

Fonte: elaborado pelo autor.

nos tempos 330s e 555s, sendo realizadas duas operações adicionais após a metade do tempo de execução da aplicação nos tempos 1012s e 1309s. Embora a organização de recursos durante a execução desses dois cenários foi diferente ao longo do tempo, o tempo total de execução da aplicação com *thresholds* fixos foi ligeiramente menor em relação ao tempo de execução com Live Thresholding representando uma diferença de 3,6%. Ainda, os tempos de execução com *thresholds* fixos e Live Thresholding, comparados com o

tempo da execução sem elasticidade, obtiveram respectivamente um resultado 56,5% e 54,9% menor.

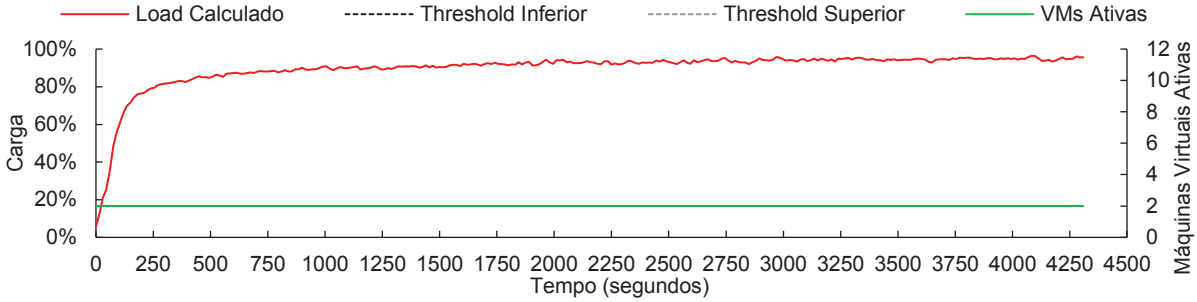
6.3.2 Comportamento de Carga Crescente

Do mesmo modo que ocorre no o comportamento de carga Constante, no o comportamento de carga Crescente, apresentado na Figura 18, o *threshold* inferior não possui nenhum impacto na execução da aplicação pois em nenhum momento esse valor é violado devido à carga do sistema possuir uma tendência crescente se aproximando mais do *threshold* superior. Apesar de no início da execução a carga do sistema ficar abaixo do *threshold* inferior, nenhuma operação de elasticidade ocorre uma vez que as execuções sempre iniciam com a quantidade mínima de recursos, não sendo possível que recursos sejam removidos. Na execução sem elasticidade retratada na Figura 18 (a), o comportamento demonstra uma crescente muito rápida nas primeira iterações da aplicação, em que a carga do sistema inicia em 6% e atinge os índices de 55% com 90 segundos de execução e 80% com 255 segundos. À partir desse ponto, o crescimento da carga ocorre com uma taxa menor à inicial ultrapassando o índice de 90% apenas após 1000 segundos de execução e permanecendo entre 94% e 96% à partir dos 2670 segundos de execução.

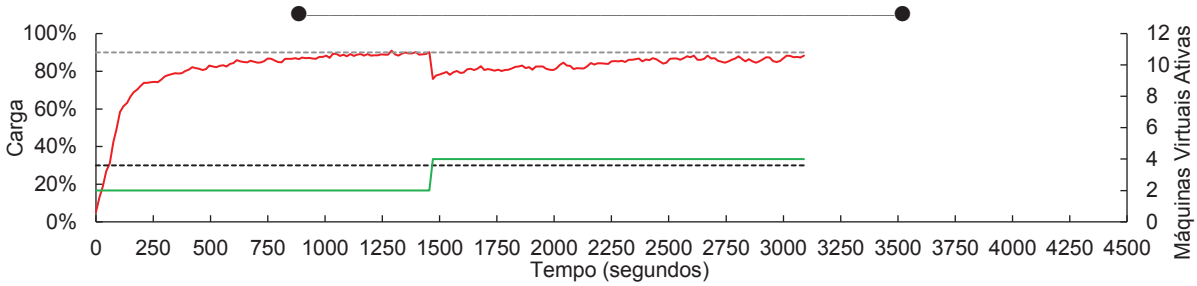
Na execução do comportamento de carga Crescente que obteve o pior Tempo (Figura 18 (b)), apenas uma alocação de recursos foi realizada, na qual novos recursos foram disponibilizados para a aplicação com 1455 segundos de execução. Neste ponto, a carga do sistema que estava em 90% sofreu uma queda para 76% pois mais processos foram disponibilizados para a aplicação dividir a carga de trabalho. À partir desse ponto a carga continuou em crescimento chegando ao máximo em 88% no final da execução da aplicação. Como o *threshold* superior nessa execução estava configurado em 90%, a carga demora mais para atingir esse valor, retardando operações de alocação de recursos. Devido a isso, a aplicação executa com poucos recursos durante períodos maiores quando feita a comparação com execuções com *thresholds* superiores menores em que mais recursos são alocados. Ainda, com o dobro de recursos disponível no ponto do tempo da execução que representa 47,1% do tempo total de execução da aplicação, essa configuração de *thresholds* obteve um tempo de execução 28,4% menor em comparação com a execução desse comportamento de carga sem a elasticidade.

Considerando a execução do comportamento de carga Crescente com *thresholds* fixos que obteve melhor Tempo (Figura 18 (c)) e a execução com Live Thresholding (Figura 18 (d)), nota-se que a mesma quantidade de recursos foi utilizada pelas duas execuções, porém os pontos que estes novos recursos foram disponibilizados foram diferentes. As duas primeiras entregas de recursos destas execuções ocorreram em tempos muito parecidos, sendo disponibilizados novos recursos para a execução com *thresholds* fixos nos tempos 300s e 480s, enquanto que para a execução com Live Thresholding foram disponibilizados

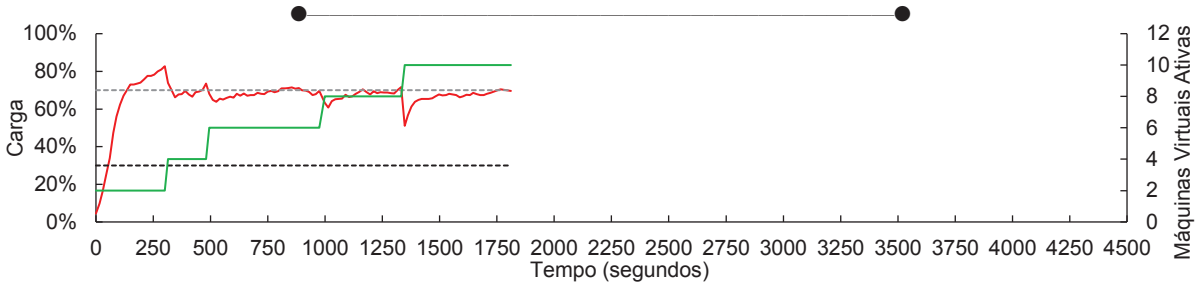
Figura 18: Carga de processamento do sistema e disponibilidade de recursos nas execuções do comportamento de carga Crescente. (b) e (c) apresentam respectivamente os resultados com *thresholds* fixos que obtiveram pior e melhor Tempo de acordo com a Tabela 12.



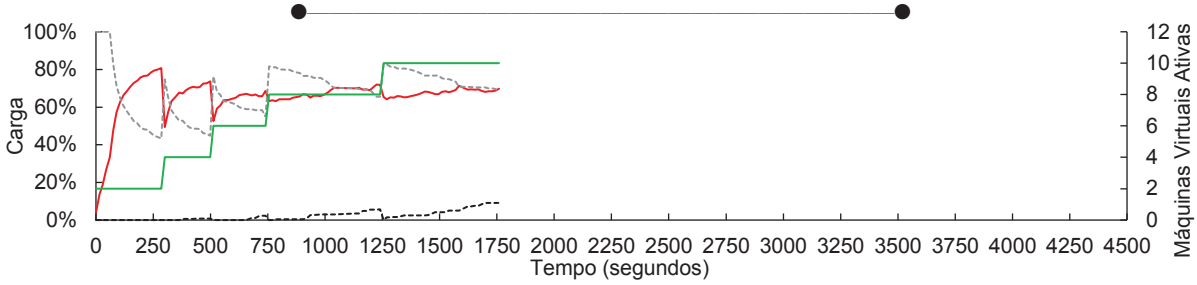
(a) C1 - Sem Elasticidade



(b) C2.p - Thresholds Fixos - Pior Resultado



(c) C2.m - Thresholds Fixos - Melhor Resultado



(d) C3 - Live Thresholding

Fonte: elaborado pelo autor.

novos recursos nos tempos 285s e 498s. Contudo, na execução com Live Thresholding foram disponibilizados novos recursos com 740 segundos de execução de maneira mais rápida que a execução com *thresholds* fixos, em que apenas com 975 segundos novos recursos foram disponibilizados. Além disso, em cada execução novos recursos foram disponibilizados nos tempos 1240s para a execução com Live Thresholding e 1332s para a execução com *thresholds* fixos. O fato de recursos terem sido disponibilizados antecipadamente

para a execução utilizando Live Thresholding contribui para o tempo de execução final desse cenário ser ligeiramente menor que o tempo de execução com *threshold* fixos. Além disso, os tempos obtidos pelas execuções com Live Thresholding e com *thresholds* fixos atingiram respectivamente um tempo de execução 59% e 57,9% menor em comparação com a execução sem elasticidade desse mesmo comportamento de carga.

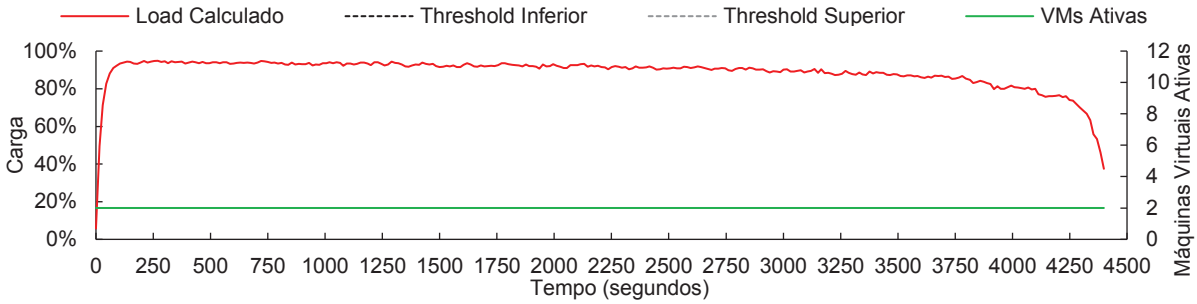
O fato de utilizar de um *threshold* superior mais baixo antecipa ações de elasticidade pois, considerando que a carga aumenta ao longo do tempo, este valor é atingido de maneira mais rápida. Analisando os pontos em que novos recursos foram entregues na Figura 18 (c) é possível notar que após novos recursos estarem disponíveis, a carga do sistema sofre uma queda pois mais processos estão disponíveis para dividir a carga de trabalho. Com o aumento gradual da carga do sistema, o *threshold* superior é novamente violado e novos recursos são alocados. Por outro lado, com a utilização de Live Thresholding e um comportamento de carga crescente, a variação de carga entre as observações provoca a queda do *threshold* superior até que este seja violado pela carga do sistema. Observando a Figura 18 (d) nota-se que após a disponibilização de novos recursos o *threshold* inferior é reconfigurado para o valor inicial 0 e o *threshold* superior é recalculado para um valor acima da nova carga do sistema que sofre uma queda devido ao aumento de processos. Em seguida, com a carga de processamento aumentando, esse *threshold* volta a diminuir seu valor até o ponto em que é violado novamente, antecipando a elasticidade.

6.3.3 Comportamento de Carga Decrescente

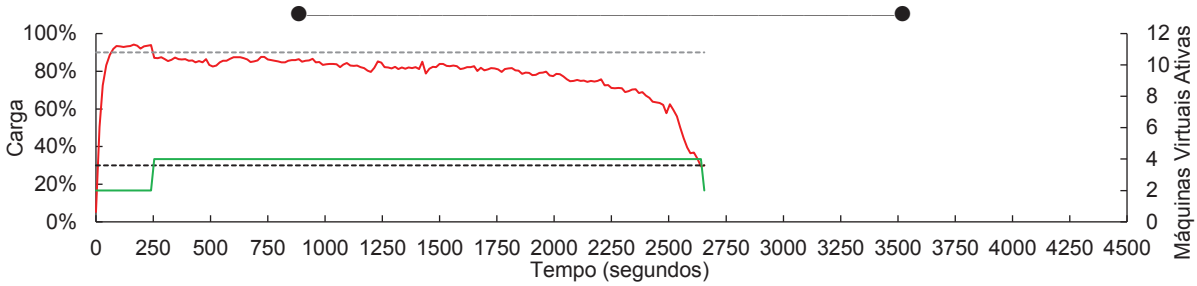
Considerando o comportamento de carga Decrescente com elasticidade, a Figura 19 demonstra que, diferentemente dos comportamentos de carga Constante e Crescente, o valor do *threshold* inferior possui impacto na execução da aplicação, ocasionando operações de elasticidade. O comportamento dessa carga pode ser visualizado na Figura 19 (a), em que a aplicação é executada sem elasticidade. De maneira contrária ao que ocorre na execução do comportamento de carga Crescente, a execução do comportamento Decrescente inicia com um índice da carga do sistema entre 94% e 96% diminuindo gradualmente ao longo do tempo. Apenas no final da execução a queda do valor da carga do sistema ocorre de maneira mais acentuada à partir do tempo 4294s em que esse valor atinge 70%, chegando a 38% na última observação no tempo 4399s. Como a carga possui uma tendência ao diminuir ao longo do tempo, isso torna possível que o *threshold* inferior seja violado desencadeando operações de remoção de recursos.

Ao executar a carga Decrescente com diferentes configurações de *thresholds*, a configuração que obteve o pior resultado apresentada na Figura 19 (b) demonstra que com o *threshold* superior em 90% esse valor é violado logo no início, pois a carga do sistema atinge 94%, e novos recursos são entregues com 240 segundos de execução. Após os recursos serem disponibilizados pela primeira vez, esse índice de carga cai para 87% e, como a

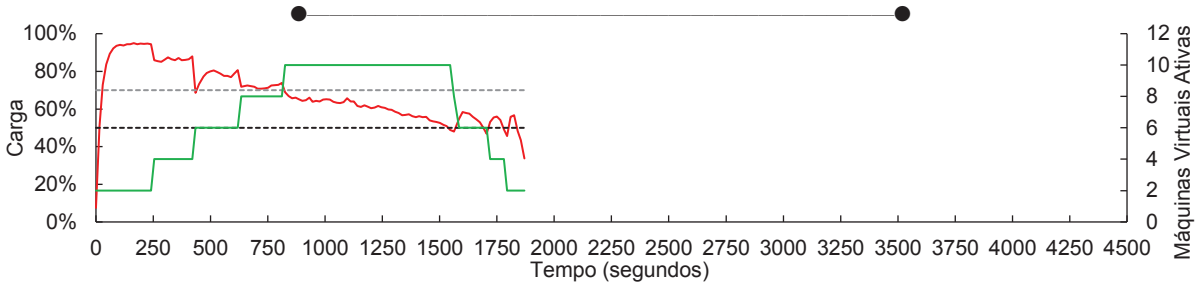
Figura 19: Carga de processamento do sistema e disponibilidade de recursos nas execuções do comportamento de carga Decrescente. (b) e (c) apresentam respectivamente os resultados com *thresholds* fixos que obtiveram pior e melhor Tempo de acordo com a Tabela 12.



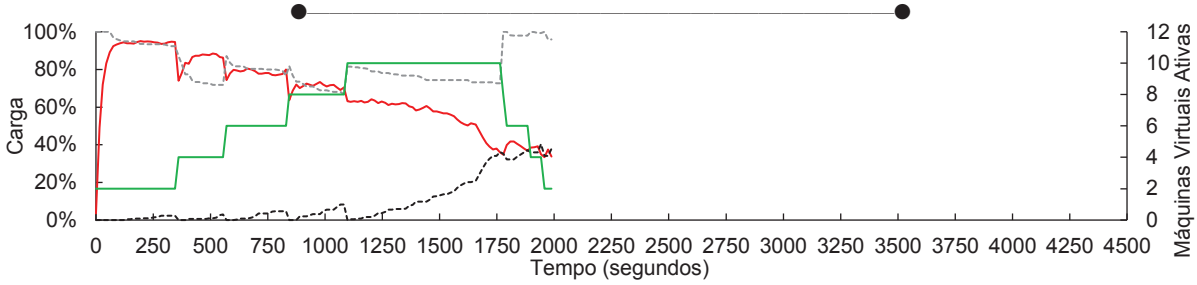
(a) C1 - Sem Elasticidade



(b) C2.p - Thresholds Fixos - Pior Resultado



(c) C2.m - Thresholds Fixos - Melhor Resultado



(d) C3 - Live Thresholding

Fonte: elaborado pelo autor.

aplicação tende a diminuir a carga ao longo do tempo, o *threshold* superior não é violado mais nenhuma vez até o final da execução da aplicação. Ainda, antes de a aplicação ser finalizada, a carga do sistema atinge o *threshold* inferior com 2640 segundos de execução ocasionando a remoção de recursos. Esses recursos adicionais acarretam um tempo 39,5% melhor em relação ao tempo de execução do cenário sem elasticidade desse mesmo comportamento de carga. Porém, tempos de execução melhores são obtidos por execuções em

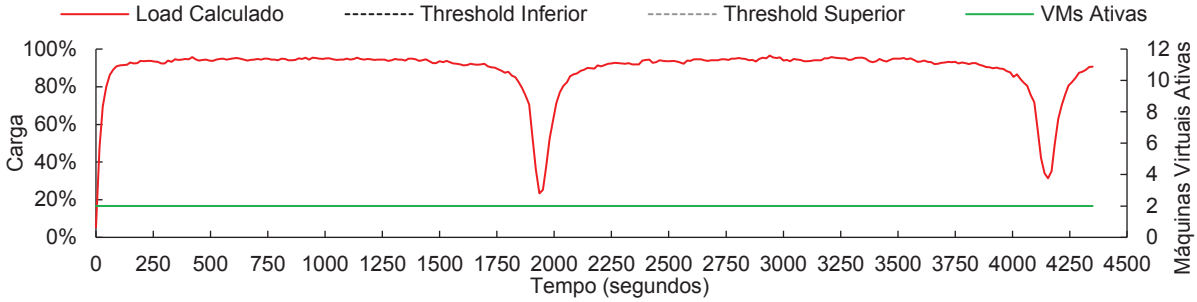
que houveram mais alocações de recursos, como demonstrado nas Figuras 19 (c) e (d).

Comparando a execução com *thresholds* fixos apresentada na Figura 19 (c) e com Live Thresholding na Figura 19 (d), observa-se que a mesma quantidade de recursos (10 máquinas virtuais) foi utilizada ao longo da execução da aplicação. Contudo, analisando especificamente os pontos de tempo expressados no eixo x, nos pontos 250s, 500s, 750s e 1000s nota-se que a execução com Live Thresholding sempre possui 2 máquinas virtuais a menos, igualando a quantidade apenas alguns segundos após. Isso ocorre pois na execução com *thresholds* fixos, logo após um novo recurso ser disponibilizado a carga permanece acima do *threshold* superior desencadeando uma nova operação. No entanto, para a execução com Live Thresholding, sempre que o *threshold* superior é recalculado devido a sua violação, seu novo valor geralmente fica acima da carga do sistema da observação seguinte, a qual sofre uma ligeira queda pois mais processos podem ser utilizados pela aplicação. Em consequência a isso, são necessárias algumas observações para que o *threshold* superior diminua novamente ao ponto de atingir a carga do sistema. Consequentemente, esse comportamento diferente entre as abordagens culmina em um tempo de execução ligeiramente menor para a execução com *thresholds* fixos, ainda que próximos. Somando-se a isso, o tempo de execução com *thresholds* fixos e Live Thresholding obtiveram respectivamente um tempo 57,4% e 54,6% menor em comparação com a execução sem elasticidade do comportamento de carga Decrescente.

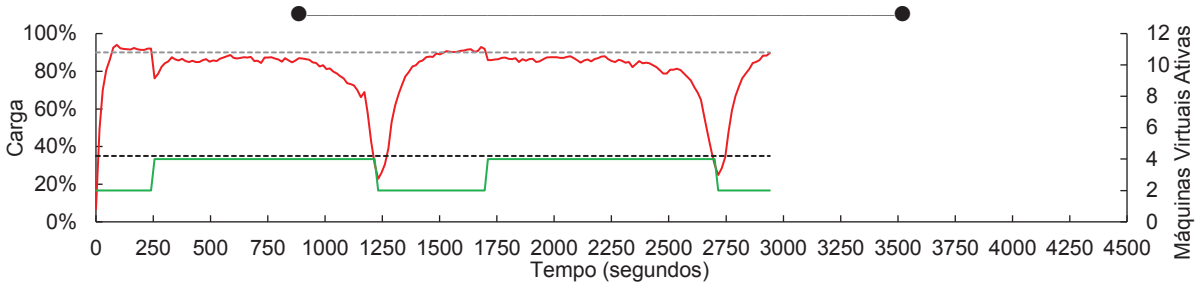
6.3.4 Comportamento de Carga Onda

Do mesmo modo que o *threshold* inferior causa impacto na execução do comportamento de carga Decrescente, o mesmo ocorre para o comportamento de carga Onda demonstrado na Figura 20. A execução sem elasticidade deste comportamento de carga (Figura 20 (a)) aponta que em determinados pontos da execução a carga do sistema atinge, por um período mais curto de tempo, índices abaixo de 25% como o ponto de tempo 1935s em que esse índice é de 23%. Durante a maior parte do tempo de execução a média do índice da carga do sistema fica em 94%, sofrendo quedas apenas nos pontos em que a quantidade de tarefas da aplicação atinge a quantidade mínima diminuindo a carga do sistema. Para este comportamento de carga, quando executado com *thresholds* fixos, as execuções utilizando o *threshold* superior 90% obtiveram os piores resultados de tempo de execução. A Figura 20 (b) apresenta a execução que obteve o pior resultado, em que apenas uma operação de elasticidade é realizada nos dois pontos da curva em que o *threshold* é violado. Não importando qual o *threshold* inferior, estes novos recursos sempre são removidos quando a carga atinge os picos mínimos. Em virtude da utilização de mais recursos em determinados períodos este cenário obteve um tempo 32,5% menor em comparação com a execução sem elasticidade. Porém, com a utilização de um *threshold* superior menor e com a utilização do Live Thresholding, esse índice pode atingir valores

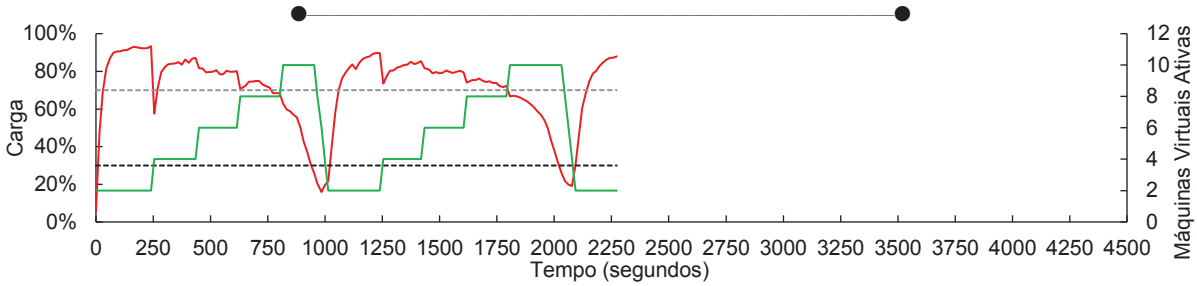
Figura 20: Carga de processamento do sistema e disponibilidade de recursos nas execuções do comportamento de carga Onda. (b) e (c) apresentam respectivamente os resultados com *thresholds* fixos que obtiveram pior e melhor Tempo de acordo com a Tabela 12.



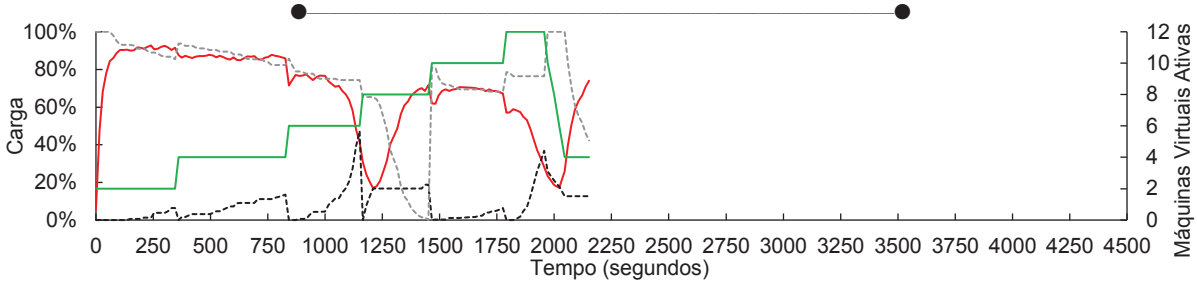
(a) C1 - Sem Elasticidade



(b) C2.p - Thresholds Fixos - Pior Resultado



(c) C2.m - Thresholds Fixos - Melhor Resultado



(d) C3 - Live Thresholding

Fonte: elaborado pelo autor.

ainda maiores.

Analisando as execuções apresentadas nas Figuras 20 (c) e (d), nota-se uma grande diferença na disponibilidade de recursos ao longo da execução da aplicação. Na execução com *thresholds* fixos (Figura 20 (c)) podem ser notados dois ciclos semelhantes iniciados nos tempos 0s e 1100s. À partir desses dois pontos a carga do sistema segue um traço semelhante assim como a alocação de recursos. Isso ocorre devido ao fato de que no

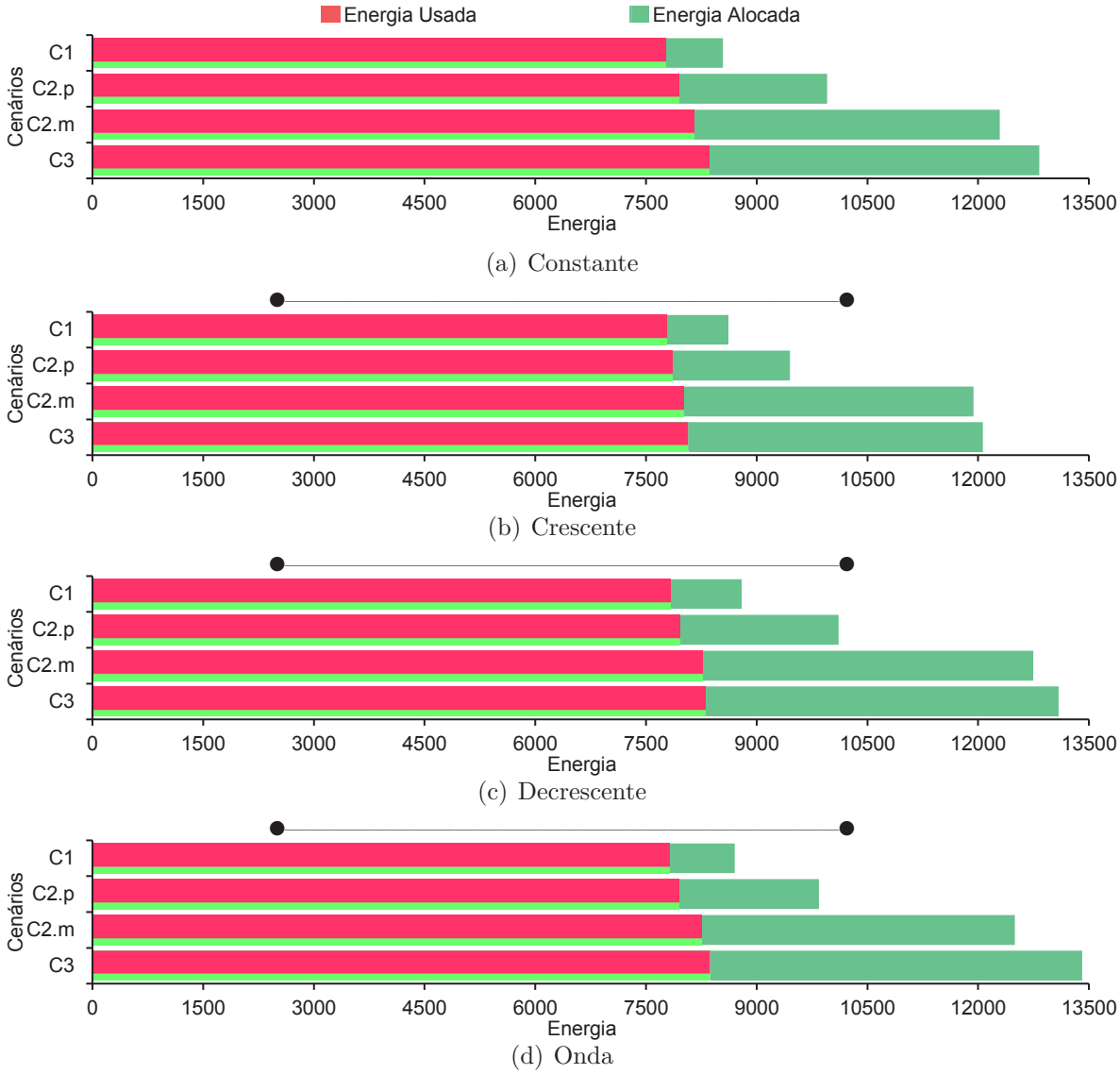
início da execução da aplicação são realizadas 4 operações de elasticidade para alocação de recursos, porém esses mesmos recursos são removidos no momento em que a aplicação atinge seu menor pico inferior de carga. Após isso, quando a carga do sistema começa a crescer novamente, o mesmo comportamento ocorre e ao final novamente estes recursos são removidos. Observando a Figura 20 (c), são alocadas 8 máquinas virtuais adicionais antes que o primeiro pico mínimo da carga ocorra, o qual foi aos 984 segundos de execução. Já, quando o segundo pico mínimo da carga ocorre aos 2078 segundos de execução, a mesma quantidade de recursos está disponível.

Em contrapartida, a execução com Live Thresholding possui um comportamento completamente diferente. Examinando a Figura 20 (d), os recursos são alocados de maneira mais gradual, sendo que o primeiro pico mínimo de carga ocorre apenas com 1210 segundos de execução e apenas 6 máquinas virtuais adicionais foram disponibilizadas até o momento, sendo que os últimos dois recursos foram liberados durante a queda da carga do sistema. Devido ao recálculo realizado quando novos recursos são disponibilizados, o *threshold* inferior foi reconfigurado para 0 quando a carga do sistema estava próxima de chegar ao pico mínimo. Em consequência a isso, como a carga começou a crescer novamente, isso evitou que o *threshold* inferior fosse violado e a quantidade de recursos foi mantida. À partir desse ponto, com a carga em crescimento novos recursos foram adicionados gradualmente aos já existentes totalizando 12 máquinas virtuais, possibilitando que o tempo restante de execução da aplicação fosse acelerado. Isso pode ser notado pelo ponto da execução em que o segundo pico mínimo ocorre, o qual foi aos 2022 segundos de execução. Agora, durante a queda da da carga do sistema, foram realizadas 4 operações de elasticidade consecutivas em que foram removidas 8 máquinas virtuais. Esse comportamento em que foi possível manter recursos a mais em comparação com a execução com *thresholds* fixos possibilitou um tempo menor para a execução com Live Thresholding, a qual obteve um tempo 50,4% menor em comparação com a execução sem elasticidade, enquanto a execução com *thresholds* fixos obteve um tempo 47,6% menor na mesma comparação.

6.4 Relação Entre Alocação e Consumo de Recursos

A Figura 21 ilustra a métrica Energia considerando os cenários e comportamentos de carga. Essa métrica representa a quantidade total de recursos que foram alocados durante a execução da aplicação. Em adição a essa métrica, cada barra também revela quanto realmente foi utilizado da capacidade disponibilizada. Como é observado a quantidade de máquinas virtuais alocadas a cada observação de monitoramento, também é obtido o valor da carga de processamento total no mesmo instante. Por exemplo, se 4 máquinas virtuais são alocadas na o^{a} observação de monitoramento, e a carga de processamento de todos os recursos equivale a 80%, o consumo de recursos indica um valor de 3,2 máquinas

Figura 21: Relação entre alocação (métrica Energia) e consumo de recursos: (C1) execução sem elasticidade; (C2.p) e (C2.m) execuções com *thresholds* fixos que obtiveram respectivamente o pior e melhor Tempo de acordo com a Tabela 12; e (C3) Live Thresholding.



(a) Constante

(b) Crescente

(c) Decrescente

(d) Onda

Fonte: elaborado pelo autor.

virtuais para esse instante. Essa mesma lógica é utilizada para realizar o cálculo parcial da energia consumida a cada observação. A figura realiza uma comparação da alocação e consumo de recursos das execuções de cada comportamento de carga nos diferentes cenários, conforme apresentado na Tabela 12: (C1) execução sem elasticidade; (C2.p) execução com *thresholds* fixos que obteve o pior tempo de execução; (C2.m) execução com *thresholds* fixos que obteve o melhor tempo de execução; (C3) execução com a utilização de Live Thresholding. As figuras correspondentes a cada comportamento de carga foram organizadas de maneira a utilizar o eixo x na mesma escala possibilitando uma comparação vertical entre eles. Em um primeiro momento, nota-se que um padrão ocorre para todos as cargas, tanto para a alocação quanto para o consumo de recursos. Considerando o

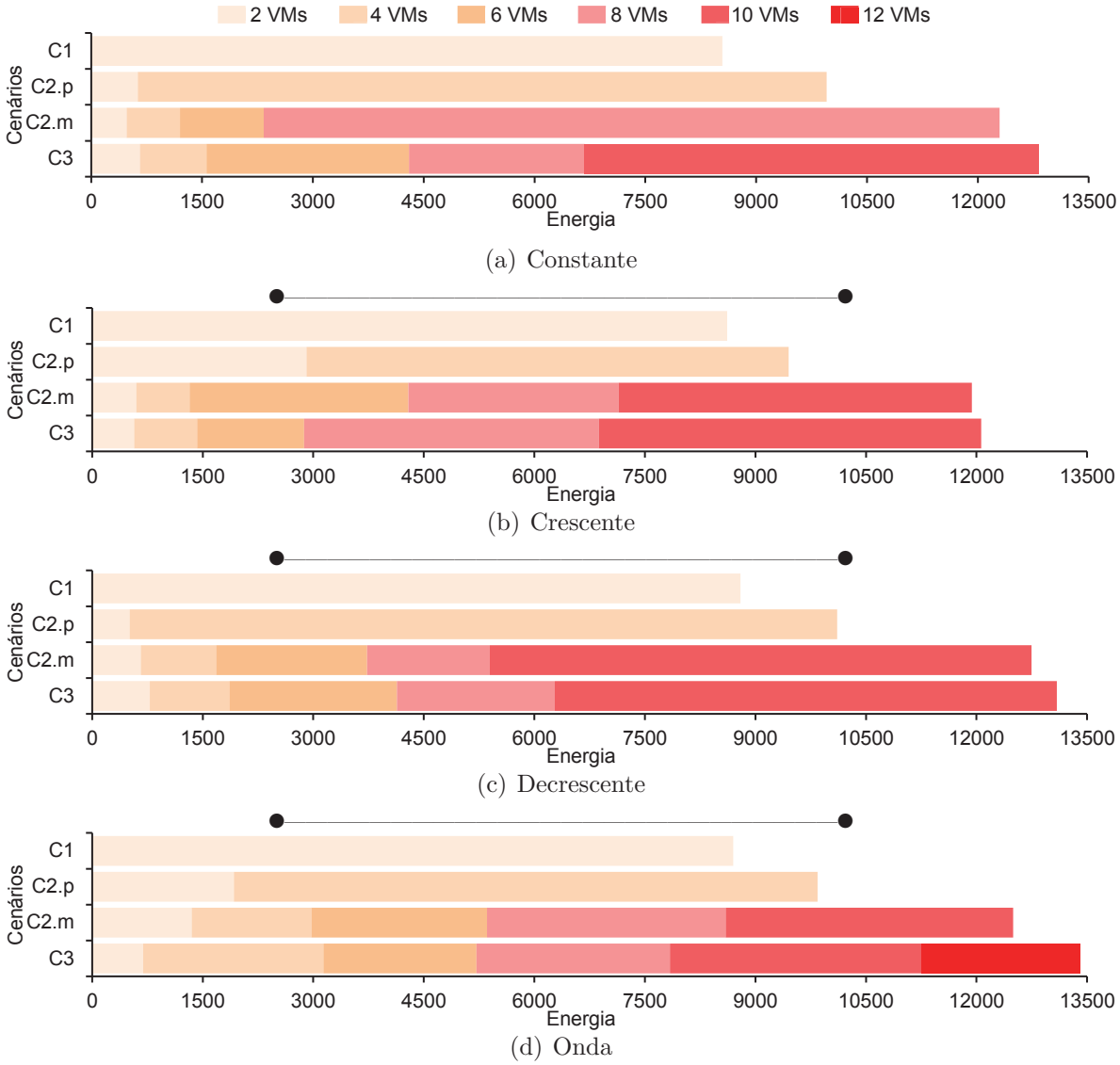
consumo de recursos, os índices das execuções apresentadas de todos os comportamentos de carga atingiram um valor muito próximo ficando entre 7765 e 8366, representando uma variação máxima de 7,7% com base no menor valor. A execução sem elasticidade possui os menores valores para alocação e consumo pois nesse cenário a aplicação é executada durante todo o tempo com a menor quantidade de recursos possível.

Aqui também é pertinente observar a indireta proporcionalidade entre Tempo e Energia: para executar de maneira mais rápida são necessários mais recursos ou ainda, ao priorizar poupar alocação de recursos possivelmente causará penalidades de tempo em aplicações paralelas. Por fim, foi percebido que o consumo de recursos é quase vertical ao longo das execuções. Em outras palavras, se com 6 máquinas virtuais a carga de processamento usada é 90%, o mesmo consumo de recursos ocorre com 8 máquinas virtuais com carga de processamento igual a 67,5%. Apesar de serem alocados mais recursos paras a execução com *thresholds* fixos que obteve melhor Tempo e a execução com Live Thresholding, o consumo de recursos ficou muito próximo às demais execuções. Em virtude disso, a eficiência nessas execuções é menor quando comparada com a eficiência das execuções sem elasticidade e com *thresholds* fixos que obteve o pior resultado. Por exemplo, considerando a Figura 21 (a), a eficiência das execuções (C1) e (C2.p) foram respectivamente 90,3% e 83,1%, enquanto que a eficiência das execuções (C2.m) e (C3) foram respectivamente 67,1% e 66,9%. Em questões de desempenho, isso demonstra que ao invés de executar uma aplicação com 1 recurso utilizando 90% da capacidade do recurso, é melhor executar a aplicação e x recursos utilizando 70% da capacidade de cada um. A análise demonstra que na perspectiva de desempenho, é melhor executar na segunda opção sempre quando se tem um grão de computação (proporção de computação e comunicação em cada processo da aplicação paralela) viável. De acordo com Galante e Bona (2015), o consumo de recursos em aplicações científicas é diferente do apresentado em aplicações baseadas em servidor. O primeiro tende a consumir todos os recursos disponíveis, independentemente da quantidade fornecida e o último consome os recursos de acordo com variações da carga.

Na Figura 22 são destacadas as diferentes quantidades de recursos alocados e sua contribuição para a métrica Energia nos cenários: (C1) execução sem elasticidade; (C2.p) execução com *thresholds* fixos que obteve o pior tempo de execução; (C2.m) execução com *thresholds* fixos que obteve o melhor tempo de execução; (C3) execução com a utilização de Live Thresholding. Considerando as diferentes cargas é possível identificar que os menores índices de energia foram obtidos pelas execuções em que menores quantidades de recursos foram utilizadas. Por exemplo, durante todo o tempo de execução nos cenários sem elasticidade (C1) foram utilizadas apenas 2 máquinas virtuais. Por outro lado, os maiores resultados da métrica Energia foram obtidos pelas execuções que empregaram uma maior quantidade de recursos. Nas execuções (C2.m) e (C3) a parcela de contribuição para para o valor final da métrica é maior para as quantidades 8, 10 e 12 máquinas virtuais, representando mais de 50% de contribuição. Outro ponto importante a ser destacado é o

fato de que apenas a execução do comportamento de carga Onda com Live Thresholding atingiu a quantidade de 12 máquinas virtuais. Isso contribuiu para essa execução com Live Thresholding ter o maior valor para a métrica Energia dentre todas as execuções apresentadas.

Figura 22: Perfil consumo de recursos considerando diferentes organizações de recursos para: (C1) execução sem elasticidade; (C2.p) e (C2.m) execuções com *thresholds* fixos que obtiveram respectivamente o pior e melhor Tempo de acordo com a Tabela 12; e (C3) Live Thresholding.



(a) Constante

(b) Crescente

(c) Decrescente

(d) Onda

Fonte: elaborado pelo autor.

6.5 Impacto da Elasticidade Assíncrona

A Tabela 14 apresenta todas as operações de alocação de recursos realizadas nas execuções: (C1) sem elasticidade; (C2.p) e (C2.m) *thresholds* fixos que obtiveram respectivamente o pior e melhor Tempo de acordo com a Tabela 12; e (C3) Live Thresholding.

No momento em que é identificada a necessidade de novos recursos, um período de tempo é necessário para liberar efetivamente estes recursos para a aplicação. Esta tabela apresenta este período de tempo que ocorre entre os pontos e que recursos são inicializados (Alocação) e o momento em que eles são liberadas para a aplicação (Entrega). Além disso, a tabela também apresenta o índice da observação do Gerenciador AutoElastic em que esses momentos ocorreram.

Considerando todas as execuções realizadas durante a fase de testes, apresentadas na Tabela 12, ocorreram um total de 363 operações de alocação de recursos. Dessas operações, 339 foram operações completas em que a aplicação efetivamente recebeu os novos recursos e os utilizou. As demais 24 operações se referem a casos em que a alocação do recurso inicia quando a aplicação está próxima ao final e termina durante a operação. O tempo médio total observado de todas as operações foi de 166,79 segundos com mediana 165 e desvio padrão de 4,23 segundos. Este período compreende a transferência da imagem de duas novas máquinas virtuais para uma máquina física e a inicialização do sistema operacional das máquinas virtuais. Apenas após as novas máquinas virtuais estarem aptas a realizarem comunicação de rede é que elas são efetivamente liberadas para a aplicação. Durante esse período a aplicação continua executando com os recursos já disponíveis.

As alocações de recursos apresentadas na Tabela 14 demonstram que maiores quantidades de operações foram realizadas nas execuções que obtiveram os melhores resultados de tempo de execução da aplicação. É possível notar ainda, que o comportamento de carga Onda é o que mais necessita de operações de elasticidade, se distribuindo ao longo das observações devido ao seu comportamento oscilatório. No caso da execução com *thresholds* fixos, nos picos mínimos de carga alguns recursos foram removidos sendo necessários novamente em um momento posterior da execução. Já o comportamento de carga Constante necessita de operações apenas nas observações iniciais. Esse mesmo comportamento ocorre com o comportamento de carga Decrescente, que necessita de operações de alocação de recursos nas observações iniciais, porém, como uma carga inicial maior em relação ao comportamento de carga Constante, são necessárias a mesma quantidade ou mais operações. Por outro lado, o comportamento de carga Crescente necessita de operações em sua maioria em observações maiores, pois apenas após um período de tempo a carga de processamento começa a atingir níveis altos, violando os *thresholds* superiores.

Buscando analisar com maior profundidade o impacto de uma operação de elasticidade para aumentar a quantidade de recursos, a Figura 23 apresenta em maiores detalhes a execução da aplicação paralela com o comportamento de carga Crescente, quando utilizada a configuração de *thresholds* fixos do cenário C2 que obteve o melhor Tempo. Nesta figura foram destacados os pontos exatos durante a execução da aplicação em que o *threshold* superior foi violado (círculo amarelo), desencadeando a alocação de novas máquinas virtuais, e o ponto em que estes novos recursos foram entregues para a aplicação (círculo azul). Em particular, para esta execução, foram realizadas quatro operações completas

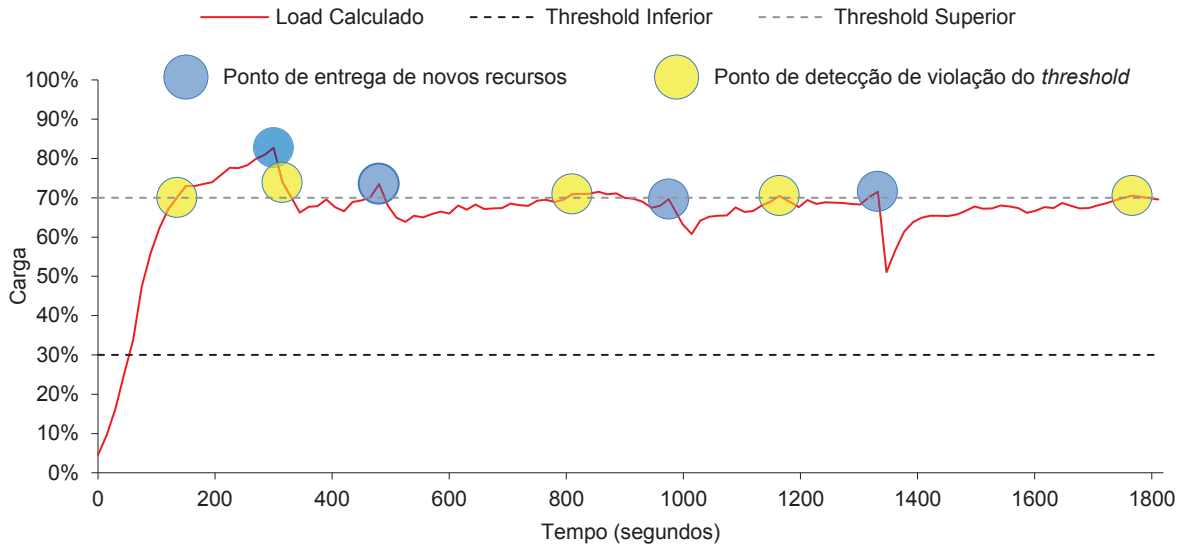
Tabela 14: Alocações de recursos realizadas nas execuções utilizando Live Thresholding e execuções que obtiveram pior e melhor tempo utilizando *thresholds* fixos.

Execução		Carga	Observação		Tempo de Execução		Tempo Total
			Alocação	Entrega	Alocação	Entrega	
Thresholds Fixos	Melhor Resultado	Constante	6	17	75	240	165
			18	29	255	420	165
			30	41	435	609	174
		Crescente	10	21	135	300	165
			22	33	315	480	165
			55	66	810	975	165
			78	89	1164	1332	168
			118	-	1767	-	-
		Decrescente	6	17	75	240	165
			18	29	255	420	165
			31	42	450	619	169
			43	54	634	811	177
		Onda	6	17	75	240	165
			19	30	270	435	165
			31	42	450	615	165
			43	54	630	802	172
			72	83	1074	1239	165
			84	95	1254	1419	165
			96	107	1434	1604	170
			108	119	1619	1792	173
			143	-	2153	-	-
	Pior Resultado	Constante	11	22	150	315	165
		Crescente	87	98	1290	1455	165
		Decrescente	6	17	75	240	165
		Onda	6	17	75	241	166
			103	114	1531	1696	165
Live Thresholding		Constante	12	23	165	330	165
			27	38	390	555	165
			57	68	840	1012	172
			76	87	1132	1309	177
		Crescente	9	20	120	285	165
			23	34	330	498	168
			38	47	558	740	182
			67	75	1040	1240	200
			98	-	1585	-	-
		Decrescente	13	24	180	345	165
			27	38	390	555	165
			45	56	660	829	169
			62	72	919	1082	163
		Onda	13	24	180	345	165
			45	56	660	827	167
			66	77	977	1150	173
			86	96	1285	1451	166
			105	115	1586	1776	190
			135	-	2105	-	-

Fonte: elaborado pelo autor.

de alocação de recursos e uma operação que foi iniciada no final da execução, porém não foi completada devido à aplicação ter finalizado sua execução durante a operação. Isso ocorre pois o Gerenciador AutoElastic não possui nenhum conhecimento prévio da aplicação, não sabendo o percentual de conclusão da computação e nem quando a aplicação será finalizada. Devido a isso, logo após identificar a necessidade de recursos, o Gerenciador inicializou novos recursos porém não conseguiu entregá-los pois a aplicação finalizou sua execução durante o período em que os recursos estavam sendo inicializados. Considerando as operações completas, é possível notar que o período entre um ponto de detecção de violação do *threshold* e o ponto de entrega de novos recursos, a carga do sistema continua estável ou em crescimento devido à aplicação continuar normalmente sua execução. Em todos os pontos em que há entrega de novos recursos, nota-se uma queda no índice da carga de sistema devido ao fato da aplicação contar com mais processos para dividir a carga de trabalho. Particularmente, após a primeira entrega de recursos ser realizada seu índice, apesar de diminuir, ainda sim continua acima do *threshold* superior desencadeando imediatamente uma nova alocação de recursos. Isso ocorre em virtude da carga do sistema estar bastante alta no ponto de entrega em comparação com os outros pontos de entrega, em que a carga está mais próxima ao *threshold* superior.

Figura 23: Avaliação de instantes de detecção de violação do *threshold* superior e entrega de novos recursos.



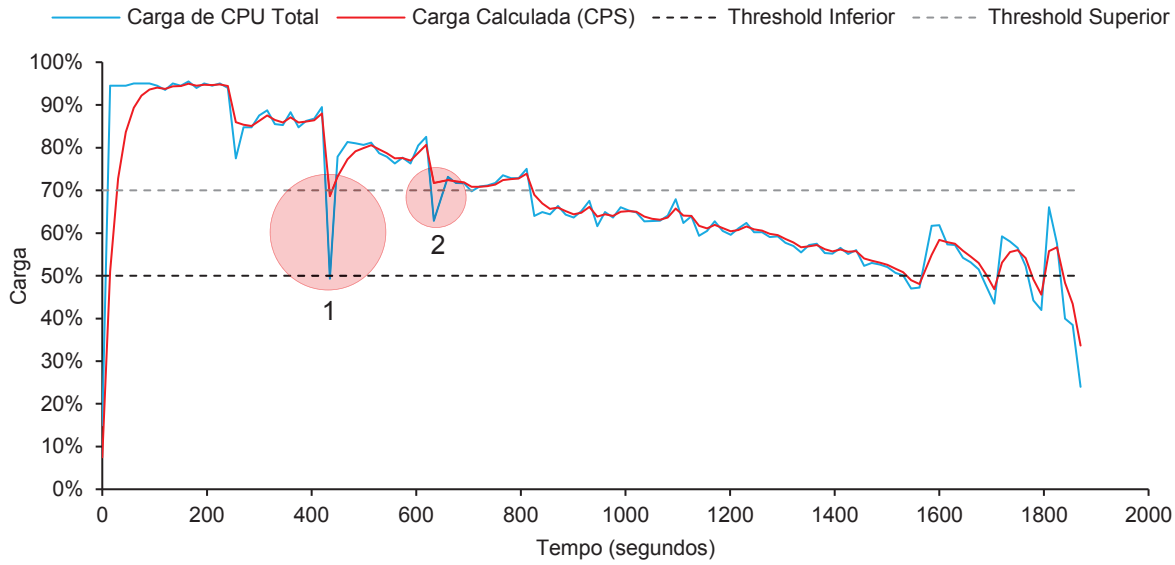
Fonte: elaborado pelo autor.

6.6 Avaliação da Técnica de *Aging* no Cálculo de Carga

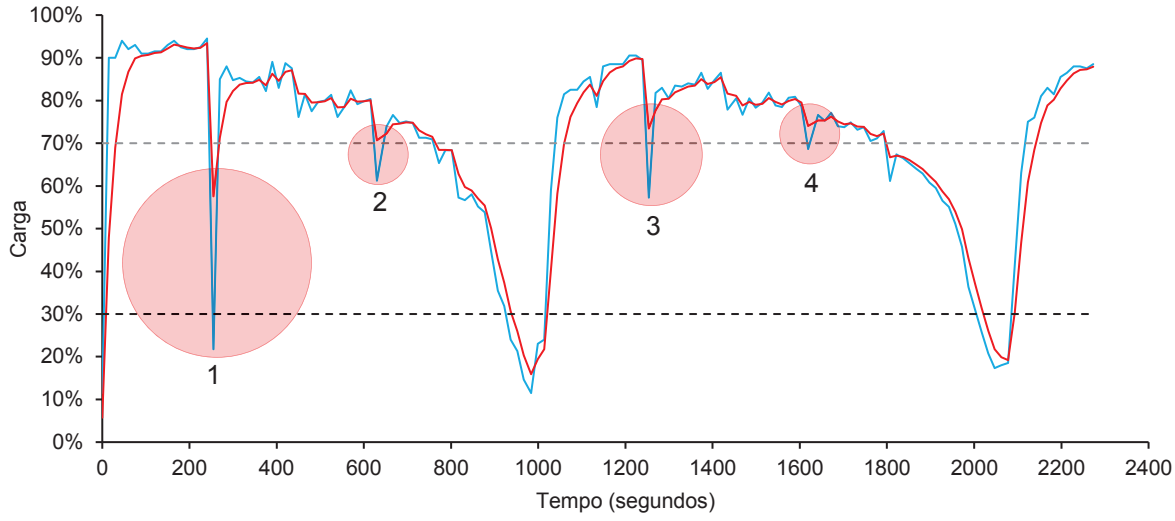
A Figura 24 apresenta uma comparação entre os dados de monitoramento e o valor da carga de processamento do sistema (CPS) que emprega a técnica de *Aging*. Considerando as execuções que obtiveram os melhores resultados de tempo quando utilizados *thresholds* fixos, a figura mostra os dados de duas execuções específicas, em que a suavização realizada

no cálculo de CPS se mostrou decisiva: (a) execução da carga Decrescente com *threshold* superior 70% e *threshold* inferior 50% e (b) execução da carga em Onda com *threshold* superior 70% e *threshold* inferior 30%. Através das figuras, nota-se que a carga total de CPU e a carga calculada CPS possuem traços próximos, porém na carga total de CPU ocorrem mais variações entre as observações. Através do emprego da técnica de *Aging*, estas variações são suavizadas para o cálculo da carga do ambiente que é efetivamente utilizada para a tomada de decisão.

Figura 24: Comparação entre a carga total de CPU do ambiente e a carga calculada utilizada para elasticidade. Os pontos destacados demonstram operações de remoção de recursos evitadas (ponto 1) e operações de adição de recursos que foram adiantadas (pontos 2, 3 e 4).



(a) Melhor Tempo Carga Decrescente - Thresholds Fixos



(b) Melhor Tempo Carga em Onda - Thresholds Fixos

Fonte: elaborado pelo autor.

As quedas da carga de CPU apresentados da figura ocorrem justamente nos pontos em que novos recursos foram disponibilizados para a aplicação. Isso ocorre pois quando

novos recursos estão disponíveis o processo mestre necessita realizar a conexão com estes recursos, pausando a distribuição de tarefas nesse período. No breve período de tempo em que isso ocorre, todos os processos escravos aguardam novas tarefas diminuindo o consumo de CPU de suas respectivas máquinas virtuais causando uma queda na carga total de CPU do ambiente. Na Figura 24 (a) foram destacados dois pontos específicos durante a execução da aplicação, em que essa queda poderia ter causado operações que foram evitadas devido ao *Aging*. O primeiro ponto ocorreu no tempo de 420 segundos, em que a carga de CPU ficou abaixo do *threshold* inferior. Porém, como a carga calculada foi suavizada, seu valor ficou acima deste *threshold* evitando que recursos recém disponibilizados fossem removidos. Por outro lado, no tempo de 619 segundos, a carga de CPU atingiu um valor abaixo do *threshold* superior. Já a carga calculada nesse ponto, ficou acima deste *threshold* adiantando uma nova operação de adição de recursos. Estes mesmos pontos foram destacados na Figura 24 (b), em que o ponto 1 no tempo de 240 segundos representa uma queda na carga total de CPU que ficou abaixo do *threshold* inferior. Já os pontos 2, 3 e 4, que ocorreram nos respectivos tempos 615, 1239 e 1604 segundos, representam suavizações realizadas que mantiveram a carga calculada acima do *threshold* superior, adiantando operações.

6.7 Considerações Parciais

Este capítulo apresentou a avaliação do modelo AutoElastic considerando diferentes cenários de testes e diferentes comportamento de carga da aplicação. Considerando a avaliação das diferentes abordagens do Live Thresholding, os experimentos que utilizaram a abordagem A_c englobaram praticamente todos os melhores resultados das métricas para todos os comportamentos de carga. Porém, em uma análise da métrica Custo foi definida como solução final para a técnica Live Thresholding a abordagem LT_{ce}. Os resultados desta abordagem foram selecionados para o restante das avaliações do capítulo. Juntamente com estes resultados, foram apresentados todos os resultados obtidos pela técnica de *thresholds* fixos considerando todas as combinações de parâmetros realizadas. Os resultados foram avaliados em conjunto com a execução de cada comportamento da aplicação com quantidade fixa de recursos. As avaliações demonstraram uma relação direta entre o consumo de recursos e o desempenho da aplicação. Quanto maior o consumo de recursos, melhor é o desempenho da aplicação. Por outro lado, se o objetivo é um menor consumo de recursos, o desempenho da aplicação é penalizado. Isso se deve ao fato da aplicação utilizada distribuir suas tarefas entre os recursos disponíveis, fazendo o uso de todos eles, acelerando a conclusão da aplicação.

Ainda, considerando uma comparação entre as diversas configurações de *thresholds* fixos, os resultados demonstraram que o *threshold* superior causa um maior impacto no desempenho e consumo de recursos em comparação com o *threshold* inferior. Particular-

mente, o *threshold* superior, quanto mais próximo de 100% pior é o desempenho e menor é o consumo de recursos. Em contrapartida, quando menor é esse *threshold* mais rapidamente ele é violado ocasionando uma quantidade maior de operações de elasticidade, aumentando o consumo de recursos, porém melhorando o desempenho da aplicação. Um ponto importante a ser destacado é que em questões de desempenho, a execução da aplicação sem elasticidade foi superada por todas as execuções com a elasticidade habilitada. Mesmo uma pior configuração de *thresholds* fixos resultou em melhores desempenhos. Ainda em questões de desempenho, é importante ressaltar os resultados obtidos pela técnica de Live Thresholding, os quais atingiram valores de desempenho muito próximos às melhores configurações de *threshols* fixos, superando-as em alguns casos. O ajuste fino de *thresholds* fixos permite que os melhores índices de desempenho sejam atingidos, porém com o Live Thresholding não há necessidade de parametrizações e o desempenho obtido é promissor.

Por fim, a elasticidade assíncrona e a técnica de *Aging* foram avaliadas neste capítulo. Os resultados mostraram que utilizando *Aging* foi possível evitar operações indesejadas e também adiantar operações necessárias durante a execução da aplicação. Já a elasticidade assíncrona foi avaliada demonstrando a capacidade de AutoElastic oferecer reconfigurações de recursos sem que a aplicação necessite interromper sua execução durante as operações. Os resultados apresentados apontam um tempo médio de 166,79 segundos para a criação de novas máquinas virtuais desde o ponto de identificação da necessidade de novos recursos até o ponto em que estes recursos estão aptos para receberem conexões. Ainda, em algumas execuções é possível notar que a criação de novos recursos foi iniciada porém não finalizada devido ao fato da aplicação terminar sua execução durante esta operação. Situações como essa podem ocorrer à medida que AutoElastic não tem conhecimento de quando a aplicação irá terminar sua execução.

7 CONCLUSÃO

Muitas abordagens de Computação em Nuvem empregam a elasticidade de recursos através da utilização de regras e ações que devem ser pré-definidas pelo usuário. Além disso, para oferecer estes recursos para aplicações HPC, algumas abordagens propõem modificações no código fonte da aplicação para que essa possa tirar proveito de um ambiente dinâmico, em que recursos podem ser adicionados ou removidos enquanto a aplicação está sendo executada. Considerando essas abordagens de configuração de regras e alteração do código da aplicação, foi desenvolvido o modelo AutoElastic para gerenciamento da elasticidade de recursos que isenta o usuário destas configurações. Para avaliar o modelo desenvolvido, foram realizados testes com uma aplicação paralela iterativa, utilizando diferentes cargas de processamento, em um ambiente OpenNebula.

A Seção 1.2 apresentou a seguinte questão de pesquisa: *Como seria um modelo de elasticidade em nuvem para aplicações HPC iterativas, transparente quanto a escrita da aplicação paralela, evitando custos elevados em termos de desempenho nos momentos de reconfiguração de recursos e processos?* Para respondê-la, AutoElastic conta com um Gerenciador que realiza realiza o monitoramento dos recursos da nuvem e realiza as reorganizações de recursos. Este Gerenciador age de forma independente da aplicação e da plataforma de nuvem, sendo necessário apenas que seja possível realizar uma conexão entre ele e o Front-End da nuvem. Além disso, o modelo conta com um *framework* que possibilita que a aplicação continue sendo executada em paralelo com operações de reorganização de recursos através da elasticidade assíncrona. A reorganização de processos pode ser realizada graças ao sistema de notificações, o qual atua de maneira que a aplicação receba novas conexões ou remova conexões ativas nos pontos exatos em que recursos estão prontos para serem adicionados ou removidos. Ainda, o modelo trabalha sem a necessidade de definição de parâmetros por parte do usuário, além de ser transparente quanto a escrita da aplicação graças ao *middleware* AutoElastic. No que diz respeito à elasticidade, o modelo conta com uma técnica nomeada Live Thresholding em que os *thresholds* são adaptados automaticamente durante a execução da aplicação.

No desenvolvimento do modelo, a Seção 4.2 apresentou três sentenças que aqui são reapresentadas seguidas de suas respostas:

(i) Quais mecanismos são necessários para prover elasticidade transparentemente nos níveis de usuário e aplicação como uma capacidade viável em aplicações HPC?

AutoElastic se destaca por atuar em nível de PaaS de uma nuvem, não impondo que o programador tenha que escrever ações de elasticidade no código da aplicação. Ainda nessa linha, AutoElastic oferece elasticidade assíncrona, que se mostrou pertinente para viabilizar o uso de aplicações HPC no contexto de nuvem computacional;

(ii) Quais aplicações HPC podem se beneficiar da elasticidade em nuvem e sob quais

restrições ela pode ser suportada?

A versão atual de AutoElastic suporta aplicações mestre-escravo iterativas, não necessitando informações prévias de seu comportamento. Essa abordagem é justificada pelo fato de que estas aplicações podem ser construídas com o estilo de programação de MPI 2.0 que segue a ideia de Sockets. Esse estilo permite que processos sejam conectados e desconectados facilmente à aplicação paralela, proporcionando um uso efetivo dos recursos disponíveis. Ainda, AutoElastic oferece um framework totalmente compatível com aplicações fortemente acopladas, então modelos como BSP e Divisão-e-Conquista podem ser adaptados no futuro para obter vantagem da elasticidade em nuvem;

(iii) Quais são as suposições mínimas para transparentemente suportar elasticidade em nuvem para aplicações HPC?

Assume-se que o usuário desenvolve uma aplicação iterativa, fornecendo diferentes modelos de máquinas virtuais para processos mestre e escravo. Em adição, o usuário possui a opção de submeter um SLA ao executar a aplicação. Se não informado, AutoElastic assume como padrão o dobro da quantidade de máquinas virtuais do ponto de início da aplicação como sendo a maior quantidade de recursos permitida.

Com o objetivo de não ser proibitivo em termos de desempenho quando operações de elasticidade devem ser realizadas, o modelo apresentou uma área de dados compartilhada entre os recursos da nuvem a qual é utilizada para uma política de comunicação entre o Gerenciador AutoElastic e a aplicação. O Gerenciador implementa notificações que tornam possível a comunicação com a aplicação, possibilitando que os únicos pontos em que a aplicação sofra interferência das ações de elasticidade sejam justamente os pontos de conexão ou desconexão com novos processos. Do lado da aplicação, dada sua natureza iterativa, pode ser realizada automaticamente uma inserção do código que implementa as notificações no início de cada ciclo de iteração. Esse sistema de notificação demonstrado torna possível que as operações de elasticidade sejam executadas em paralelo com a execução da aplicação, a qual não se envolve com tais operações.

No que diz respeito à avaliação do modelo, variar *thresholds*, cargas e cenários de execução é pertinente para a avaliação de uma política de provisionamento de recursos em nuvem adequada. Através da análise de resultados, foi possível demonstrar o impacto do uso de diferentes técnicas para disparar ações de elasticidade no desempenho e consumo de recursos da aplicação. Foram desenvolvidas métricas de avaliação de desempenho e consumo de recursos estendendo os conceitos de *Speedup* e Eficiência para a avaliação de resultados. Através dessas métricas notou-se que, quando utilizando abordagens tradicionais de elasticidade reativa baseada em *thresholds*, os *thresholds* que influenciam a alocação de recursos (*thresholds* superiores) causam um maior impacto no desempenho e

consumo de recursos em comparação com *thresholds* que influenciam a desalocação de recursos (*thresholds* inferiores). Porém, parametrizar o sistema de monitoramento de forma correta não é uma tarefa trivial. Para suprir isso, a abordagem de Live Thresholding apresentada demonstrou que é possível realizar adaptações nos *thresholds* baseando-se nos dados de monitoramento, evitando a necessidade de configurações prévias de parâmetros. É importante destacar que a técnica Live Thresholding aumenta a reatividade da elasticidade pois na medida que a carga aumenta, o *threshold* superior diminui indo de encontro com a carga e consequentemente ocasionando adição de novos recursos de maneira mais rápida.

Para análise de resultados foi realizada uma comparação de desempenho da execução de uma aplicação paralela com a quantidade mínima de recursos sem elasticidade. Os resultados demonstraram que, com o uso de uma abordagem de elasticidade baseada em *thresholds* fixos, considerando diversas execuções com diversas configurações de parâmetros, no melhor caso foi possível obter ganhos de desempenho de 47,6% a 57,9%. Enquanto que no pior caso, as configurações que obtiveram pior desempenho alcançaram índices de 28,4% a 39,5% melhores na mesma comparação. Já, com a utilização da abordagem Live Thresholding, esses índices foram de 50,4% a 59% melhores comparados com a execução da aplicação sem elasticidade. Isso demonstra uma proximidade entre os resultados do melhor caso com *thresholds* fixos, porém com Live Thresholding a necessidade de parametrização não existe.

7.1 Contribuições

O modelo AutoElastic buscou atender lacunas identificadas no estado da arte através da avaliação de trabalhos relacionados. Nesse sentido, o desenvolvimento do modelo resultou em 5 contribuições científicas que representam adições ao estado da arte que contempla pesquisa de programação paralela e elasticidade em nuvem. Durante a realização da pesquisa, essas contribuições foram publicadas em artigos científicos, como aqueles apresentados na Seção 7.3. A seguir, são destacadas as contribuições deste trabalho:

- Arcabouço para HPC + Elasticidade Automática: a arquitetura de AutoElastic, juntamente com componentes de comunicação e notificações permite que aplicações HPC iterativas sejam executadas de maneira transparente em nuvem com o recurso de elasticidade. Através de um gerenciador externo, que realiza o monitoramento e reorganizações de recursos, é possível que recursos sejam adicionados ou removidos ao ambiente. Atuando em nível de PaaS, AutoElastic oferece um *middleware* que transforma uma aplicação não elástica em uma aplicação elástica de maneira transparente. Hoje AutoElastic atua sobre aplicações mestre-escravo, mas esse *framework* pode ser usado para suportar outros tipos de aplicações, uma vez que todos os processos e não somente o mestre podem ler/escrever na região de memória com-

partilhada;

- Métricas de Avaliação de Elasticidade em Aplicações Paralelas: para a avaliação de aplicações com comportamento elástico, foram definidas novas métricas de desempenho e eficiência. *Speedup* Elástico (SE) foi desenvolvido para avaliar o desempenho de aplicações considerando sua natureza elástica em que a quantidade de recursos pode variar ao logo da execução da aplicação. Assim como SE, Eficiência Elástica (EE) considera a elasticidade para definir a eficiência de uma aplicação executada com este recurso;

- Técnica de *Aging* para Definição de Carga de Trabalho: durante o monitoramento dos recursos do ambiente em nuvem, podem ocorrer ruídos que podem causar operações de elasticidade indesejadas. A fim de evitar isso, o monitoramento realizado por AutoElastic utiliza uma técnica de *Aging* aplicada ao histórico da carga de processamento da aplicação para as tomadas de decisão do mecanismo de elasticidade. Através do *Aging* é realizada uma suavização do valor de decisão que é composto por todos os dados obtidos dos recursos;

- Elasticidade Assíncrona: um dos desafios era não causar nenhum impacto à execução da aplicação durante a reorganização de recursos. Para isso, a elasticidade assíncrona permite que o Gerenciador AutoElastic realize todas as operações de elasticidade utilizando uma área de dados compartilhada para gerar notificações que, em conjunto com o *middleware*, permitem disponibilizar estes recursos para a aplicação;

- Live Thresholding: técnicas de elasticidade proativas e reativas possuem certos desafios devido a necessidade de configurações. Tendo isso em vista, Live Thresholding oferece uma técnica híbrida em que a estratégia de *thresholds* é utilizada, porém seus valores são adaptados e recalculados em tempo de execução. Essa técnica elimina a necessidade de parametrizações prévias.

7.2 Trabalhos Futuros

Aqui, são destacadas algumas limitações do modelo AutoElastic que podem ser vistas como oportunidades de otimizações para trabalhos futuros:

- O modelo considera um simples modelo de SLA, em que o usuário informa apenas a quantidade máxima e mínima de recursos. Se o SLA não é informado, o Gerenciador AutoElastic assume que a quantidade máxima de recursos equivale ao dobro da quantidade de recursos utilizada na inicialização do monitoramento. Aqui, é possível incorporar as métricas utilizadas no modelo (Tempo, Energia, Custo) as quais podem ser combinadas com a carga do sistema para dirigir ações de elasticidade;

- Uma simples máquina física com n máquinas virtuais, em que n denota a quantidade de núcleos de processamento dentro da máquina física, é adicionada ou removida em cada operação de elasticidade. Apesar de apresentar resultados satisfatórios, outras possibilidades incluem uma adaptação no grão de elasticidade para melhorar a reatividade, reagindo de maneira mais rápida à variações de carga.

- Mesmo avaliando AutoElastic com uma aplicação simples, foram utilizados quatro comportamentos de carga para demonstrar o comportamento da elasticidade sobre diferentes situações. Todos eles foram gerados por uma equação, sendo classificados como cargas regulares. Portanto, investigações futuras podem observar outras aplicações paralelas irregulares.

7.3 Publicações

Ao logo de todo o período de realização da pesquisa, foram produzidos diversos artigos para publicação em revistas e eventos. Além dos artigos referentes à presente dissertação, foram produzidos outros artigos através da colaboração em projetos da universidade com outros pesquisadores do grupo de pesquisa. A seguir são listados os artigos publicados e os artigos que foram submetidos para avaliação:

- Artigos publicados em **revistas internacionais**:

  1. RIGHI, R. R.; **RODRIGUES, V. F.**; DA COSTA, C. A.; GALANTE, G.; BONA, L.; FERRETO, T. *AutoElastic: Automatic Resource Elasticity for High Performance Applications in the Cloud*. IEEE Transactions on Cloud Computing, v. 4, n. 1, p. 6–19, Jan 2016.
  2. RIGHI, R. R.; COSTA, C. A.; JUNIOR, L. G. S.; **RODRIGUES, V. F.**; GOMES, R. Q.; GUERREIRO, V. M. *Exploiting Data-Parallelism on Multicore and SMT Systems for Implementing the Fractal Image Compressing Problem*. Computer and Information Science 2016.
  3. RIGHI, R. R.; DA COSTA, C. A.; **RODRIGUES, V. F.**; ROSTIROLLA, G. *Joint-analysis of performance and energy consumption when enabling cloud elasticity for synchronous HPC applications*. Concurrency and Computation, v. 1, p. 1–14, 2015.
  4. RIGHI, R. R.; **RODRIGUES, V. F.**; DA COSTA, C. A.; KREUTZ, D. L.; HEISS, H.U. *Towards Cloud-based Asynchronous Elasticity for Iterative HPC Applications*. Journal of Physics. Conference Series (Online), v. 649, p. 012006, 2015.

- Artigos publicados em **eventos internacionais**:

5. **RODRIGUES, V. F.**; ROSTIROLLA, G.; RIGHI, R. R.; COSTA, C. A. *Towards an Elastic Energy Consumption Model for HPC Applications in the Cloud*. 16th International Conference on Parallel and Distributed Computing, Applications and Technologies (PDCAT-15), 2015, Jeju, Korea.

6. ROSTIROLLA, G.; RIGHI, R. R.; **RODRIGUES, V. F.**; VELHO, P.; PADOIN, E. L. *GreenHPC: a novel framework to measure energy consumption on HPC applications*. In: 2015 Sustainable Internet and ICT for Sustainability (SustainIT), 2015, Madrid. 2015 Sustainable Internet and ICT for Sustainability (SustainIT). p. 1-8.

7. RIGHI, R. R.; GOMES, R. Q.; **RODRIGUES, V. F.**; DA COSTA, C. A.; ALBERTI, A. M. *MigBSP++: Improving Process Rescheduling on Bulk-Synchronous Parallel Applications*. In: 2015 IEEE/ACS 12th International Conference on Computer Systems and Applications (AICCSA), 2015, Marrakech, Morocco. 2015 IEEE/ACS 12th International Conference on Computer Systems and Applications (AICCSA).

8. RIGHI, R. R.; VEITH, A.; **RODRIGUES, V. F.**; ROSTIROLLA, G.; DA COSTA, C. A.; FARIAS, K.; ALBERTI, A. M. *Rescheduling and Checkpointing as Strategies to Run Synchronous Parallel Programs on P2P Desktop Grids*. In: Symposium on Applied Computing (SAC 2015), 2015, Salamanca. Proc. of the Int. Symposium on Applied Computing (SAC 2015), 2015. v. 14. p. 501-504.

9. RIGHI, R. R.; VEITH, A. S.; ROSTIROLLA, G.; **RODRIGUES, V. F.**; COSTA, C. A. *BSPonP2P: Towards Running Bulk-Synchronous Parallel Applications on P2P Desktop Grids*. In: The 2015 International Conference on Parallel and Distributed Processing Techniques and Applications (PDPTA 2015), 2015, Las Vegas, USA. Proceedings of the 21st International Conference on Parallel and Distributed Processing Techniques and Applications (PDPTA 2015), 2015. p. 189-196.

10. **RODRIGUES, V. F.**; ROSTIROLLA, G.; RIGHI, R. R.; DA COSTA, C. A.; BARBOSA, J. L. V. *Cloud elasticity for HPC applications: Observing energy, performance and cost*. In: 2015 XLI Latin American Computing Conference (CLEI), 2015, Arequipa. 2015 Latin American Computing Conference (CLEI). v. 41. p. 1-200.

11. PIRES, J. C. S.; **RODRIGUES, V. F.**; COSTA, C. A.; RIGHI, R. R. *BSPMON: Sistema de Monitoramento de Recursos Preditivo para Aplicações Paralelas em Cloud Computing*. In: 2nd Ibero-Americana Computação Aplicada 2014 (CIACA 2014), 2014, Porto. Proceedings of the 2nd Ibero-Americana Computação Aplicada 2014. Lisboa: IADIS Press, 2014.

12. RIGHI, R. R.; **RODRIGUES, V. F.**; DA COSTA, C. A.; CHIWIACOWSKY, L.; KREUTZ, D. L.; ANDRADE, A. *A novel framework for supporting the exponential worldwide adoption of electronic transactions*. In: 2014 IEEE/ACS 11th International Conference on Computer Systems and Applications (AICCSA), 2014, Doha. 2014 IEEE/ACS 11th International Conference on Computer Systems and Applications (AICCSA). v. 11. p. 198-205.

- Artigos publicados em **eventos nacionais**:

13. **RODRIGUES, V. F.**; COSTA, C. A.; RIGHI, R. R.; KREUTZ, D. L. *Explorando a Elasticidade Assíncrona em Nuvem para Aplicações Paralelas Iterativas*. In: XV Simpósio em Sistemas Computacionais de Alto Desempenho (WSCAD 2014), 2014, São José dos Campos, SP. Anais do XV Simpósio em Sistemas Computacionais de Alto Desempenho (WSCAD 2014), 2014. p. 63-74.

- Artigos publicados em **eventos regionais**:

14. **RODRIGUES, V. F.**; ROSTIROLLA, G.; RIGHI, R. R.; DA COSTA, C. A. *Combinando Elasticidade Reativa e Preditiva para a Execução de Aplicações Paralelas Iterativas em Nuvem*. In: 16ª Escola Regional de Alto Desempenho (ERAD 2016), 2016, São Leopoldo, RS. Anais da 16ª Escola Regional de Alto Desempenho (ERAD2016), 2016.

15. ROSTIROLLA, G.; **RODRIGUES, V. F.**; RIGHI, R. R.; DA COSTA, C. A. *Correlacionando Modelos de Energia para Aplicações Paralelas em Nuvem*. In: 16ª Escola Regional de Alto Desempenho (ERAD 2016), 2016, São Leopoldo, RS. Anais da 16ª Escola Regional de Alto Desempenho (ERAD2016), 2016.

16. ROSTIROLLA, G.; **RODRIGUES, V. F.**; RIGHI, R. R.; COSTA, C. A. *Analyzing Energy Consumption of Elastic HPC Applications in the Cloud*. In: XIII Workshop de Processamento Paralelo e Distribuído (WSPPD 2015), 2015, Porto Alegre, RS. Anais do XIII Workshop de Processamento Paralelo e Distribuído (WSPPD 2015), 2015. p. 1-4.

17. **RODRIGUES, V. F.**; ROSTIROLLA, G.; RIGHI, R. R.; COSTA, C. A. *Analyzing Performance and Efficiency of HPC Applications in the Cloud*. In: XIII Workshop de Processamento Paralelo e Distribuído (WSPPD 2015), 2015, Porto Alegre, RS. Anais do XIII Workshop de Processamento Paralelo e Distribuído (WSPPD 2015), 2015. p. 25-28.

18. **RODRIGUES, V. F.**; ROSTIROLLA, G.; RIGHI, R. R. *Elasticidade Assíncrona: Transferência Não Bloqueante de VMs para Viabilizar a Reorganização de Aplicações HPC em Cloud Computing*. In: 13ª Escola Regional de Redes

de Computadores (ERRC 2015), 2015, Passo Fundo, RS. Anais da 13ª Escola Regional de Redes de Computadores (ERRC 2015), 2015.

19. ROSTIROLLA, G.; **RODRIGUES, V. F.**; RIGHI, R. R. *Um Modelo de Consumo de Energia para Ambientes de Nuvem com Elasticidade.* In: 13ª Escola Regional de Redes de Computadores (ERRC 2015), 2015, Passo Fundo, RS. Anais da 13ª Escola Regional de Redes de Computadores (ERRC 2015), 2015.
20. **RODRIGUES, V. F.**; ROSTIROLLA, G.; RIGHI, R. R. *Elasticidade Automática Baseada em Thresholds para Aplicações Paralelas Iterativas em Nuvem Computacional.* In: 15ª Escola Regional de Alto Desempenho (ERAD 2015), 2015, Gramado, RS. Anais da 15ª Escola Regional de Alto Desempenho (ERAD 2015), 2015. v. 15. p. 107-108.
21. ROSTIROLLA, G.; **RODRIGUES, V. F.**; RIGHI, R. R. *Análise da Economia de Energia Através do Desligamento de Nós Ociosos em um Cluster com Processadores ARM.* In: 15ª Escola Regional de Alto Desempenho (ERAD 2015), 2015, Gramado, RS. Anais da 15ª Escola Regional de Alto Desempenho (ERAD 2015), 2015. v. 15. p. 97-98.
22. **RODRIGUES, V. F.**; ROSTIROLLA, G.; RIGHI, R. R. *Elasticidade Reativa em Nuvem para Aplicações de Alto Desempenho.* In: 12ª Escola Regional de Redes de Computadores (ERRC 2014), 2014, Canoas, RS. Anais da 12ª Escola Regional de Redes de Computadores (ERRC 2014), 2014.
23. ROSTIROLLA, G.; **RODRIGUES, V. F.**; RIGHI, R. R.; VELHO, P. A. M. C. *Análise de Desempenho e Consumo de um Cluster Baseado em Computadores de Placa Única.* In: 12ª Escola Regional de Redes de Computadores (ERRC 2014), 2014, Canoas, RS. Anais da 12ª Escola Regional de Redes de Computadores (ERRC 2014), 2014.

- Artigos submetidos para avaliação:

a. RIGHI, R. R.; **RODRIGUES, V. F.**; ROSTIROLLA, G.; DA COSTA, C. A.; ROLOFF, E.; NAVAUX, P. O. A. *A Lightweight Plug-and-Play Elasticity Service for Self-Organizing Resource Provisioning on Parallel Applications.* IEEE Transactions on Services Computing.

b. RIGHI, R. R.; ROSTIROLLA, G.; **RODRIGUES, V. F.**; BARBOSA, J. L. V.; DA COSTA, C. A.; ALBERTI, A. M.; CHANG, V. *Towards Enabling Live Thresholding as Utility to Manage Elastic HPC Applications in the Cloud.* IEEE Transactions on Cloud Computing.

c. **RODRIGUES, V. F.**; ROSTIROLLA, G.; RIGHI, R. R.; DA COSTA, C. A.;

BARBOSA, J. L. V. *Impact of Thresholds and Load Patterns when Executing HPC Applications with Cloud Elasticity.* CLEI Electronic Journal.

d. RIGHI, R. R.; GOMES, R. Q.; **RODRIGUES, V. F.**; DA COSTA, C. A.; ALBERTI, A. M.; PILLA, L. L.; NAVAUX, P. O. A. *MigPF: Towards on Self-Organizing Process Rescheduling of Bulk-Synchronous Parallel Applications.* Future Generation Computing Systems.

e. ROSTIROLLA, G.; **RODRIGUES, V. F.**; RIGHI, R. R.; DA COSTA, C. A.; KREUTZ, D. L.; SINGH. D. *EME: an Energy Model for Elastic Cloud-based HPC Applications.* Revista Brasileira de Computação Aplicada.

f. RIGHI, R. R.; GUERREIRO, V. M.; ROSTIROLLA, G.; **RODRIGUES, V. F.**; DA COSTA, C. A.; CHIWIACOWSKY, L. D. *Using Computational Geometry to Improve Process Rescheduling on Round-Based Parallel Applications.* Scalable Computing: Practice and Experience.

REFERÊNCIAS

AL-HAIDARI, F.; SQALLI, M.; SALAH, K. Impact of CPU Utilization Thresholds and Scaling Size on Autoscaling Cloud Resources. In: CLOUD COMPUTING TECHNOLOGY AND SCIENCE (CLOUDCOM), 2013 IEEE 5TH INTERNATIONAL CONFERENCE ON, 2013. **Anais...** IEEE, 2013. v. 2, p. 256–261.

ALMASI, G. S.; GOTTLIEB, A. **Highly Parallel Computing**. Redwood City, CA, USA: Benjamin-Cummings Publishing Co., Inc., 1989.

ANDERSON, D. BOINC: a system for public-resource computing and storage. In: GRID COMPUTING, 2004. PROCEEDINGS. FIFTH IEEE/ACM INTERNATIONAL WORKSHOP ON, 2004. **Anais...** IEEE, 2004. p. 4–10.

AZMANDIAN, F.; MOFFIE, M.; DY, J.; ASLAM, J.; KAELI, D. Workload Characterization at the Virtualization Layer. In: MODELING, ANALYSIS SIMULATION OF COMPUTER AND TELECOMMUNICATION SYSTEMS (MASCOTS), 2011 IEEE 19TH INTERNATIONAL SYMPOSIUM ON, 2011. **Anais...** IEEE, 2011. p. 63–72.

BALIGA, J.; AYRE, R.; HINTON, K.; TUCKER, R. Green Cloud Computing: balancing energy in processing, storage, and transport. **Proceedings of the IEEE**, New York, NY, USA, v. 99, n. 1, p. 149–167, 2011.

BASET, S. A. Cloud SLAs: present and future. **SIGOPS Oper. Syst. Rev.**, New York, NY, USA, v. 46, n. 2, p. 57–66, July 2012.

BEERNAERT, L.; MATOS, M.; VILAçA, R.; OLIVEIRA, R. Automatic elasticity in OpenStack. In: WORKSHOP ON SECURE AND DEPENDABLE MIDDLEWARE FOR CLOUD MONITORING AND MANAGEMENT, 2012, New York, NY, USA. **Proceedings...** ACM, 2012. p. 2:1–2:6. (SDMCMM '12).

BERSANI, M. M.; BIANCULLI, D.; DUSTDAR, S.; GAMBI, A.; GHEZZI, C.; KRSTIć, S. Towards the Formalization of Properties of Cloud-based Elastic Systems. In: INTERNATIONAL WORKSHOP ON PRINCIPLES OF ENGINEERING SERVICE-ORIENTED AND CLOUD SYSTEMS, 6., 2014, New York, NY, USA. **Proceedings...** ACM, 2014. p. 38–47. (PESOS 2014).

BING, H.; YING-LAN, F.; BAI, L. Y. e. Research and Improvement of Congestion Control Algorithms Based on TCP Protocol. In: SOFTWARE ENGINEERING, 2009. WCSE '09. WRI WORLD CONGRESS ON, 2009. **Anais...** IEEE, 2009. v. 1, p. 440–443.

BOX, G. E. P.; JENKINS, G. M.; REINSEL, G. C. **Time series analysis** : forecasting and control. 4. ed. Hoboken, NJ, USA: John Wiley & Sons, Ltd., 2008.

BRYANT, R.; TUMANOV, A.; IRZAK, O.; SCANNELL, A.; JOSHI, K.; HILTUNEN, M.; LAGAR-CAVILLA, A.; LARA, E. de. Kaleidoscope: cloud micro-elasticity via vm state coloring. In: COMPUTER SYSTEMS, 2011, New York, NY, USA. **Proceedings...** ACM, 2011. p. 273–286. (EuroSys '11).

BUYYA, R. **High Performance Cluster Computing**: programming and applications. 1st. ed. Upper Saddle River, NJ, USA: Prentice Hall PTR, 1999.

CAI, B.; XU, F.; YE, F.; ZHOU, W. Research and application of migrating legacy systems to the private cloud platform with cloudstack. In: AUTOMATION AND LOGISTICS (ICAL), 2012 IEEE INTERNATIONAL CONFERENCE ON, 2012. **Anais...** IEEE, 2012. p. 400–404.

CALHEIROS, R. N.; RANJAN, R.; BELOGLAZOV, A.; DE ROSE, C. A. F.; BUYYA, R. CloudSim: a toolkit for modeling and simulation of cloud computing environments and evaluation of resource provisioning algorithms. **Software: Practice and Experience**, Hoboken, NJ, USA, v. 41, n. 1, p. 23–50, 2011.

CHANGQING, G.; QINGHUI, W.; GUANGXING, W. The impact of TCP segment size and routing change on congestion control protocol performance in mobile ad hoc networks. In: WIRELESS COMMUNICATIONS, NETWORKING AND MOBILE COMPUTING, 2005. PROCEEDINGS. 2005 INTERNATIONAL CONFERENCE ON, 2005. **Anais...** IEEE, 2005. v. 2, p. 820–823.

CHIU, D.; AGRAWAL, G. Evaluating caching and storage options on the Amazon Web Services Cloud. In: GRID COMPUTING (GRID), 2010 11TH IEEE/ACM INTERNATIONAL CONFERENCE ON, 2010. **Anais...** IEEE, 2010. p. 17–24.

COMANESCU, M. Implementation of time-varying observers used in direct field orientation of motor drives by trapezoidal integration. In: POWER ELECTRONICS, MACHINES AND DRIVES (PEMD 2012), 6TH IET INTERNATIONAL CONFERENCE ON, 2012. **Anais...** IET, 2012. p. 1–6.

COPIL, G.; MOLDOVAN, D.; TRUONG, H.-L.; DUSTDAR, S. SYBL: an extensible language for controlling elasticity in cloud applications. In: CLUSTER, CLOUD AND GRID COMPUTING (CCGRID), 2013 13TH IEEE/ACM INTERNATIONAL SYMPOSIUM ON, 2013. **Anais...** IEEE, 2013. p. 112–119.

COSTA, R.; BRASILEIRO, F.; SOUZA FILHO, G. L. de; SOUSA, D. M. **Just in Time Clouds**: enabling highly-elastic public clouds over low scale amortized resources. Campina Grande, PB, BR: Universidade Federal de Campina Grande, 2010.

COUTINHO, E. F.; CARVALHO SOUSA, F. R. de; REGO, P. A. L.; GOMES, D. G.; SOUZA, J. N. de. Elasticity in cloud computing: a survey. **Annals of Telecommunications - Annales des Télécommunications**, Paris, França, p. 1–21, 2014.

COUTINHO, E. F.; PAILLARD, G.; SOUZA, J. N. de. Performance Analysis on Scientific Computing and Cloud Computing Environments. In: EURO AMERICAN CONFERENCE ON TELEMATICS AND INFORMATION SYSTEMS, 7., 2014, New York, NY, USA. **Proceedings...** ACM, 2014. p. 5:1–5:6. (EATIS '14).

DAWOUD, W.; TAKOUNA, I.; MEINEL, C. Elastic VM for Cloud Resources Provisioning Optimization. In: ABRAHAM, A.; LLORET MAURI, J.; BUFORD, J.; SUZUKI, J.; THAMPI, S. (Ed.). **Advances in Computing and Communications**. Heidelberg, Germany: Springer Berlin Heidelberg, 2011. p. 431–445. (Communications in Computer and Information Science, v. 190).

DEAN, J.; GHEMAWAT, S. MapReduce: simplified data processing on large clusters. **Commun. ACM**, New York, NY, USA, v. 51, n. 1, p. 107–113, Jan. 2008.

DOWD, K.; SEVERANCE, C. R. **High performance computing - RISC architectures, optimization and benchmarks**. 2nd. ed. Sebastopol, CA, USA: O'Reilly Media, Inc., 1998.

DUTTA, S.; GERA, S.; VERMA, A.; VISWANATHAN, B. SmartScale: automatic application scaling in enterprise clouds. In: CLOUD COMPUTING (CLOUD), 2012 IEEE 5TH INTERNATIONAL CONFERENCE ON, 2012. **Anais...** IEEE, 2012. p. 221–228.

EXPóSITO, R. R.; TABOADA, G. L.; RAMOS, S.; TOURIñO, J.; DOALLO, R. Evaluation of Messaging Middleware for High-performance Cloud Computing. **Personal Ubiquitous Comput.**, London, UK, v. 17, n. 8, p. 1709–1719, Dec. 2013.

FRINCU, M. E.; GENAUD, S.; GOSSA, J. Comparing Provisioning and Scheduling Strategies for Workflows on Clouds. In: IEEE 27TH INTERNATIONAL SYMPOSIUM ON PARALLEL AND DISTRIBUTED PROCESSING WORKSHOPS AND PHD FORUM, 2013., 2013, Washington, DC, USA. **Proceedings...** IEEE Computer Society, 2013. p. 2101–2110. (IPDPSW '13).

FU, L.; GONDI, C. Cloud Computing hosting. In: COMPUTER SCIENCE AND INFORMATION TECHNOLOGY (ICCSIT), 2010 3RD IEEE INTERNATIONAL CONFERENCE ON, 2010. **Anais...** IEEE, 2010. v. 3, p. 194–198.

FUJII, T.; KIMURA, M. Analysis Results on Productivity Variation in Force.com Applications. In: SOFTWARE MEASUREMENT, 2011 JOINT CONFERENCE OF THE 21ST INT'L WORKSHOP ON AND 6TH INT'L CONFERENCE ON SOFTWARE PROCESS AND PRODUCT MEASUREMENT (IWSM-MENSURA), 2011. **Anais...** IEEE, 2011. p. 314–317.

GALANTE, G.; BONA, L. C. E. D. A programming-level approach for elasticizing parallel scientific applications. **Journal of Systems and Software**, Amsterdam, The Netherlands, The Netherlands, v. 110, p. 239 – 252, 2015.

GALANTE, G.; BONA, L. de. A Survey on Cloud Computing Elasticity. In: UTILITY AND CLOUD COMPUTING (UCC), 2012 IEEE FIFTH INTERNATIONAL CONFERENCE ON, 2012. **Anais...** IEEE, 2012. p. 263–270.

GUTIERREZ-GARCIA, J. O.; SIM, K. M. Agent-based Cloud bag-of-tasks execution. **Journal of Systems and Software**, Amsterdam, The Netherlands, The Netherlands, v. 104, p. 17 – 31, 2015.

HAN, R.; GUO, L.; GHANEM, M. M.; GUO, Y. Lightweight Resource Scaling for Cloud Applications. **Cluster Computing and the Grid, IEEE International Symposium on**, Los Alamitos, CA, USA, v. 0, p. 644–651, 2012.

HAYES, B. Cloud Computing. **Commun. ACM**, New York, NY, USA, v. 51, n. 7, p. 9–11, July 2008.

HENDRICKSON, B. Computational science: emerging opportunities and challenges. **Journal of Physics: Conference Series**, Bristol, England, v. 180, n. 1, p. 012013, 2009.

HENNESSY, J. L.; PATTERSON, D. A. **Computer Architecture, Fifth Edition**: a quantitative approach. 5th. ed. San Francisco, CA, USA: Morgan Kaufmann Publishers Inc., 2011.

HERBST, N. R.; HUBER, N.; KOUNEV, S.; AMREHN, E. Self-adaptive Workload Classification and Forecasting for Proactive Resource Provisioning. In: ACM/SPEC INTERNATIONAL CONFERENCE ON PERFORMANCE ENGINEERING, 4., 2013, New York, NY, USA. **Proceedings...** ACM, 2013. p. 187–198. (ICPE '13).

IMAI, S.; CHESTNA, T.; VARELA, C. A. Elastic Scalable Cloud Computing Using Application-Level Migration. In: IEEE/ACM FIFTH INTERNATIONAL CONFERENCE ON UTILITY AND CLOUD COMPUTING, 2012., 2012, Washington, DC, USA. **Proceedings...** IEEE Computer Society, 2012. p. 91–98. (UCC '12).

ISLAM, S.; LEE, K.; FEKETE, A.; LIU, A. How a consumer can measure elasticity for cloud platforms. In: WOSP/SIPEW INTERNATIONAL CONFERENCE ON PERFORMANCE ENGINEERING, 2012, New York, NY, USA. **Proceedings...** ACM, 2012. p. 85–96. (ICPE '12).

JACKSON, K. R.; RAMAKRISHNAN, L.; MURIKI, K.; CANON, S.; CHOLIA, S.; SHALF, J.; WASSERMAN, H. J.; WRIGHT, N. J. Performance Analysis of High Performance Computing Applications on the Amazon Web Services Cloud. In: IEEE SECOND INTERNATIONAL CONFERENCE ON CLOUD COMPUTING TECHNOLOGY AND SCIENCE, 2010., 2010, Washington, DC, USA. **Proceedings...** IEEE Computer Society, 2010. p. 159–168. (CLOUDCOM '10).

JAMSHIDI, P.; AHMAD, A.; PAHL, C. Autonomic Resource Provisioning for Cloud-based Software. In: INTERNATIONAL SYMPOSIUM ON SOFTWARE ENGINEERING FOR ADAPTIVE AND SELF-MANAGING SYSTEMS, 9., 2014, New York, NY, USA. **Proceedings...** ACM, 2014. p. 95–104. (SEAMS 2014).

JAMSHIDI, P.; AHMAD, A.; PAHL, C. Autonomic Resource Provisioning for Cloud-based Software. In: INTERNATIONAL SYMPOSIUM ON SOFTWARE ENGINEERING FOR ADAPTIVE AND SELF-MANAGING SYSTEMS, 9., 2014, New York, NY, USA. **Proceedings...** ACM, 2014. p. 95–104. (SEAMS 2014).

JENNINGS, B.; STADLER, R. Resource Management in Clouds: survey and research challenges. **Journal of Network and Systems Management**, New York, NY, USA, p. 1–53, 2014.

KNAUTH, T.; FETZER, C. Scaling Non-elastic Applications Using Virtual Machines. In: CLOUD COMPUTING (CLOUD), 2011 IEEE INTERNATIONAL CONFERENCE ON, 2011. **Anais...** IEEE, 2011. p. 468 –475.

KOUKI, Y.; OLIVEIRA, F. A. d.; DUPONT, S.; LEDOUX, T. A Language Support for Cloud Elasticity Management. In: CLUSTER, CLOUD AND GRID COMPUTING (CCGRID), 2014 14TH IEEE/ACM INTERNATIONAL SYMPOSIUM ON, 2014. **Anais...** IEEE, 2014. p. 206–215.

KROESE, D. P.; BRERETON, T.; TAIMRE, T.; BOTEV, Z. I. Why the Monte Carlo method is so important today. **Wiley Interdisciplinary Reviews: Computational Statistics**, Hoboken, New Jersey, USA, v. 6, n. 6, p. 386–392, 2014.

KUMAR, K.; FENG, J.; NIMMAGADDA, Y.; LU, Y.-H. Resource Allocation for Real-Time Tasks Using Cloud Computing. In: COMPUTER COMMUNICATIONS AND NETWORKS (ICCCN), 2011 PROCEEDINGS OF 20TH INTERNATIONAL CONFERENCE ON, 2011. **Anais...** IEEE, 2011. p. 1 –7.

KUMAR, V. **Introduction to Parallel Computing**. 2nd. ed. Boston, MA, USA: Addison-Wesley Longman Publishing Co., Inc., 2002.

LEE, Y.; AVIZIENIS, R.; BISHARA, A.; XIA, R.; LOCKHART, D.; BATTEN, C.; ASANOVIC, K. Exploring the tradeoffs between programmability and efficiency in data-parallel accelerators. In: COMPUTER ARCHITECTURE (ISCA), 2011 38TH ANNUAL INTERNATIONAL SYMPOSIUM ON, 2011. **Anais...** ACM, 2011. p. 129–140.

LEHRIG, S.; EIKERLING, H.; BECKER, S. Scalability, Elasticity, and Efficiency in Cloud Computing: a systematic literature review of definitions and metrics. In: INTERNATIONAL ACM SIGSOFT CONFERENCE ON QUALITY OF SOFTWARE ARCHITECTURES, 11., 2015, New York, NY, USA. **Proceedings...** ACM, 2015. p. 83–92. (QoSA '15).

LIN, W.; WANG, J. Z.; LIANG, C.; QI, D. A Threshold-based Dynamic Resource Allocation Scheme for Cloud Computing. **Procedia Engineering**, Amsterdam, Netherlands, v. 23, n. 0, p. 695 – 703, 2011. {PEEA} 2011.

LONEA, A.; POPESCU, D.; PROSTEAN, O. A survey of management interfaces for eucalyptus cloud. In: APPLIED COMPUTATIONAL INTELLIGENCE AND INFORMATICS (SACI), 2012 7TH IEEE INTERNATIONAL SYMPOSIUM ON, 2012. **Anais...** IEEE, 2012. p. 261 –266.

LORIDO-BOTRAN, T.; MIGUEL-ALONSO, J.; LOZANO, J. A Review of Auto-scaling Techniques for Elastic Applications in Cloud Environments. **Journal of Grid Computing**, Houten, Netherlands, v. 12, n. 4, p. 559–592, 2014.

LUSK, E. MPI-2: standards beyond the message-passing model. In: MASSIVELY PARALLEL PROGRAMMING MODELS, 1997. PROCEEDINGS. THIRD WORKING CONFERENCE ON, 1997. **Anais...** IEEE, 1997. p. 43–49.

MAO, M.; HUMPHREY, M. Auto-scaling to Minimize Cost and Meet Application Deadlines in Cloud Workflows. In: INTERNATIONAL CONFERENCE FOR HIGH PERFORMANCE COMPUTING, NETWORKING, STORAGE AND ANALYSIS, 2011., 2011, New York, NY, USA. **Proceedings...** ACM, 2011. p. 49:1–49:12. (SC '11).

MAO, M.; LI, J.; HUMPHREY, M. Cloud auto-scaling with deadline and budget constraints. In: GRID COMPUTING (GRID), 2010 11TH IEEE/ACM INTERNATIONAL CONFERENCE ON, 2010. **Anais...** IEEE, 2010. p. 41 –48.

MARIANI, S.; TRUONG, H.-L.; COPIL, G.; OMICINI, A.; DUSTDAR, S. Coordination-aware Elasticity. In: IEEE/ACM INTERNATIONAL CONFERENCE ON UTILITY AND CLOUD COMPUTING (UCC 2014), 7., 2014, London, UK. **Anais...** IEEE Computer Society, 2014. p. 56–63.

MARSHALL, P.; KEAHEY, K.; FREEMAN, T. Elastic Site: using clouds to elastically extend site resources. In: IEEE/ACM INTERNATIONAL CONFERENCE ON CLUSTER, CLOUD AND GRID COMPUTING, 2010., 2010, Washington, DC, USA. **Proceedings...** IEEE Computer Society, 2010. p. 43–52. (CCGRID '10).

MARTIN, A.; KNAUTH, T.; CREUTZ, S.; BECKER, D.; WEIGERT, S.; FETZER, C.; BRITO, A. Low-Overhead Fault Tolerance for High-Throughput Data Processing Systems. In: DISTRIBUTED COMPUTING SYSTEMS (ICDCS), 2011 31ST INTERNATIONAL CONFERENCE ON, 2011. **Anais...** IEEE, 2011. p. 689–699.

MARTIN, P.; BROWN, A.; POWLEY, W.; VAZQUEZ-POLETTI, J. L. Autonomic management of elastic services in the cloud. In: IEEE SYMPOSIUM ON COMPUTERS AND COMMUNICATIONS, 2011., 2011, Washington, DC, USA. **Proceedings...** IEEE Computer Society, 2011. p. 135–140. (ISCC '11).

MATOS, M.; CORREIA JR., A.; PEREIRA, J.; OLIVEIRA, R. Serpentine: adaptive middleware for complex heterogeneous distributed systems. In: ACM SYMPOSIUM ON APPLIED COMPUTING, 2008., 2008, New York, NY, USA. **Proceedings...** ACM, 2008. p. 2219–2223. (SAC '08).

MELL, P. M.; GRANCE, T. **SP 800-145. The NIST Definition of Cloud Computing**. Gaithersburg, MD, United States: National Institute of Standards & Technology, 2011.

MICHON, E.; GOSSA, J.; GENAUD, S. Free Elasticity and Free CPU Power for Scientific Workloads on IaaS Clouds. In: PARALLEL AND DISTRIBUTED SYSTEMS (ICPADS), 2012 IEEE 18TH INTERNATIONAL CONFERENCE ON, 2012. **Anais...** IEEE, 2012. p. 85 –92.

MILOJICIC, D.; LLORENTE, I. M.; MONTERO, R. S. OpenNebula: a cloud management tool. **Internet Computing, IEEE**, New York, NY, USA, v. 15, n. 2, p. 11–14, march-april 2011.

MORENO, I.; XU, J. Customer-aware resource overallocation to improve energy efficiency in realtime Cloud Computing data centers. In: SERVICE-ORIENTED COMPUTING AND APPLICATIONS (SOCA), 2011 IEEE INTERNATIONAL CONFERENCE ON, 2011. **Anais...** IEEE, 2011. p. 1 –8.

ORGERIE, A.-C.; ASSUNCAO, M. D. d.; LEFEVRE, L. A Survey on Techniques for Improving the Energy Efficiency of Large-scale Distributed Systems. **ACM Comput. Surv.**, New York, NY, USA, v. 46, n. 4, p. 47:1–47:31, Mar. 2014.

PUTHAL, D.; SAHOO, B.; MISHRA, S.; SWAIN, S. Cloud Computing Features, Issues, and Challenges: a big picture. In: COMPUTATIONAL INTELLIGENCE AND NETWORKS (CINE), 2015 INTERNATIONAL CONFERENCE ON, 2015. **Anais...** IEEE, 2015. p. 116–123.

R. RIGHI, R. d.; RODRIGUES, V. F.; COSTA, C. A. da; GALANTE, G.; BONA, L. C. E. de; FERRETO, T. AutoElastic: automatic resource elasticity for high performance applications in the cloud. **IEEE Transactions on Cloud Computing**, [S.l.], v. 4, n. 1, p. 6–19, Jan 2016.

RAJAN, D.; CANINO, A.; IZAGUIRRE, J. A.; THAIN, D. Converting a High Performance Application to an Elastic Cloud Application. In: IEEE THIRD INTERNATIONAL CONFERENCE ON CLOUD COMPUTING TECHNOLOGY AND SCIENCE, 2011., 2011, Washington, DC, USA. **Proceedings...** IEEE Computer Society, 2011. p. 383–390. (CLOUDCOM '11).

RAJARAMAN, V. Cloud computing. **Resonance**, New Delhi, India, v. 19, n. 3, p. 242–258, 2014.

RAVEENDRAN, A.; BICER, T.; AGRAWAL, G. A Framework for Elastic Execution of Existing MPI Programs. In: IEEE INT. SYMPOSIUM ON PARALLEL AND DISTRIBUTED PROCESSING WORKSHOPS AND PHD FORUM, 2011., 2011, Washington, DC, USA. **Proceedings...** IEEE Computer Society, 2011. p. 940–947. (IPDPSW '11).

ROLOFF, E.; BIRCK, F.; DIENER, M.; CARISSIMI, A.; NAVAUX, P. Evaluating High Performance Computing on the Windows Azure Platform. In: CLOUD COMPUTING (CLOUD), 2012 IEEE 5TH INTERNATIONAL CONFERENCE ON, 2012. **Anais...** IEEE, 2012. p. 803 –810.

SAH, S.; JOSHI, S. Scalability of efficient and dynamic workload distribution in autonomic cloud computing. In: ISSUES AND CHALLENGES IN INTELLIGENT COMPUTING TECHNIQUES (ICICT), 2014 INTERNATIONAL CONFERENCE ON, 2014. **Anais...** IEEE, 2014. p. 12–18.

SHARMA, U.; SHENOY, P.; SAHU, S.; SHAIKH, A. A Cost-Aware Elasticity Provisioning System for the Cloud. In: INTERNATIONAL CONFERENCE ON DISTRIBUTED COMPUTING SYSTEMS, 2011., 2011, Washington, DC, USA. **Proceedings...** IEEE Computer Society, 2011. p. 559–570. (ICDCS '11).

SHEN, Z.; SUBBIAH, S.; GU, X.; WILKES, J. CloudScale: elastic resource scaling for multi-tenant cloud systems. In: ACM SYMPOSIUM ON CLOUD COMPUTING, 2., 2011, New York, NY, USA. **Proceedings...** ACM, 2011. p. 5:1–5:14. (SOCC '11).

SPINNER, S.; KOUNEV, S.; ZHU, X.; LU, L.; UYSAL, M.; HOLLER, A.; GRIFFITH, R. Runtime Vertical Scaling of Virtualized Applications via Online Model Estimation. In: IEEE 8TH INTERNATIONAL CONFERENCE ON SELF-ADAPTIVE AND SELF-ORGANIZING SYSTEMS (SASO), 2014., 2014. **Proceedings...** IEEE, 2014.

SULEIMAN, B. Elasticity Economics of Cloud-Based Applications. In: IEEE NINTH INTERNATIONAL CONFERENCE ON SERVICES COMPUTING, 2012., 2012, Washington, DC, USA. **Proceedings...** IEEE Computer Society, 2012. p. 694–695. (SCC '12).

SURHONE, L.; TENNOE, M.; HENSSONOW, S. **Rightscale**. Saarbrücken, Germany: VDM Publishing, 2010.

TAN, L.; KOTHAPALLI, S.; CHEN, L.; HUSSAINI, O.; BISSIRI, R.; CHEN, Z. A survey of power and energy efficient techniques for high performance numerical linear algebra operations. **Parallel Computing**, Amsterdam, The Netherlands, The Netherlands, v. 40, n. 10, p. 559 – 573, 2014.

TANENBAUM, A. **Computer Networks**. 4th. ed. Upper Saddle River, New Jersey: Prentice Hall PTR, 2003. 912 p.

TRAN, N.-L.; SKHIRI, S.; ZIMáNYI, E. EQS: an elastic and scalable message queue for the cloud. In: IEEE THIRD INTERNATIONAL CONFERENCE ON CLOUD COMPUTING TECHNOLOGY AND SCIENCE, 2011., 2011, Washington, DC, USA. **Proceedings...** IEEE Computer Society, 2011. p. 391–398. (CLOUDCOM '11).

TRIANTAPHYLLOU, E. **Multi-Criteria Decision Making Methodologies**: a comparative study. Dordrecht: Springer, 2000. 320 pages p. (Applied Optimization, v. 44).

VALIANT, L. G. A Bridging Model for Parallel Computation. **Commun. ACM**, New York, NY, USA, v. 33, n. 8, p. 103–111, Aug. 1990.

VARELA, C.; AGHA, G. Programming Dynamically Reconfigurable Open Systems with SALSA. **SIGPLAN Not.**, New York, NY, USA, v. 36, n. 12, p. 20–34, Dec. 2001.

WARD, J. S.; BARKER, A. Self Managing Monitoring for Highly Elastic Large Scale Cloud Deployments. In: SIXTH INTERNATIONAL WORKSHOP ON DATA INTENSIVE DISTRIBUTED COMPUTING, 2014, New York, NY, USA. **Proceedings...** ACM, 2014. p. 3–10. (DIDC '14).

WEN, X.; GU, G.; LI, Q.; GAO, Y.; ZHANG, X. Comparison of open-source cloud management platforms: openstack and opennebula. In: FUZZY SYSTEMS AND KNOWLEDGE DISCOVERY (FSKD), 2012 9TH INTERNATIONAL CONFERENCE ON, 2012. **Anais...** IEEE, 2012. p. 2457 –2461.

WILKINSON, B.; ALLEN, C. **Parallel Programming**: techniques and applications using networked workstations and parallel computers. Upper Saddle River, NJ, USA: Pearson/Prentice Hall, 2005. (An Alan R. Apt book).

WOOD, T.; SHENOY, P.; VENKATARAMANI, A.; YOUSIF, M. Black-box and gray-box strategies for virtual machine migration. In: USENIX CONFERENCE ON NETWORKED SYSTEMS DESIGN & IMPLEMENTATION, 4., 2007, Berkeley, CA, USA. **Proceedings...** USENIX Association, 2007. p. 17–17. (NSDI'07).

YANG, J.; PANG, J.; QI, N.; QI, T. On-Demand Self-Adaptivity of Service Availability for Cloud Multi-tier Applications. In: CLUSTER, CLOUD AND GRID COMPUTING (CCGRID), 2015 15TH IEEE/ACM INTERNATIONAL SYMPOSIUM ON, 2015. **Anais...** IEEE, 2015. p. 1237–1240.

YU, L.; MORETTI, C.; THRASHER, A.; EMRICH, S.; JUDD, K.; THAIN, D. Harnessing parallelism in multicore clusters with the All-Pairs, Wavefront, and Makeflow abstractions. **Cluster Computing**, New York, NY, USA, v. 13, n. 3, p. 243–256, 2010.

ZHANG, X.; JEONG, S.; KUNJITHAPATHAM, A.; GIBBS, S. Towards an Elastic Application Model for Augmenting Computing Capabilities of Mobile Platforms. In: CAI, Y.; MAGEDANZ, T.; LI, M.; XIA, J.; GIANNELLI, C. (Ed.). **Mobile Wireless Middleware, Operating Systems, and Applications**. Heidelberg, Germany: Springer Berlin Heidelberg, 2010. p. 161–174. (Lecture Notes of the Institute for Computer Sciences, Social Informatics and Telecommunications Engineering, v. 48).

ZHANG, X.; SHAE, Z.-Y.; ZHENG, S.; JAMJOOM, H. Virtual machine migration in an over-committed cloud. In: NETWORK OPERATIONS AND MANAGEMENT SYMPOSIUM (NOMS), 2012 IEEE, 2012. **Anais...** IEEE, 2012. p. 196–203.

ZHANG, Y.; SUN, W.; INOGUCHI, Y. Predict task running time in grid environments based on {CPU} load predictions. **Future Generation Computer Systems**, Amsterdam, The Netherlands, The Netherlands, v. 24, n. 6, p. 489 – 497, 2008.