
**Programa Interdisciplinar de Pós-Graduação em**

# Computação Aplicada

**Mestrado Acadêmico**

Vinícius Meyer

# Pipel: Modelo de gerência da Elasticidade para Aplicações organizadas em Pipeline

São Leopoldo, 2016

Vinícius Meyer

# PIPEL: MODELO DE GERÊNCIA DA ELASTICIDADE PARA APLICAÇÕES ORGANIZADAS EM PIPELINE

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

M612p Meyer, Vinícius
Pipel : modelo de gerência da elasticidade para aplicações organizadas em pipeline/ por Vinícius Meyer – 2016.
101 f.: il. ; 30 cm.

Dissertação (mestrado) — Universidade do Vale do Rio dos Sinos, Programa de Pós-graduação em Computação Aplicada, São Leopoldo, RS, 2016.
"Orientação: Prof. Dr. Rodrigo da Rosa Righi."

1. Computação em nuvem. 2. Pipeline. 3. Elasticidade (Computação). I. Título.

CDU: 004.75

Catalogação na Publicação:
Bibliotecário Alessandro Dietrich - CRB 10/2338

Vinícius Meyer

Pipel: Modelo de gerência da Elasticidade para Aplicações organizadas em Pipeline

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 23 de agosto de 2016

BANCA EXAMINADORA

Prof. Dr. Rodrigo da Rosa Righi – UNISINOS

Prof. Dr. Jorge Luis Victória Barbosa – UNISINOS

Prof. Dr. Tiago Coelho Ferreto – PUCRS

Prof. Dr. Rodrigo da Rosa Righi

Visto e permitida a impressão
São Leopoldo,

Prof. Dr. Sandro José Rigo
Coordenador PPG em Computação Aplicada

Dedico este trabalho à memória do meu pai Werner Heinz Meyer e à minha mãe Ira Lied Meyer, pelo incentivo e amor incondicional.

## AGRADECIMENTOS

Agradeço: à todos os professores e colegas do PIPCA, pelo convívio e aprendizado;

Ao professor Dr. Rodrigo da Rosa Righi, pela orientação e ensinamentos, que tornaram possível a conclusão deste trabalho;

À Escola Estadual de Ensino Profissional de Estrela, por disponibilizar o ambiente para a realização dos experimentos deste estudo;

À toda a minha família, pelo apoio, paciência e incentivo;

À minha esposa Nádia, meu presente, por toda felicidade, compreensão, carinho e incentivo. Te amo.

## RESUMO

No ambiente da computação *workflows* tornam-se um padrão crescente para diversos experimentos científicos. *Workflows* científicos são compostos por várias aplicações estruturadas em um fluxo de atividades, onde o resultado de uma delas torna-se a entrada de outra. Uma aplicação *pipeline* é um tipo de *workflow* que recebe um conjunto de tarefas, as quais devem passar por todas as fases desta aplicação de forma sequencial, o que pode levar a um tempo de execução proibitivo. Tendo em vista este problema, aplicações *pipeline* podem se beneficiar da utilização de recursos distintos para cada um dos estágios, ou seja, executadas em plataformas distribuídas. Entretanto, dependências e necessidade especificas da computação distribuída surgem devido à interação entre os estágios de processamento e a grande quantidade de dados que devem ser processadas. O fluxo de entrada para aplicações que utilizam padrões *pipeline* pode ser intenso, inconstante ou irregular. De acordo com o comportamento do fluxo de tarefas, alguns estágios da aplicação podem ter seu desempenho prejudicado, atrasando os estágios subsequentes e por fim interferindo no desempenho da aplicação. Uma alternativa para resolver isto é alocar o máximo de recursos disponíveis (*over-provisioning*) em cada estágio da aplicação. Entretanto, esta técnica pode gerar um alto custo de infraestrutura, além da possibilidade que em alguns momentos os recursos fiquem ociosos. Sendo assim, a elasticidade em ambiente de nuvem computacional aparece como uma alternativa, explorando o conceito "pagar somente pelo que usar" (*pay-as-you-go*). Nesse contexto é proposto um modelo de elasticidade baseado na camada *PaaS* (*Plataform as a Service*) da nuvem, intitulado de Pipel. Este modelo permite que aplicações *pipeline* tirem vantagem do provisionamento dinâmico de recursos da infraestrutura de nuvem computacional. Pipel utiliza uma abordagem reativa, fazendo uso de *thresholds* para a tomada de decisões da elasticidade, baseados na carga de CPU das máquinas virtuais em cada estágio da aplicação. Cada estágio possui um balanceador de carga (chamado de controlador de estágio) e um determinado número de recursos em operação. O controlador do estágio recebe as tarefas que o estágio deve executar, as aloca em uma fila onde são distribuídas nas máquinas virtuais disponíveis em seu estágio. De acordo com regras estabelecidas Pipel realiza ações de elasticidade sobre o ambiente de nuvem. Para validar esta proposta foi desenvolvido um protótipo, o qual foi testado em dois cenários: (i) sem uso de elasticidade e (ii) com uso da elasticidade. Em cada cenário utilizou-se quatro cargas de processamento: (i) Crescente; (ii) Decrescente; (iii) Constante e (iv) Oscilante. Os resultados apresentam uma redução de 38% no tempo da execução da aplicação com o uso da elasticidade provida por Pipel.



**Palavras-chave:** Computação em Nuvem. *Pipeline*. Elasticidade.

## ABSTRACT

In the computing environment workflows has become a standard for many scientific experiments. Scientific workflows consist of several applications structured in an activity flow, where the output of one becomes the input of another. A pipeline application is a type of workflow that receives a set of tasks, which must pass through all stages of this application in a sequential manner, which can lead to a prohibitive execution time. Considering this problem, pipeline applications can benefit from the use of different resources for each stage, or performed in a distributed way. However, specific dependencies and distributed computing problems arise due to the interaction between the processing stages and the mass of data that must be processed. The input stream for applications that use pipeline standards can be intense, erratic or irregular. According to the task flow behavior some stages may have degraded performance, delaying subsequent stages and ultimately interfering in the application's performance. An alternative to solve this is to allocate the maximum available resources (over-provisioning) in each application stage. However, this technique can generate a high infrastructure costs and the possibility that some resources remain idle in certain moments. Thus, the elasticity in cloud computing environments appears as an alternative, exploring the pay-as-you-go concept. In this context, we propose an elastic model based on the PaaS layer (Platform as a Service) cloud, named Pipel. This model allows pipeline applications to take advantage of the dynamic resource provisioning capabilities of cloud computing infrastructure. Pipel uses a reactive approach, using thresholds for elasticity decisions based on the CPU load of virtual machines in each application stage. Each stage has a load balancer (called stage controller) and a number of operating resources. The stage controller receives the tasks that the stage should run, allocates in a queue and then distribute the tasks in virtual machines available at each stage. According to established rules Pipel performs elasticity actions on the cloud environment. To validate this proposal we developed a prototype that has been tested in two different scenarios: (i) without elasticity and (ii) with elasticity. In each scenario we used four different processing loads: (i) Increasing; (ii) Decreasing; (iii) Constant and (iv) Oscillating. The results showed a reduction of 38% of the application's execution time using the elasticity provided by Pipel.



**Keywords:** Cloud Computing. Pipeline. Elasticity.

## LISTA DE FIGURAS

## LISTA DE TABELAS

## LISTA DE SIGLAS

| | |
|---|---|
| AAR | Alocação Aleatória de Recursos |
| API | *Application Programming Interface* (Interface de Programação de Aplicações) |
| ARIMA | *Autoregressive Integrated Moving Average* (Modelo Auto-Regressivo Integrado de Média Móvel) |
| ASR | Alocação Sequencial de Recursos |
| CAPES | Coordenação de Aperfeiçoamento de Pessoal de Nível Superior |
| CE | Controlador de Estágio |
| CPU | *Central Processing Unit* (Unidade Central de Processamento) |
| DAG | *Directed Acyclic Graph* (Gráfico Acíclico Dirigido) |
| FIFO | *First In, First Out* (Primeiro que entra, primeiro que sai) |
| GE | Gerenciador de Elasticidade |
| HPC | *High Performance Computing* (Computação de Alto Desempenho) |
| HTTP | *Hypertext Transfer Protocol* (Protocolo de Transferência de Hipertexto) |
| IaaS | *Infrastructure as a Service* (Infraestrutura como Serviço) |
| ID | *Identity Document* (Documento de Identidade) |
| LDAP | *Lightweight Directory Access Protocol* (Protocolo de Acesso Direto) |
| NIST | *National Institute of Standards and Technology* (Instituto Nacional de Padrões e Tecnologia) |
| MPI | *Message Passing Interface* |
| MA | *Moving Average* (Média Móvel) |
| MEC | Ministério da Educação |
| MPI | *Message Passing Interface* |
| MPMD | *Multiple Program Multiple Data* (Múltiplo Programa Múltiplos Dados) |
| MV | Máquina Virtual |
| MVs | Máquinas Virtuais |
| NFS | Network File System (Sistema de Arquivos de Rede) |
| PaaS | *Platform as a Service* (Plataforma como Serviço) |
| PAD | Processamento de Alto Desempenho |
| QoS | *Quality of Service* (Qualidade de Serviço) |
| SaaS | *Software as a Service* (Software como Serviço) |
| SES | *Simple Exponential Smoothing* (Suavização Exponencial Simples) |
| SLA | *Service Level Agreement* (Acordo de Nível de Serviço) |
| SOA | *Service-Oriented Architecture* (Arquitetura Orientada a Serviços) |

| | |
|---|---|
| SSH | *Secure Shell* |
| TET | Tempo de Execução Total |
| TI | Tecnologia da Informação |

**SUMÁRIO**

# 1 INTRODUÇÃO

Na área da computação, *workflows* tornam-se um padrão para a modelagem de experimentos científicos(ZHANG et al., 2016; WU et al., 2016; HENDRIX et al., 2016). *Workflows* científicos geralmente são compostos por várias aplicações estruturadas em um fluxo de atividades que acessam recursos compartilhados como bancos de dados ou repositórios distribuídos (DEELMAN et al., 2009; MATTOSO et al., 2010). Também são utilizados para processar massas de dados e para realizar experimentos de larga escala. Existe um tipo de aplicação *workflow*, conhecido como *pipeline* linear, onde cada conjunto de dados deve passar por todas as fases da aplicação de uma forma sequencial. Podem ser citados dois exemplos de aplicação: (i) processamento de imagem, onde os fluxos de imagens (conjuntos de dados) entram no *pipeline* e devem passar por várias fases, tais como filtros, codificadores, e assim por diante (RAMANATH et al., 2005; F. GUIRADO A. RIPOLL, 2006; HARTLEY et al., 2009); e (ii) processamento de vídeo (SCHNEIDER et al., 2009). Estes modelos de aplicações baseados no formato *pipeline* podem ser demorados e podem necessitar de recursos intensivos que se beneficiam da execução em plataformas distribuídas. Com isso, dependências e necessidade de alto processamento surgem devido à interação entre as atividades de fluxos de trabalho e o volume de informações que devem ser executadas (RODRIGUEZ; BUYYA, 2015).

Neste cenário, os cientistas que utilizam aplicações no formato *pipeline* precisam executar fluxos de trabalho em paralelo. Muitos destes fluxos científicos são compostos por atividades categorizadas como de computação intensiva. Para reduzir o tempo de execução total (TET), este tipo de aplicação geralmente se beneficia de técnicas de computação paralela em ambientes HPC (*High Performance Computing*, Computação de Alto Desempenho) bem conhecidos, tais como *clusters* e *grids* (MATTOSO et al., 2010; COSTA et al., 2012).

Além dos *clusters* e *grids* tradicionais, outras soluções estão sendo utilizadas para atender a essa demanda por recursos de computação mais poderosos e escaláveis. Computação em nuvem tem emergido como uma alternativa viável para o ambiente de muitos tipos de aplicações HPC, uma vez que oferecem elasticidade de *hardware* e *software* sob demanda. Com o uso das nuvens computacionais, os cientistas não precisam necessariamente adquirir equipamentos caros, uma vez que é possível instanciar o poder de computação necessário através da criação de um ambiente virtual, que é composto por várias máquinas virtuais independentes (VAQUERO et al., 2009; MATTOSO et al., 2010; COSTA et al., 2012).

Uma característica fundamental da computação em nuvem é a elasticidade, ajustando automaticamente os recursos do sistema para diversas cargas de trabalho de uma determinado aplicação. Abordagens reativas e horizontais representam uma estratégia para oferecer esta capacidade, em que as regras previamente definidas, ditam as ações a serem tomadas de acordo com os limites superiores e inferiores para instanciar ou consolidar nós de computação e/ou máquinas virtuais. Embora a elasticidade pode ser benéfica para muitas aplicações de alto desempenho, também impõe desafios significativos em seu desenvolvimento (ROSA RIGHI et al.,

2015).

## 1.1 Motivação

Devido à complexidade dos processos científicos, aplicações científicas organizadas em *pipeline* têm se tornado cada vez mais dependentes do uso da computação/dados intensivos (LIN; LU, 2011; FRINCU; GENAUD; GOSSA, 2013). Os fluxos de trabalhos, em um conjunto, normalmente têm uma estrutura semelhante, mas diferem nos seus dados de entrada, número de tarefas e tamanhos de tarefas individuais (MALAWSKI et al., 2012).

Processamento de *workflow* é um paradigma de programação que naturalmente utiliza paralelismo de tarefas. Aplicações que utilizam dados de maneira independentes uns do outros, e são alimentados com correntes contínuas dos mesmos podem executá-los em paralelo. A única comunicação entre os operadores acontece através dos fluxos que os conectam. Quando os operadores estão ligados em cadeias, eles expõem paralelismo *pipeline*. Quando os mesmos fluxos são enviados para múltiplos operadores que executam tarefas distintas, eles expõem paralelismo de tarefa. Ser capaz de explorar facilmente paralelismo de tarefas e *pipeline* é comum em domínios como as telecomunicações, processamento de imagens, negociações financeiras, análise de dados *web* em grande escala e análises de mídias sociais. Esses domínios exigem alto processamento e aplicações de baixa latência que podem ser dimensionados com o número de núcleos em um nó computacional para cada máquina virtual (SCHNEIDER et al., 2015; SHUANG et al., 2015). O fluxo de entrada para aplicações que utilizam padrões *pipeline* pode ser intenso, inconstante ou irregular. Uma alternativa para resolver isto é alocar o máximo de recursos disponíveis (*over-provisioning*) em cada estágio da aplicação. Porém, isso gera um alto custo de infraestrutura, considerando ainda o fato de que em alguns momentos diversos recursos podem ficar ociosos (pagar e não utilizar). Sendo assim, a elasticidade aparece como solução para esta situação, explorando o conceito "pagar somente pelo que usa".

Ambientes de computação distribuídos, como nuvens computacionais, se tornam uma opção para executar fluxos de trabalhos científicos. Este tipo de ambiente pode fornecer diversos recursos que se diferenciam dos ambientes tradicionais (*grids e clusters*): (i) recursos de computação em nuvem são oferecidos como serviços que fornecem uma interface padronizada para o acesso através da rede; (ii) O tipo e o número de recursos de computação atribuídos a um fluxo de trabalho são determinados por requisições de serviços; (iii) O número de recursos atribuídos a um fluxo de trabalho pode ser alterado dinamicamente em tempo de execução, para que os recursos de computação possam ser elasticamente escalados sob demanda; (iv) Nem todos os recursos de computação solicitados precisam ser atribuídos no início de uma execução do fluxo de trabalho. Os recursos podem ser atribuídos apenas quando eles são necessários (LIN; LU, 2011).

Um dos problemas que envolvem experimentos com nuvens computacionais é o mapeamento de tarefas aos recursos. Para conseguir resolver estes problemas se torna necessário de-

senvolver uma programação que atinja três níveis: (i) localizar as máquinas físicas adequadas para um conjunto de máquinas virtuais; (ii) determinar o melhor esquema de provisionamento para as máquinas virtuais; e (iii) agendar as tarefas nas máquinas virtuais. Dependendo do nível de acesso que um fornecedor de serviços libera sobre a infraestrutura para o seu cliente, o desempenho da aplicação pode ser afetado (FRINCU; GENAUD; GOSSA, 2013). Neste ambiente, o número de recursos atribuídos a um fluxo de trabalho pode ser elasticamente escalado por solicitações de serviço. O número de recursos não pode ser automaticamente determinado com a demanda do tamanho de um fluxo de trabalho. Os recursos ideais atribuídos a um *workflow* geralmente não são liberados até que ele complete uma execução. Como resultado, os recursos atribuídos a um fluxo de trabalho podem algumas vezes tornar-se insuficientes para a execução do mesmo, o que leva a uma execução de longa duração. Também, muitos recursos podem tornar-se inativos por um longo tempo durante sua execução, o que leva a um desperdício de recursos e orçamento (LIN; LU, 2011).

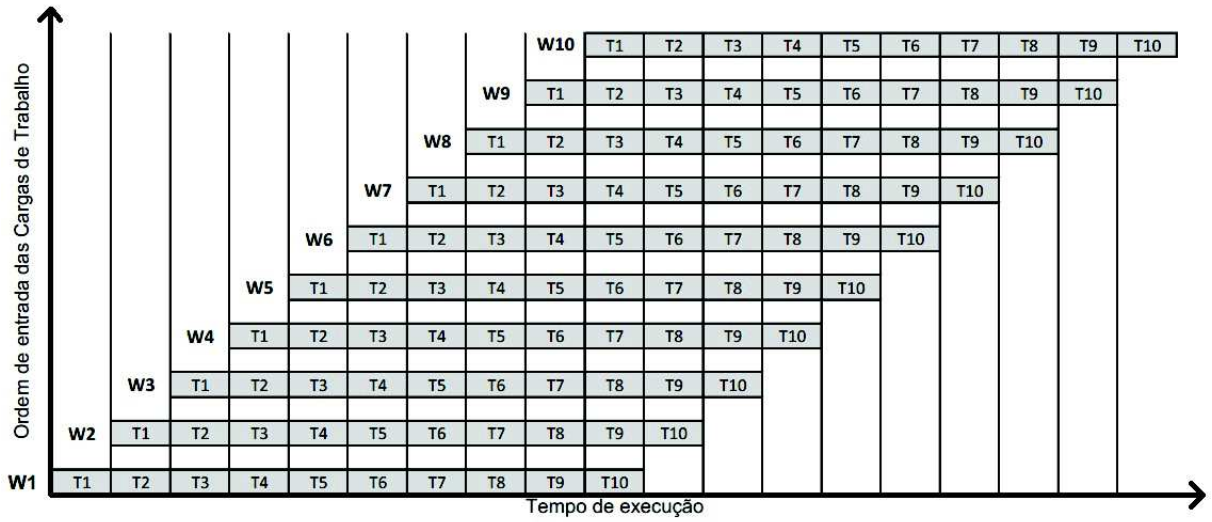
Um problema tradicional em aplicações organizadas em *pipeline* ocorre quando um estágio se torna demorado e os demais processam suas tarefas de forma mais rápida. Em função deste atraso, em determinado estágio, o desempenho da aplicação, como um todo, é prejudicado. Neste cenário, torna-se interessante a utilização da estratégia do *pipeline* superescalar, que tem como objetivo criar várias unidades computacionais por estágio. Dessa maneira, os estágios adquirem tempos de execução semelhantes e melhoram a vazão do fluxo de tarefas em toda a aplicação. Neste caso, vazão refere-se ao número de tarefas concluídas por unidade de tempo (estágio da aplicação *pipeline*).

## 1.2 Questão de Pesquisa

A questão de pesquisa que o modelo proposto busca responder é: *O quanto pode melhorar o desempenho de aplicações organizadas em pipeline utilizando um modelo de elasticidade automática e reativa em ambiente de nuvem computacional?*

Aplicações *pipeline* paralelas tradicionais são denominados como sistemas baseados em filas, orientados a processamento em lote. Este modelo de aplicação pode se tornar muito interessante para problemas que demandam alto poder de processamento (EVANGELINOS; HILL, 2008). O uso de MPI (*Message Passing Interface*) é um dos modelos de programação mais usados para processamento de alto desempenho. Para serviços distribuídos que exigem alto desempenho, acesso à rede em nível de usuário, MPI promete ser uma alternativa viável (ZOUNMEVO et al., 2013). A computação em nuvem oferece uma dimensão atraente para computação intensiva, em que os recursos virtualizados podem ser utilizados, em um formulário personalizado para atingir um cenário específico, no momento e na forma que eles são desejados. Este conceito se aplica em computação interativa sensível (EVANGELINOS; HILL, 2008). Considerando o fator desempenho, aplicações *pipeline* são sensíveis quanto a gargalos no processamento e sincronismo das tarefas. A ideia é construir uma plataforma que utilize a

**Figura 1:** Ordem de entrada das Cargas de Trabalho *versus* Tempo de execução




Fonte: Desenvolvido pelo autor.

elasticidade sem precisar alterar a aplicação. Para isso, deve-se explorar a aplicação, a infraestrutura de nuvem e o mecanismo de elasticidade. Assim sendo, elasticidade automática se refere a capacidade de uma nuvem alocar e/ou consolidar recursos em tempo de execução da aplicação, sem que haja a necessidade do usuário intervir. Além disso, o conceito de elasticidade reativa refere-se à regras previamente definidas de limiares de processamento. Caso esses limites sejam ultrapassados, o modelo toma ações sobre os recursos da infraestrutura de nuvem computacional. E ainda, a aplicação que será monitorada pelo modelo é organizada em formato *pipeline*. Esta aplicação é um tipo de *workflow* que recebe uma lista de tarefas, as quais devem passar por todas as etapas desta aplicação de maneira sequencial.

## 1.3 Objetivos

Aplicações *pipeline* são aplicações que tem um fluxo de execução sequencial e dividido por estágios, aonde o início de cada estágio depende da finalização do estágio anterior. A figura 1 ilustra um exemplo da execução de dez *workflows* (W1, W2, W3, ..., W10), cada um com dez tarefas (T1, T2, T3, ..., T10). O segundo *workflow* só inicia quando o primeiro *workflow* passa para o estágio 2 e assim sucessivamente até que todos os *workflows* passem pelo sistema e completem sua execuções. É possível haver apenas uma execução de cada estágio por intervalo de tempo, de acordo com a ordem de entrada dos *workflows*. Esse comportamento de aplicação é caracterizado como aplicação *pipeline*.

Com o intuito de proporcionar elasticidade em ambiente de Nuvem Computacional para aplicações *pipeline* de maneira mais eficiente e transparente possível, este trabalho propõe um modelo de elasticidade intitulado Pipel.

Pipel é um sistema que monitora os estágios da aplicação de forma individual e a aplicação

como um todo, proporcionando ações de elasticidade horizontal e decidindo qual tratamento será dado aos recursos da nuvem. Desta maneira é possível aumentar ou diminuir o número de instâncias de máquinas virtuais ou ainda reutilizar as mesmas para outros estágios da aplicação, caso necessário. Também, com estas ações elásticas em tempo de execução, o objetivo é manter o mesmo intervalo de tempo para todos os estágio da aplicação, assegurando a característica de execução no formato *pipeline*.

O objetivo geral deste trabalho é: Criar um modelo de elasticidade automática e transparente para aplicações organizadas em *pipeline* em ambientes de Nuvem Computacional sem a necessidade de intervenção do usuário. Assim sendo, elasticidade automática se refere a capacidade de uma nuvem adicionar/remover/remanejar a quantidade de recursos durante a execução da aplicação sem a necessidade de intervenção do usuário. Além disso, o conceito de transparência refere-se a execução das ações de elasticidade ocorrerem sem que a execução da aplicação seja afetada e a mesma não precise prever estas alterações. Finalmente, o foco principal do modelo é executar aplicações *pipeline* com utilização de elasticidade.

Para alcançar o objetivo geral, foram definidos os seguintes objetivos específicos:

- Realizar um levantamento e análise de conteúdos com assuntos referentes aos modelos de elasticidades atuais (estado da arte);
- Pesquisar sobre trabalhos relacionados que mais se aproximam do assunto do modelo proposto;
- Criar um protótipo para o modelo proposto;
- Efetuar testes com diversos padrões de carga para suas execuções.

## 1.4 Organização do Texto

Este trabalho está organizado em sete capítulos. O Capítulo 2 apresenta os conceitos relacionados aos assuntos pesquisados, introduzindo tecnologias e conceitos que são utilizados para o entendimento e desenvolvimento do modelo proposto. Em seguida, no Capítulo 3 são discutidos os trabalhos relacionados que mais se aproximam do assunto abordado neste estudo: elasticidade em aplicações para diferentes cargas de trabalho no formato *pipeline*. No Capítulo 4 são apresentadas as decisões do modelo, sua arquitetura, o modelo de elasticidade utilizado e os métodos de avaliação. O Capítulo 5 explica como foram realizados os testes, as diferentes cargas de processamento e cenários para execução dos experimentos. No Capítulo 6 são apresentados os resultados obtidos com os experimentos avaliados. Por fim, o Capítulo 7 apresenta as conclusões obtidas com o trabalho.

# 2 FUNDAMENTAÇÃO TEÓRICA

Este capítulo apresenta os principais conceitos relacionados ao estudo desenvolvido. A Seção 2.1 aborda os conceitos de Computação em Nuvem propriamente dita e suas principais características. A Seção 2.2 explica o que são Aplicações Organizadas em *Pipeline* e algumas de suas derivações. Por fim, são apresentados também os conceitos relacionados à Computação Paralela na Seção 2.3.

## 2.1 Computação em Nuvem

O conceito de Computação em Nuvem é definido como um grande conjunto de recursos virtualizados, facilmente utilizáveis e acessíveis (como *hardware*, plataformas de desenvolvimento e serviços). Esses recursos podem ser reconfigurados dinamicamente para se ajustar a uma determinada carga variável, permitindo também uma utilização de recursos otimizada. Este conjunto de recursos é normalmente explorado por um modelo de "pague pelo uso" em que garantias são oferecidas pelo fornecedor de infraestrutura por meio de SLAs (*Service Level Agreement*, Acordo de Nível de Serviço) customizados (VAQUERO et al., 2009). O NIST (*National Institute of Standards and Technology*, Instituto Nacional de Padrões e Tecnologia), define computação em nuvem como um modelo para habilitar o acesso por rede, conveniente e sob demanda a um conjunto compartilhado de recursos de computação (como redes, servidores, armazenamento, aplicações e serviços) que possam ser rapidamente provisionados e liberados com o mínimo de esforço de gerenciamento ou interação com o provedor de serviços. Ainda segundo o NIST, cinco são as características essenciais de uma Nuvem Computacional (MELL; GRANCE, 2011):

- Serviço automático sob demanda - Um consumidor pode provisionar recursos de computação (tempo de servidor e armazenamento em rede) automaticamente conforme necessário, sem a necessidade de interação humana com cada provedor de serviço;

- Amplo acesso à rede - Os recursos estão disponíveis através da rede e são acessados através de mecanismos que promovem o uso por dispositivos com ampla diversidade de plataformas como por exemplo: celulares, *tablets*, *laptops* e estações de trabalho;

- Conjunto de recursos - Os recursos de computação do provedor são agrupados para servir múltiplos consumidores que utilizam um modelo multi-hospedeiro, com diferentes recursos físicos e virtuais atribuídos e realocados dinamicamente de acordo com a demanda dos consumidores. Há um senso de independência local em que o cliente geralmente não tem controle ou conhecimento sobre a localização exata dos recursos disponibilizados, mas pode ser capaz de especificar o local em um nível mais alto de abstração (por exemplo, país, estado, ou *datacenter*). Exemplos de recursos incluem armazenamento, processamento, memória e largura de banda de rede;

- Elasticidade rápida - Os recursos podem ser elasticamente provisionados e liberados, em alguns casos, automaticamente, para escalar rapidamente para corresponder à demanda. Para o consumidor, os recursos disponíveis para realizar o provisionamento muitas vezes parecem ser ilimitados e podem ser consumidos em qualquer quantidade a qualquer momento;

- Serviço mensurado - Os sistemas de nuvem controlam e otimizam o uso dos recursos, aproveitando a capacidade de medição em nível de abstração adequado. O uso de recursos pode ser monitorado, controlado e relatado, proporcionando transparência para o provedor e consumidor do serviço utilizado.

### 2.1.1 Modelos de serviço

Existem diferentes categorias de serviços de Computação em Nuvem como infraestrutura, plataforma e aplicação etc. Estes serviços são disponibilizados e consumidos em tempo real sobre a internet (RIMAL; CHOI; LUMB, 2009). Os serviços prestados pela computação em nuvem podem ser divididos em três principais categorias:

- *IaaS* (*Infrastructure as a Service*, Infraestrutura como Serviço) - Infraestrutura como Serviço é uma forma de hospedagem que inclui acesso à rede, serviços de roteamento e armazenamento. O provedor de IaaS geralmente fornece o hardware e os serviços administrativos necessários para armazenar aplicativos junto a uma plataforma para a execução dos mesmos (CHAVAN et al., 2013). Os recursos da nuvem são fornecidos para os consumidores sob demanda na forma de máquinas virtuais (AKHANI; CHUADHARY; SOMANI, 2011). A partir destes recursos o consumidor é capaz de implantar e executar softwares arbitrários, que podem incluir sistemas operacionais e/ou aplicativos. O consumidor não gerencia ou controla a infraestrutura da nuvem subjacente, mas tem controle sobre os sistemas operacionais, armazenamento e aplicativos implementados (MELL; GRANCE, 2011). O provedor da nuvem possui os equipamentos e é responsável pela execução e manutenção dos mesmos. IaaS pode ser adquirido na forma de contrato ou pelo modelo 'pague pelo que usar' (CHAVAN et al., 2013);

- *PaaS* (*Platform as a Service*, Plataforma como Serviço) - Plataforma como Serviço fornece aos desenvolvedores de software uma plataforma que compreende todo o ciclo de vida de desenvolvimento, testes, implantação e hospedagem de aplicações web sofisticadas. Ele fornece de maneira mais fácil um ambiente para desenvolver aplicações de negócios e vários serviços através da internet (RIMAL; CHOI; LUMB, 2010). O consumidor deste modelo tem a capacidade de criar e executar aplicações para infraestrutura de nuvem. Estas aplicações podem ser criadas utilizando linguagens de programação, bibliotecas, serviços e ferramentas suportadas pelo fornecedor deste serviço (MELL; GRANCE, 2011). A criação e manutenção de uma infraestrutura é o trabalho mais oneroso para os

sistemas. *PaaS* foi criado para resolver exatamente esse problema. Exemplos importantes são o Google AppEngine, o Microsoft Azure, Heroku.com, etc. Comparado com o desenvolvimento de aplicações convencionais, esta estratégia pode reduzir o tempo de desenvolvimento, oferecendo centenas de ferramentas e serviços prontamente disponíveis (RIMAL; CHOI; LUMB, 2010);

- *SaaS* (*Software as a Service*, Software como Serviço) - Software como Serviço é a capacidade fornecida ao usuário acessar aplicações que executam sobre infraestrutura de nuvem. As aplicações podem ser acessadas a partir de diversos dispositivos, como um navegador web (por exemplo, e-mail baseado na web), ou a interface de um programa (MELL; GRANCE, 2011). Outra característica importante é que uma aplicação pode ser acessada por diversos usuários simultaneamente (RAJARAMAN, 2014). 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 (MELL; GRANCE, 2011).

### 2.1.2 Tipos de nuvens

Para prover uma solução de computação em nuvem segura, é importante decidir qual o tipo de nuvem que deve ser implementado. Existem quatro tipos de modelos de implantação em nuvem disponíveis (JADEJA; MODI, 2012):

- Nuvem Privada - As nuvens privadas servem para atender um centro de dados de uma única organização. As principais vantagens são: (i) facilidade de gerenciamento, segurança, manutenção e atualizações e (ii) também fornecimento de maior controle sobre a implantação e utilização do ambiente (JADEJA; MODI, 2012). A infraestrutura pode ser de propriedade da organização ou de terceiros. Entretanto é de uso exclusivo da organização que adquiri ou contrata este serviço, não havendo o compartilhamento de recursos com outras empresas ou entidades (MELL; GRANCE, 2011). A utilização da nuvem privada pode ser muito mais segura do que o da nuvem pública pois somente a organização e as partes interessadas tem acesso para manipular e operar a própria nuvem (RAMGOVIND; ELOFF; SMITH, 2010).

- Nuvem Pública - As nuvens públicas permitem o acesso de usuários na nuvem através de interfaces de navegadores web (browsers). Os usuários pagam apenas pelo tempo que utilizam o serviço. Isso pode ser comparado ao sistema elétrico que recebemos em nossas casas. Pagamos apenas pela quantidade que usamos. O mesmo conceito é aplicado aqui. Isto ajuda a reduzir os custos de operação de TI (RAMGOVIND; ELOFF; SMITH, 2010). A infraestrutura de nuvem é fornecida para uso aberto pelo público em geral. Pode ser gerenciada e operada por uma empresa, uma comunidade científica ou organização do governo, ou ainda alguma combinação entre eles (MELL; GRANCE, 2011). As nuvens

públicas são menos seguras em comparação com outros modelos de nuvem em função de que todos os aplicativos e dados na nuvem pública são mais suscetíveis a ataques maliciosos. A solução para isto pode ser a implantação de segurança através da validação de ambos os lados: pelo fornecedor da nuvem e pelo cliente da mesma. Além disso, tanto as partes devem identificar as suas responsabilidades dentro de seus limites de operação (JADEJA; MODI, 2012).

- Nuvem Híbrida - A infraestrutura de nuvem híbrida é uma composição de duas ou mais infraestruturas distintas de nuvem (privada, comunitária ou pública) que permanecem como entidades únicas, mas estão unidas por tecnologias padronizadas ou proprietárias que permitem que os dados e aplicações sejam remanejados (por exemplo, estouro de recursos para balanceamento de carga entre nuvens) (MELL; GRANCE, 2011). Este serviço permite uma melhor organização para atender necessidades específicas. No caso da aplicação precisar de recursos de computação intensiva, pode escolher qual o tipo que melhor se adéqua a sua necessidade (JADEJA; MODI, 2012).

- Nuvem Comunitária - É considerada nuvem comunitária quando diversas organizações (exemplo, comunidades científicas) compartilham uma infraestrutura de nuvem em conjunto de forma a contribuir e atender suas necessidades computacionais. Essa infraestrutura pode ser hospedada em um provedor de terceiro ou dentro de uma das organizações da comunidade (JADEJA; MODI, 2012; MELL; GRANCE, 2011).

### 2.1.3 Elasticidade

A Elasticidade é uma das principais características da computação em nuvem. É definida pelo grau em que um sistema é capaz de se adaptar à carga de trabalho através de alterações de provisionamento e desprovisionamento de recursos de forma autônoma. Desta maneira, em determinado ponto da linha de tempo da aplicação, os recursos disponíveis correspondem a demanda da carga de trabalho (HERBST et al., 2013). Elasticidade é essencial para o conceito de computação em nuvem tornando-se amplamente utilizado para várias finalidades. Do ponto de vista do fornecedor de nuvem, a elasticidade garante uma melhor utilização de recursos de computação, proporcionando economias de escala e permitindo que múltiplos usuários possam utilizar o sistema de maneira simultânea. Da perspectiva do usuário, a elasticidade tem sido usado principalmente para evitar a disposição inadequada de recursos e, consequentemente, a degradação do desempenho do sistema. A definição de Elasticidade pode ser entendida pelas métricas de modelos e métodos (GALANTE; BONA, 2012).

#### 2.1.3.1 Modelos

Existem dois modelos que podem ser assumidos na elasticidade de recursos: o modelo manual e o automático. No modelo manual o usuário é responsável por monitorar seu ambiente e aplicações e realizar todas as ações de reescalonamento de recursos. O provedor de nuvem deve fornecer através de uma interface (geralmente uma API, *Application Programming Interface*, Interface de Programação de Aplicações) com a qual o utilizador interage com o sistema. No modelo automático, o controle e as ações são tomadas pelo sistema e/ou pela aplicação. Em conformidade com as regras e configurações definidas pelo usuário, ou especificado no Acordo de Nível de Serviço (SLA - *Service Level Agreement*). O sistema de controle automático utiliza os serviços de monitoramento para coletar informações como a carga da CPU, memória e tráfego de rede para tomar decisões sobre os recursos disponíveis. De acordo com a técnica utilizada para acionar as ações de elasticidade, é possível diferenciar os modelos automáticas em: reativa e preditiva. As soluções reativas são baseadas em condições/ações de uma regra que, quando satisfeita ativa gatilhos de tomada de decisões sobre a nuvem subjacente. Cada condição considera um evento ou uma métrica do sistema que é comparado com um limiar. A informação sobre os valores de métrica e eventos é fornecida pelo sistema de monitorização da infraestrutura, ou pela aplicação. Por sua vez, a abordagem preditiva utiliza heurística e técnicas matemáticas/analíticas para antecipar o comportamento de carga do sistema e, com base nestes resultados, tomar decisões de quando e como dimensionar os recursos (GALANTE; BONA, 2012).

#### 2.1.3.2 Métodos

Existem três métodos para implementação de soluções de elasticidade: a replicação, a migração e o redimensionamento. A replicação, também conhecido como elasticidade horizontal, consiste em adicionar ou remover instâncias de máquinas virtuais do ambiente do usuário. A replicação é atualmente o método mais utilizado para proporcionar elasticidade, sendo usado na maioria dos fornecedores de nuvens públicas. Para complementar a elasticidade, sistemas em nuvem oferecem um serviço adicional chamado de balanceamento de carga, com o princípio de balancear a carga em várias réplicas para melhorar o serviço. O método de redimensionamento, conhecido também como elasticidade vertical, compreende a remoção e(ou) adição dos recursos de memória, processamento e armazenamento de uma instância virtual em execução. Redimensionamento é o método padrão para aplicações que adotam a adição de dados ou estruturas de controle, *threads* por exemplo, a fim de explorar novos recursos disponíveis nas máquinas virtuais. A migração de máquina virtual é a transferência de uma máquina virtual que está sendo executada em um servidor físico (*host*) para outro. A elasticidade pode ser implementada pela migração de uma máquina virtual para uma máquina física que melhor se adapta à carga aplicação ou, através da consolidação de uma ou um conjunto de máquinas vir-

tuais em um *host*. Geralmente, a migração é usada para simular o comportamento obtido com o redimensionamento nas nuvens (GALANTE; BONA, 2012).

### 2.1.4 Middlewares

No contexto de sistemas de computação distribuída, *Middleware*, pode ser descrito como um conjunto de intermediários para os componentes de um sistema de computação distribuída. Este conceito tem sido amplamente utilizado durante a evolução da Arquitetura Orientada a Serviços (*Service-Oriented Architecture*, SOA), onde os serviços em questão eram de fato fornecidos por sistemas de *middlewares*. Em geral são usados para abstrair as diferenças entre sistemas heterogêneos e expor uma interface uniforme (BERNSTEIN, 1996). Para Nuvem Computacional pode-se citar alguns *middlewares* existentes, entre eles: OpenNebula, OpenStack e Amazon EC2 (WANG; LI; WANG, 2016).

#### 2.1.4.1 OpenNebula

OpenNebula[1] é um projeto de pesquisa que começou como uma ferramenta de gestão para a manipulação e reconfiguração de máquinas virtuais em centros de dados, desenvolvido na Universidad Complutense de Madrid. O principal objetivo era enfrentar os desafios que os modelos de negócios ofereciam em condições do mundo real. OpenNebula é um projeto *open source* (código aberto) que visa proporcionar os recursos para o gerenciamento e monitoramento de máquinas virtuais e de computação em nuvem. Além de dar suporte a construção em nuvem privada, o OpenNebula apoia a ideia de nuvens híbridas. Uma nuvem híbrida permite combinar uma infraestrutura em nuvem privada com uma infraestrutura em nuvem pública (como a Amazon[2]) para garantir graus ainda maiores de escalabilidade. O OpenNebula suporta diversos *hypervisors*, entre eles: Xen, KVM/Linux e VMware. (MILOJICIC; LLORENTE; MONTERO, 2011)

OpenNebula é escrito em linguagem *C++*, *Ruby* e *Shell*. A rede é gerenciada manualmente pelo administrador. O gerenciamento das cópias padrões das máquinas virtuais são acessados via SSH (*Secure Shell*). OpenNebula suporta autenticação via LDAP (*Lightweight Directory Access Protocol*) (SEFRAOUI; AISSAOUI; ELEULDJ, 2012; CHILIPIREA et al., 2016).

#### 2.1.4.2 OpenStack

OpenStack[3] é uma plataforma inicialmente desenvolvida com ajuda da NASA[4], dedicada às infraestruturas maciças. As principais características são: (i) Escalabilidade, suporta até 1

[1] http://opennebula.org/
[2] https://www.amazon.com/
[3] https://www.openstack.org/
[4] https://www.nasa.gov/

milhão de máquinas físicas, até 60 milhões de máquinas virtuais e bilhões de objetos armazenados; (ii) Compatibilidade e Flexibilidade, suporta a maioria das soluções de virtualização do mercado: ESX, Hyper-V, KVM, LXC, QEMU, UML, Xen e XenServer; (iii) Código Aberto, todo o código pode ser modificado e adaptado conforme necessidade. OpenStack é escrito em linguagem *python* e atualmente implementa duas APIs de controle, a API EC2 e Rackspace. Ele usa *drivers* diferentes para fazer a interface com um máximo número de *hypervisors* (Xen, KVM, HyperV, Qemu). Este projeto é dedicado a fornecer a oportunidade de construir uma arquitetura de hospedagem e escalabilidade massiva e totalmente *open source*, enquanto supera as limitações do uso de tecnologias proprietárias. (SEFRAOUI; AISSAOUI; ELEULDJ, 2012).

#### 2.1.4.3 Amazon AWS

Amazon AWS é uma coleção de serviços de computação em nuvem ou serviços *web*, que formam uma plataforma de computação na nuvem oferecida pela Amazon[5]. Os serviços são oferecidos por diversas regiões distribuídas no mundo. Os serviços mais conhecidos são o Amazon Elastic Compute Cloud[6] e o Amazon S3[7] (WANG; NG, 2010). A Amazon AWS usa a técnica de virtualização Xen para gerenciar servidores físicos. Podem haver várias máquinas virtuais Xen rodando em um servidor físico. Para aplicações que requerem maior capacidade de computação, a Amazon AWS fornece configurações de máquinas virtuais mais robustas. A Amazon organiza a infraestrutura em diferentes regiões e zonas de disponibilidade. Cada região é completamente independente e contém várias zonas de disponibilidade que são usadas para melhorar a tolerância a falhas no interior da região (WANG; LI; WANG, 2016).

## 2.2 Aplicações Organizadas em *Pipeline*

Diferente da abordagem em que uma aplicação é considerada um único elemento que executa todas as tarefas, a aplicação organizada em *pipeline* consiste em uma coleção de vários componentes que interagem entre si e que precisam ser executados em uma determinada ordem para o sucesso da aplicação como um todo. Estes componentes têm dependências de controle e de dados específicos entre eles. Aplicações *pipeline* são aplicações multiestágios em que as tarefas precisam ser processadas uma por uma com restrições de precedência (MAO; HUMPHREY, 2011). Na maioria dos casos, o fluxo de trabalho da aplicação pode ser representado como um gráfico acíclico dirigido (*Directed Acyclic Graph*, DAG), onde cada nó no DAG representa um componente da aplicação e as arestas denotam dependências de controle e/ou dados (MANDAL et al., 2005).

O conceito de *pipeline* representa uma estrutura de processamento que opera com dados de entrada para produzir dados de saída. Vários desses trabalhos de processamento de dados

[5]https://aws.amazon.com/pt/
[6]https://aws.amazon.com/pt/ec2
[7]https://aws.amazon.com/pt/s3

**Figura 2:** *Workflow* Sequencial


Fonte: Desenvolvido pelo autor.

podem ser combinados sequencialmente para produzir a estrutura *pipeline*. Esta estrutura pode ser caracterizada com vários fluxos de trabalho e, como consequência, todos estes fluxos se juntam formando um fluxo único. Neste caso, cada estágio de trabalho de entrada opera após a saída da etapa anterior e a saída produzida é alimentada como entrada para a próxima fase no *pipeline* (BUYYA, 1999; BHARATHI et al., 2008). Aplicações modeladas no formato *pipeline* são caracterizadas como um tipo de *workflow* científico. Estas são muito utilizadas para executar tarefas de processamento de alto desempenho (PAD). Sistemas de *workflow* tem uma série de vantagens para a construção e gestão de tarefas computacionais. Eles fornecem um modelo de programação simples pelo qual uma sequência de tarefas é composta por ligar as saídas de uma tarefa para as entradas de outra. Além disso, os sistemas de fluxo de trabalho muitas vezes fornecem interfaces de programação visual intuitiva, que os tornam mais adequado para usuários que não têm conhecimentos de programação substancial (DAVIDSON; FREIRE, 2008; COSTA et al., 2012).

### 2.2.1 Modelos estruturais de aplicação

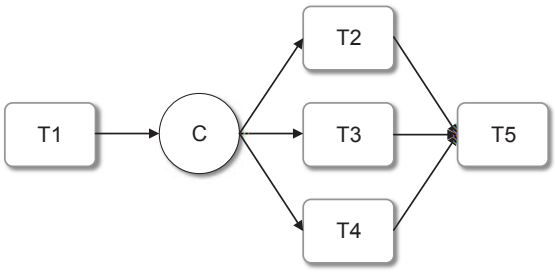
Segundo (SADIQ S., 1996), existem diversos modelos de estruturas de execução de aplicações de fluxos de tarefas, que permitem modelar a forma como as atividades vão sendo executadas ao longo de um processo, entre eles:

- **Sequencial** - é a construção mais básica de modelagem de *workflow*. Especifica a ordem em que as tarefas devem ser executadas, ligando modelagem de objetos com os fluxos. Conforme Figura 2, após a tarefa “T1” terminar sua execução, ela irá passar para a tarefa “T2”. A tarefa “T2” só pode iniciar após que a tarefa “T1” finalizar por completo.

- **Alternativo** - é utilizada para modelar dois ou mais caminhos alternativos mutuamente exclusivos. Este modelo tem um estado condicional que permite a escolha de um dos caminhos possíveis de execução. Para isso é necessário um conjunto de controle de dados e um estado de condição. Na Figura 3, é possível visualizar que após a tarefa “T1”, no estado “C” é tomada a decisão de qual será o próximo estado (“T2”, “T3” ou “T4”) que a aplicação deverá ser executada.

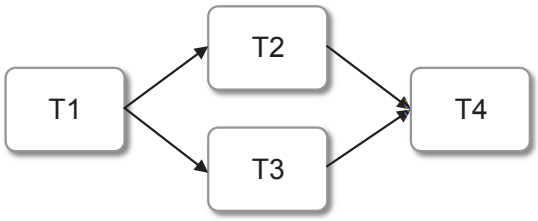
- **Junção Exclusiva** - é o oposto da construção alternativa. Tem o objetivo de juntar dois ou mais caminhos. É construída ligando dois ou mais fluxos de entrada para uma única

**Figura 3:** *Workflow* Alternativo




Fonte: Desenvolvido pelo autor.

**Figura 4:** *Workflow* Concorrente




Fonte: Desenvolvido pelo autor.

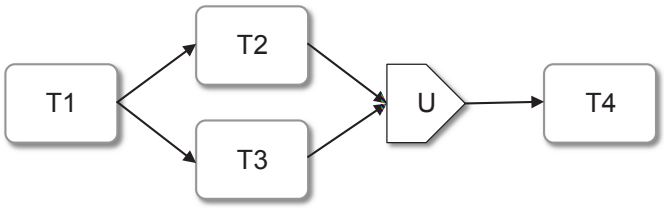
tarefa. Pode ser visto na Figura 3, aonde as tarefas “T2”, “T3” e “T4” convergem para a tarefa “T5”. Aonde a tarefa “T5” representa uma junção exclusiva das atividades anteriores.

- **Concorrente** - é o modelo em que as execuções acontecem de forma concorrente, possibilitando a execução de encaminhamento paralelo. É possível exemplificar com a Figura 4, na qual depois tarefa “T1”, a próxima etapa é dividida em 2 tarefas (“T2” e “T3”), depois volta a ser uma tarefa única novamente (“T4”).

- **Sincronização** - no modelo de sincronização é possível que haja diversas situações em que as atividades de vários fluxos concorrentes são colocados em modo de espera até que todas as tarefas tenham sido realizadas. Na Figura 5, a sincronização é representada por “U”, que permite que a tarefa “T4” comece a ser processada depois de concluídas as tarefas “T2” e “T3”.

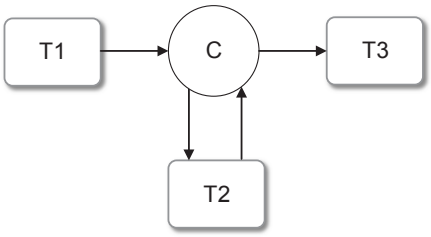
- **Iteração** - é representado por repetição de um conjunto de tarefas, de acordo com o estado de uma condição. Na Figura 6, a tarefa “T2” será executada repetidamente até que a condição “C” seja satisfeita e o fluxo passará a execução de “T2” para “T3”.

**Figura 5:** *Workflow* com Sincronização




Fonte: Desenvolvido pelo autor.

**Figura 6:** *Workflow* com Iteração




Fonte: Desenvolvido pelo autor.

## 2.3 Computação Paralela

Próximo do ano de 1980 acreditava-se que o desempenho do computador só poderia melhorar (crescer) através da criação de processadores mais rápidos e eficientes, ou seja, aumentar a frequência de processamento. Anos depois essa ideia foi contestada pelo processamento paralelo, que tem o significado de reunir dois ou mais computadores em conjunto para resolver um problema computacional. A partir do ano de 1990 houve uma forte tendência para deixar de utilizar os supercomputadores (equipamentos caros e muito específicos) para usar computadores em rede (PCs ou estações de trabalho). Esta ideia se tornou um veículo atraente para o processamento paralelo porque tem baixo custo e é eficiente. Com este conceito surgiram os Clusters de computação escaláveis. São clusters de PCs ou estações de trabalho (estrutura homogênea ou heterogênea) que rapidamente se tornaram padrão para as plataformas de alta performance e computação de grande escala (BUYYA, 1999).

O processamento paralelo é o processamento simultâneo de duas ou mais unidades de processamento. Este pode ser obtido por uma de duas abordagens principais: (I) utilizando *hardware* para processamento paralelo e/ou (II) usando vários processadores ligados através de uma interconexão, operando de forma cooperativa ou concorrente, na execução de um ou vários aplicativos (processamento distribuído). Na primeira abordagem, um sistema de computador que tem vários processadores é usado para realizar a tarefa de cálculo. Na segunda abordagem, a quantidade de máquinas ligadas em rede são utilizadas para realizar tarefas de computação. Ambas as abordagens têm seus prós e contras. No entanto, dependendo dos requisitos da apli-

cação e do orçamento disponível, a seleção da arquitetura já está determinada (PRAJAPATI; VIJ, 2011).

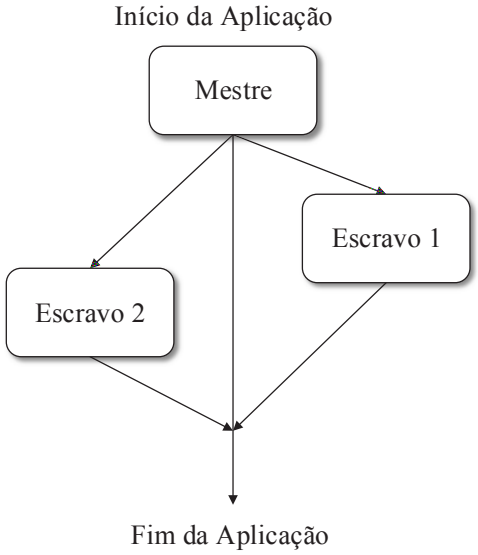
O modelo de paralelismo reflete na estrutura do aplicativo ou nos dados. O paralelismo resultante da estrutura da aplicação é nomeado como paralelismo funcional. Neste caso, partes diferentes do programa podem realizar tarefas diferentes de uma maneira simultânea e cooperativa. Mas paralelismo também pode ser encontrado nas estruturas de dados. Este tipo de paralelismo permite a execução de processos paralelos com operação idêntica mas em partes diferentes dos dados (BUYYA, 1999).

Tem sido largamente reconhecido que as aplicações paralelas podem ser classificadas em modelos de programação paralela. Cada modelo é uma classe de algoritmos que têm a mesma estrutura de controle. A escolha do modelo é determinada pelos recursos disponíveis de computação paralelas e pelo tipo de paralelismo inerente ao problema. Os recursos de computação podem definir o nível de granularidade que pode ser suportado eficientemente pelo sistema (BUYYA, 1999). Um dos principais problemas de desempenho em programação paralela é a escolha entre muitas tarefas pequenas (grão fino) ou algumas tarefas grandes (grão grosso) (BALDO et al., 2005). Neste cenário é possível citar três modelos de programação paralela: Mestre/Escravo, Divisão e Conquista e Pipeline.

**Mestre/Escravo** - No modelo Mestre/Escravo, um nó mestre gera e distribui a carga trabalho, geralmente em formato de tarefas, para os nós escravos. Cada nó escravo recebe tarefas, executa estas tarefas e envia o resultado de volta. Dessa maneira, o nó mestre é responsável por receber e analisar os resultados computados pelos nós escravos. Este modelo é geralmente adequado para a memória distribuída ou plataformas de passagem de mensagens. O principal problema neste modelo é que o comandante pode se tornar um gargalo. Isto pode acontecer se as tarefas são demasiadas pequenas ou se houver muitos escravos conectados a um mestre. A escolha da quantidade de trabalho em cada tarefa é chamado de granularidade. O modelo Mestre/Escravo permite interação síncrona que é quando as tarefas precisam ser executado por fases, ou seja, todas as tarefas em cada fase deve terminar antes que a distribuição de tarefas próxima fase. Também existe a interação entre o mestre e os escravos de forma assíncrona que é quando o mestre distribui novas tarefas cada vez que um escravo termina seu computação. A Figura 7 demonstra a estrutura deste modelo (BALDO et al., 2005; BUYYA, 1999).

**Divisão e Conquista** - O modelo de Divisão e Conquista tem como princípio que um problema é dividido em dois ou mais subproblemas. Cada um destes subproblemas é resolvido de forma independente e os seus resultados são combinados para dar o resultado final. Muitas vezes, os problemas menores são instâncias menores apenas do problema original, dando origem a uma solução recursiva. O processamento pode ser necessário para dividir o problema original ou para combinar os resultados dos subproblemas. Nesta perspectiva os subproblemas podem ser resolvidos ao mesmo tempo, dada suficiente capacidade de computação paralela. O processo de divisão e recombinação também faz uso do paralelismo e estas operações exigem comunicação entre os processos. No entanto, os subproblemas são independentes, a comunicação não se faz

**Figura 7:** Estrutura do modelo Mestre/Escravo

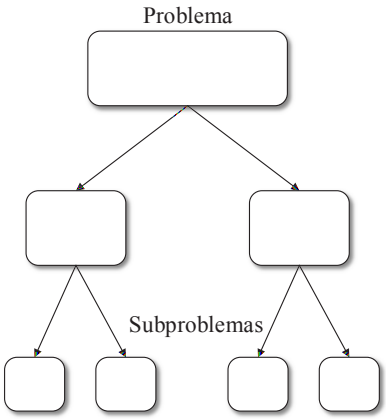



Fonte: Desenvolvido pelo autor.

necessária entre os processos de trabalho (entre subproblemas diferentes). Podemos identificar três operações computacionais genéricas para Divisão e Conquista: (I) Dividir, (II) Calcular (ou Processar) e (III) Unir o resultado. A aplicação deve estar organizada em uma espécie de árvore virtual: alguns dos processos criam subtarefas e tem que combinar os resultados daqueles para produzir um resultado agregado. As tarefas são, na verdade, calculadas pelos processos de computação nos nós da árvore virtual conforme mostra a Figura 8 (BUYYA, 1999; HOROWITZ; ZORAT, 1983).

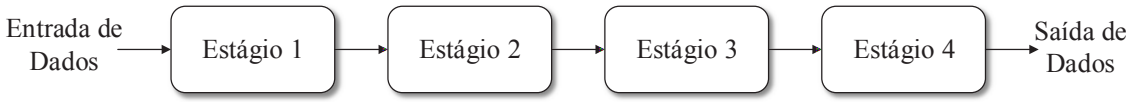
***Pipeline*** - Este pode ser considerado o paralelismo mais refinado porque é baseado em uma abordagem de decomposição funcional, no qual as tarefas do algoritmo são identificadas e cada processador executa uma pequena parte do algoritmo total. Os processos são organizados em etapas sequenciais. Cada processo corresponde a uma etapa do fluxo de processamento e são responsáveis por executar tarefas específicas. A Figura 9 apresenta a estrutura deste modelo. Os dados processados na sequência de execução, passam do estágio atual para o próximo, de forma adjacente, respeitando a ordem dos estágios. Por este motivo, este tipo de paralelismo é também conhecido como paralelismo de fluxo de dados. A comunicação entre os estágios pode ser completamente assíncrona. A eficiência desse modelo de paralelismo é diretamente dependente da capacidade de equilibrar a carga entre os estágios do *pipeline* (BUYYA, 1999; BHARATHI et al., 2008).

**Figura 8:** Estrutura do modelo Divisão e Conquista




Fonte: Desenvolvido pelo autor.

**Figura 9:** Estrutura do modelo Pipeline




Fonte: Desenvolvido pelo autor.

### 2.3.1 Séries Temporais e Análise de Desempenho

Em razão da adoção da computação em nuvem como alternativa para a execução de aplicações organizadas em *pipeline*, diversas estratégias para prover elasticidade são ofertadas pela plataforma de nuvem e trabalhos acadêmicos. Destas estratégias, diversas utilizam técnicas de monitoramento que dependem de 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 tornam-se relevantes em contextos de tomadas de decisões nas quais são baseadas em previsões futuras, o que pode impactar diretamente na qualidade de serviço e desempenho de uma determinada aplicação. Cada método possui diferentes parâmetros e características. Estas diferenças definem a complexidade e objetivo de cada um dos métodos. Um dos modelos mais usados para a previsão de séries temporais é o modelo ARIMA (*Autoregressive Integrated Moving Average*) (BOX; JENKINS; REINSEL, 2003). Análises muitas vezes envolvem a identificações de segmentos de séries temporais, onde ocorrem fenômenos e comparações entre segmentos de tempo para testes de padrões interessantes que podem ser utilizados para formar, provar ou refutar uma hipótese (WALKER; BORGO; JONES, 2015).

Existem métodos que necessitam de uma série de valores para seus cálculos. Os que utilizam *Moving Average* (MA) dependem de uma quantidade de valores fixas da janela de tempo para

calcular. Os que utilizam *Simple Exponential Smoothing* (SES) dependem de todos os valores coletados para a realização dos cálculos (HERBST et al., 2013). No caso do SES, quanto mais antigo o valor coletado menor seu peso para o cálculo final. Em adição, para essa técnica o valor mais recente recebe o peso maior, o segundo mais recente um pouco menor que o primeiro e assim sucessivamente. Com isso, é realizada uma suavização exponencial do valor reduzindo o impacto de ruídos nos valores coletados.

# 3 TRABALHOS RELACIONADOS

Este capítulo está organizado de maneira a apresentar a elasticidade em Nuvem Computacional, demonstrando a lacuna existente no gerenciamento de elasticidade para aplicação do modelo *pipeline* em ambiente de computação de alto desempenho. A análise dos assuntos explorados resulta em uma comparação de características relevantes para a realização deste trabalho.

## 3.1 Escolha dos Trabalhos Relacionados e Metodologia de Pesquisa

A pesquisa por trabalhos relacionados foi realizada utilizando o Portal de Periódicos CAPES/MEC[1]. De acordo com o site, o Portal de Periódicos disponibiliza acesso a textos completos disponíveis em mais de 37 mil publicações periódicas, internacionais e nacionais. Através de diversas bases de dados é possível realizar buscas que reúnem desde referências e resumos de trabalhos acadêmicos, científicos e outros tipos de materiais, cobrindo todas as áreas do conhecimento. Inclui uma coletânea de fontes importantes de informação científica e tecnológica na web.

Para a realização da pesquisa, foram utilizados os termos "*pipeline* + *elasticity*", "*elasticity* + *high performance computing*", "*pipeline* + *cloud computing*" e "*workflow* + *cloud computing*". Os resultados das pesquisas foram avaliados com o objetivo de encontrar trabalhos que explorassem a elasticidade em ambientes de computação em nuvem para aplicações no modelo *pipeline* (ou o mais próximo disso), também foram avaliados os trabalhos que exploram a utilização de aplicações de alto desempenho nos mesmos ambientes. Os trabalhos considerados são apresentados na Seção 3.2. Na Seção 3.3 é feita uma análise sobre as características de cada trabalho, pertinentes ao estudo realizado. Na Seção 3.4 é apontada a oportunidade de pesquisa encontrada.

## 3.2 Elasticidade e *Pipeline* em Nuvem

Li et al. (2010) apresentam em seu trabalho, um modelo de redução de processamento de volume de dados obtidos de satélites. Para isso é utilizada a plataforma de computação em nuvem Windows Azure. O estudo utiliza a aplicação no formato *pipeline* baseada em nuvem e tem como objetivo esconder as complexidades, processamentos subsequentes e transformação de dados para usuários finais. Esse *pipeline* é altamente flexível e extensível para acomodar diferentes tarefas de processamento de dados científicos, e pode ser escalado de forma dinâmica para cumprir vários requisitos computacionais. Os experimentos mostram que, executando uma carga de processamento de dados científicos em larga escala no *pipeline* utilizando 150 instâncias de máquinas virtuais Azure de médio porte, foi possível produzir resultados analíticos

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

de aproximadamente 90 vezes menos tempo na execução na aplicação do que com uma máquina *desktop* comum.

Rajan, Canino e Izaguirre (2011) estudaram as desvantagens da computação paralela como uma plataforma para construir e executar aplicações científicas de alto desempenho. Os autores afirmam que a computação distribuída proporciona mecanismos que podem superar as dificuldades da computação paralela. Para isso utilizam a computação em nuvem, que é construída sobre um modelo de computação distribuída. É descrito, no trabalho, o comportamento da implementação elástica sobre diferentes plataformas (Amazon EC2 e Windows Azure). Com o objetivo de explorar a elasticidade, é executada uma troca molecular dinâmica que explora o modelo de computação Mestre Trabalhador (Mestre Escravo). Como resultados são observados: (i) uma melhor escalabilidade, (ii) adaptabilidade dos recursos de tempo de execução, (iii) tolerância a falhas e (iv) portabilidade entre plataformas de computação em nuvem, exigindo o mínimo esforço e intervenção do usuário.

Mao e Humphrey (2011) desenvolveram um mecanismo de escalonamento de recursos em Nuvem Computacional para os modelos de aplicações em *workflow*. O objetivo desta abordagem é finalizar todos os trabalhos nos prazos especificados pelo usuário de forma mais eficiente e consumindo o mínimo de recursos possíveis. Para isso foi desenvolvido um gerenciador que adapta os recursos às mudanças de forma dinâmica (automaticamente). Os testes foram realizados em três modelos de *workflows*: Pipeline, Paralelo e Híbrido. Esta abordagem foi avaliada em quatro padrões de carga de trabalho (estável, crescente, oscilatório e intercalado em baixo e alto) representativos em Nuvem Computacional e demonstraram redução de custos de 9,8% para 40,4%, em comparação com outras abordagens.

Com o objetivo de melhorar o desempenho e reduzir custos em ambiente de Nuvem Computacional, Sharma et al. (2011) criaram um protótipo chamado *KingFisher*. Esse sistema oferece elasticidade para plataforma de nuvens públicas e privadas e tem suporte tanto para Amazon EC2 (nuvem publica) quanto para nuvens baseadas em Xen (privadas). Através de uma plataforma provedora de nuvem, este modelo fornece suporte para a implementação e gestão de imagem de MVs em conjunto com um método de elasticidade no ambiente. *KingFisher* utiliza uma versão modificada dos mecanismos e ferramentas disponibilizadas pelo OpenNebula. Através da API XML-RPC é possível criar, consolidar ou reconfigurar as MVs dos servidores. Seu funcionamento é baseado em *IaaS* e sua elasticidade segue o modelo automático reativo e proativo. Com isto foi capaz de reduzir os custos dos recursos de servidores virtuais e reduzir o tempo da aplicação em 24%.

No trabalho de Apostol et al. (2011) é apresentado um mecanismo de gerenciamento para sistemas em Nuvem Computacional. O modelo proposto aloca recursos virtuais de forma flexível, levando em consideração o tempo, o custo e os recursos físicos. Ele fornece elasticidade sobre demanda ou com base em políticas pré-definidas. Através de um algoritmo genético, o mecanismo avalia a capacidade das MVs e/ou políticas definidas e executa uma ação para reorganizar as MVs de modo que o ambiente atinja o melhor uso dos recursos. Este algoritmo

genético avalia a "qualidade" (recursos) das populações (MVs) de acordo com suas características (CPU, memória RAM e disco). A partir dessa avaliação reorganiza as MVs com o objetivo de conquistar maior desempenho e escalabilidade. Os hypervisors utilizados são Xen e KVM, por suportarem virtualização total e otimizarem a execução no ambiente de testes. A solução proposta neste trabalho é eficiente e tem uma boa taxa de utilização.

Foi desenvolvido por Imai, Chestna e Varela (2012) o *Cloud Operation System* (COS, Sistema Operacional de Nuvem). Trata-se de uma estrutura de *middleware* para suportar cargas de trabalho de forma elástica, escalável e sem a intervenção do usuário em Nuvem Computacional. O trabalho é baseado em uma estratégia de aplicação de reconfiguração de MVs para conseguir desempenho e redução de consumo energético. Para escalar uma carga de trabalho esse modelo cria novas MVs e migra os atores das aplicações para essas novas MVs. O mesmo ocorre quando um nó fica com recursos ociosos. Com a migração desses atores é possível desligar as MVs sem utilização. Para criar ou desligar uma MV é utilizada a estratégia de heurística. Através de uma elasticidade automática e proativa, o gerente de recursos do COS decide criar uma nova MV somente se ele recebeu eventos de alta utilização da CPU de todos os nós que funcionam dentro de um determinado intervalo de tempo (por exemplo, 10 segundos). A razão disso é que podem existir ainda MVs subutilizadas capazes de equilibrar a carga de processamento e diminuir a utilização geral da CPU das máquinas virtuais. Dessa forma é possível conseguir uma granularidade mais fina.

Intitulado de *Scattered*, Zhang et al. (2012) desenvolveram um algoritmo para ambientes de Nuvem Computacional que tem a função de fazer o balanceamento dos recursos das MVs com o propósito de reduzir o risco de sobrecarga do centro de dados da nuvem. Para realizar este algoritmo foram considerados apenas o CPU das MVs e a rede de comunicação entre elas. Primeiro é feita uma análise de toda a nuvem, depois o algoritmo procura a melhor MV para ser migrada e após isso procura o melhor nó para receber essa MV. Esta abordagem tem a vantagem de minimizar o número de migrações necessárias entre as MVs, reduzindo o impacto na execução dos aplicativos.

Rodriguez e Buyya (2015) apresentam uma abordagem de escalonamento de recursos em Nuvem Computacional para aplicações científicas do tipo *workflow*. Neste artigo é explorado o modelo de serviço *IaaS* com a alocação de tarefas em MVs (Máquinas Virtuais), de maneira a ser utilizado o mínimo de MVs necessárias para executar a aplicação de forma mais eficiente possível. Também é considerado o uso da menor quantidade de tempo dos recursos para economizar gastos, visto que a simulação é realizada com base no *Google Cloud Plataform* [2]. Foi modelado um centro de dados (*datacenter*) com quatro tipos de MVs e os recursos elásticos foram aplicados sob demanda. Este modelo explora a estratégia de provisionamento de recursos que funciona em conjunto com o algoritmo de escalonamento. Também é utilizada uma heurística que decide não só o tipo e número de MVs, mas também quando é o melhor momento para alocá-las e desalocá-las. Os testes e avaliações de escalonamento foram realizados através de

[2]https://cloud.google.com/

simulação e comparados com dois algoritmos:

- O primeiro é um algoritmo que atribui prazos para as tarefas e aloca elas em MVs (existentes ou em novas) de forma a minimizar custos. É um algoritmo estático capaz de gerar soluções de alta qualidade. Sua desvantagem é a sua incapacidade de cumprir os prazos quando ocorrem atrasos inesperados (MALAWSKI et al., 2012).

- O segundo é um algoritmo dinâmico que tem um mecanismo que aloca e desaloca MVs com base no status atual de tarefas. Ele determina o tipo de MV para cada tarefa mais eficiente em termos de custo e cria um vetor de carga. Este vetor é atualizado a cada ciclo de programação e indica quantas MVs de cada tipo são necessários para realizar as tarefas no prazo com um custo mínimo (MAO; HUMPHREY, 2011).

Os resultados apresentaram que o modelo elástico proativo dos autores se mostrou mais eficiente do que os demais.

Righi et al. (2015) propõem um modelo de elasticidade para aplicações HPC baseado em *PaaS*, intitulado de AutoElastic. A principal ideia desta proposta é prover elasticidade horizontal em aplicações que demandam alto processamento sem a intervenção do usuário. Para isso é utilizado o modelo de programação paralela Mestre/Escravo, no qual o nó Mestre distribui tarefas para o(s) nó(s) Escravo(s), o mesmo explora o conceito de MPMD (*Multiple Program Multiple Data*, múltiplo programa múltiplos dados), em que várias MVs autônomas executam simultaneamente um tipo de programa. Através de *thresholds* (limites) predefinidos, o *middleware* OpenNebula consegue incluir ou consolidar recursos (MVs) em tempo de execução para prover maior desempenho de maneira elástica, dado que cada MV é gerada para cada tipo de programa. Os autores ainda inserem quatro padrões de carga no sistema: constante, ascendente, descendente e onda. Comparado com um sistema não elástico, o AutoElastic se mostrou mais eficiente em 5 de 6 cenários testados.

## 3.3 Análise dos trabalhos relacionados

Com a ajuda de uma tabela, é possível elencar as principais características consideradas. A análise dos trabalhos consistiu na avaliação das seguintes características:

- **Modelo de Serviço**: nível de operação em que o ambiente de nuvem é aplicado;

- **Método de Elasticidade**: faz referência ao método que a elasticidade é fornecida (recursos e propriedades);

- **Modelo(s) de Elasticidade**: qual a estratégia abordada pelos autores com relação ao(s) modelo(s) de elasticidade escolhido(s);

- **Elasticidade por estágio(s) em *Pipeline***: se o trabalho fornece elasticidade independente por estágios para aplicações organizadas no modelo *pipeline*;

- **Modelo de Computação Paralela**: qual/quais modelos de computação paralela são abordados no trabalho;

- **Carga de Trabalho**: apresenta qual o objetivo da aplicação que é executada;

Com a ajuda da Tabela 1 é possível visualizar a comparação entre as principais características citadas na análise.

**Tabela 1:** Comparação entre as principais características dos trabalhos relacionados. (NI - Não Informado)

| **Trabalho** | **Modelo de Serviço** | **Método de Elasticidade** | **Modelo(s) de Elasticidade** | **Elasticidade por estágio(s) em *Pipeline*** | **Modelo de Computação Paralela** | **Carga de Trabalho** |
|---|---|---|---|---|---|---|
| (LI et al., 2010) | *PaaS* | Horizontal | Manual | Não fornece | *Pipeline* | Redução de Imagens de Satélites |
| (RAJAN; CANINO; IZAGUIRRE, 2011) | *PaaS* | Horizontal | Manual | Não fornece | Mestre Escravo | Troca Molecular Dinâmica |
| (MAO; HUMPHREY, 2011) | *IaaS* | Horizontal | Automático Reativo e Proativo | Não fornece | *Pipeline*, Paralelo e Híbrido | Aplicações no modelo *Workflow* |
| KingFisher (SHARMA et al., 2011) | *IaaS* | Horizontal | Automático Reativo e Proativo | Não fornece | *Bag of Task* | Saturação de E-commerce |
| (APOSTOL et al., 2011) | *IaaS* | Horizontal | Automático Reativo | Não fornece | NI | NI |
| Cloud Operation System (IMAI; CHESTNA; VARELA, 2012) | *PaaS* | Horizontal | Automático Proativo | Não fornece | NI | Problema de Difusão de Calor |
| Scattered (ZHANG et al., 2012) | *IaaS* | Horizontal | Automático Proativo | Não fornece | *Workflows* e *Bag of Pipeline* | Aplicação Comercial Real |
| (RODRIGUEZ; BUYYA, 2015) | *IaaS* | Horizontal | Automático Reativo e Proativo | Não fornece | *Pipeline*, *Bag of Task* e *Bag of Pipeline* | Aplicações no modelo *Workflow* |
| Auto-Elastic (RIGHI et al., 2015) | *PaaS* | Horizontal | Automático Reativo | Não fornece | Mestre Escravo | Cálculo Diferencial |

Fonte: Desenvolvido pelo autor.

## 3.4 Oportunidade de Pesquisa

De maneira geral é possível afirmar que a elasticidade em nuvem computacional é uma abordagem amplamente utilizada para aplicações de requerem alto processamento e que exploraram desempenho e o custo dos recursos. Mao e Humphrey (2011), Zhang et al. (2012) e Rodriguez e Buyya (2015) exploram a nuvem computacional com o modelo de serviço *IaaS* e modelo de elasticidade automático para conquistar melhor desempenho e redução de custos com infraestrutura em aplicações *workflow*. Já Righi et al. (2015) aborda o modelo de serviço PaaS junto com modelo de elasticidade automático sobre a nuvem para adquirir melhor desempenho em aplicações iterativas que demandam alto poder de processamento.

Os trabalhos que abordam aplicações organizadas em *pipeline* apresentam lacunas à serem exploradas no que referem-se ao monitoramento dos estágios da aplicação e tomadas de decisões elásticas de maneira independente. Conforme mostra a coluna “Elasticidade em *Pipeline*” da tabela 1, é possível perceber que nenhum dos trabalhos têm esse nível de gerenciamento

de elasticidade de recursos. Considerando as características apresentadas, a oportunidade de pesquisa que este estudo identificou é o gerenciamento independente da elasticidade de cada estágio da aplicação organizada em *pipeline* de forma automática e sem a necessidade de intervenção do usuário.

Todos os trabalhos pesquisados utilizaram o método horizontal de elasticidade. A maioria dos trabalhos que exploram *workflows* utilizam o modelo automático de elasticidade. A maioria dos trabalhos que apresentaram melhores resultados de desempenho utilizaram a elasticidade de forma reativa. Desta maneira, será utilizada a elasticidade horizontal, automática e reativa no modelo proposto neste trabalho. Ainda sobre computação em nuvem, estes conceitos se tornam necessários para o bom entendimento do contexto em que será realizado este trabalho.

# 4 MODELO PIPEL

Este capítulo tem como objetivo apresentar o Pipel, um sistema criado para prover elasticidade de maneira automática e dinâmica nos estágios de aplicações no formato *pipeline* em ambiente de computação em nuvem. A Seção 4.1 apresenta as decisões de projeto para a realização do modelo proposto. A Seção 4.2 apresenta a arquitetura e componentes do Pipel de forma detalhada. A Seção 4.3 demonstra como a elasticidade é aplicada sobre o sistema. Por fim, o método de avaliação da proposta é apresentado na Seção 4.4.

Desenvolver um modelo para nuvem computacional, que possa tirar proveito da elasticidade não é uma tarefa trivial. A aplicação deve ser pensada para estar particionada em vários nós diferentes. Deve ser possível adicionar mais nós à infraestrutura facilmente e ter benefícios imediatos na escalabilidade do sistema. A aplicação deve continuar disponível mesmo quando um nó seja consolidado ou reescalonado para outra atividade dentro da nuvem. Os componentes da aplicação também precisam coordenar suas atividades, saber como escalonar tarefas paralelas e possuir capacidade de se comunicar de forma assíncrona. Essas são as premissas básicas que tornam uma aplicação adequada a um ambiente elástico de computação em nuvem.

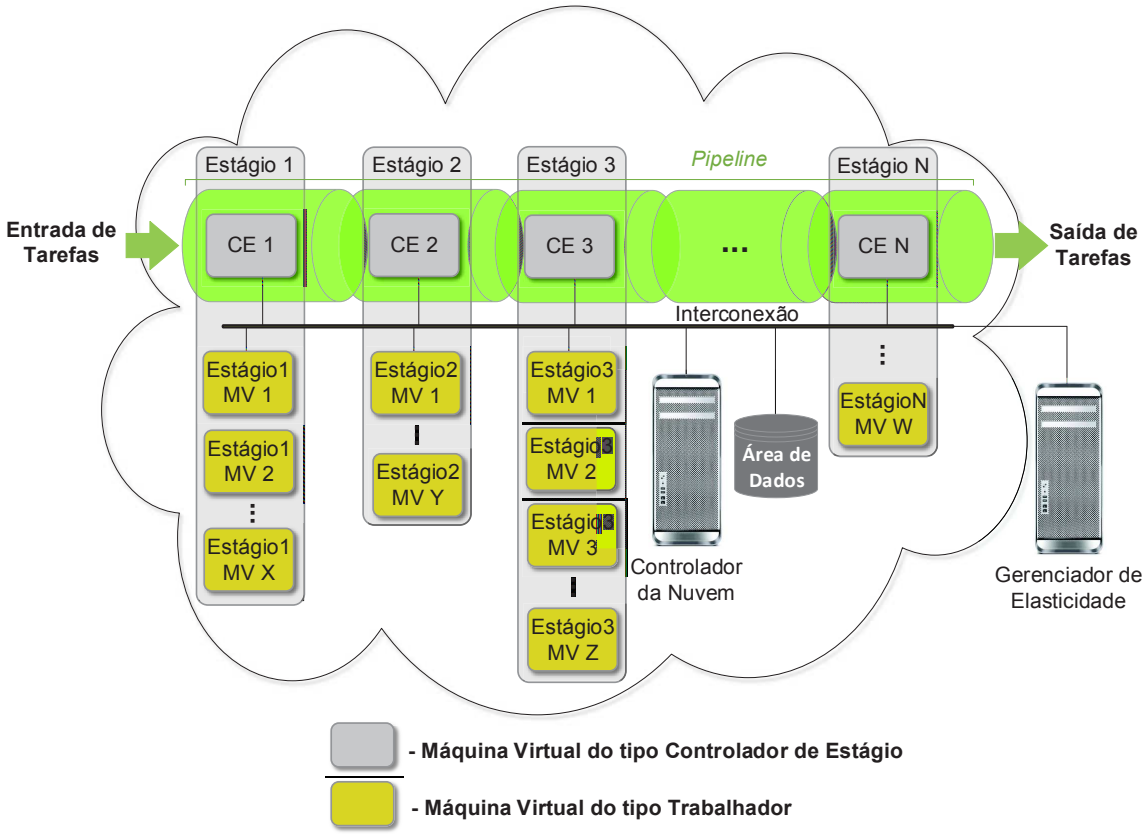
## 4.1 Decisões de Projeto

O modelo Pipel proporciona ações elásticas na infraestrutura de nuvem computacional de forma reativa, horizontal e automática para aplicações *pipeline*. Tudo isso ocorre em tempo de execução, de forma transparente para a aplicação e sem que seja necessária a intervenção do usuário. Assim é possível acrescentar ou remover recursos de computação (máquinas virtuais) e ainda se torna factível reutilizar as máquinas virtuais já instanciadas entre os estágios, caso necessário.

As características que fazem parte do escopo do modelo proposto são:

- Criar um mecanismo de coordenação entre os nós e as máquinas virtuais existentes, sabendo qual máquina virtual pertence a cada estágio e quando um nó está ativo ou inativo;
- Realizar balanceamento de carga em cada estágio, de forma que as máquinas virtuais de cada estágio não sejam sobrecarregadas;
- Utilizar uma estratégia de comunicação entre os componentes do Pipel que seja assíncrona e opere independentemente da dimensão atual da infraestrutura da nuvem;
- Distribuir tarefas entre as várias máquinas virtuais distintas, independentes de sua capacidade de processamento;
- Garantir a elasticidade em cada estágio (micro) e também no sistema como um todo (macro).

**Figura 10:** Arquitetura do Pipel




Fonte: Desenvolvido pelo autor.

## 4.2 Arquitetura Proposta

Pipel é um modelo projetado para operar em nível de *PaaS*, utilizando uma infraestrutura de nuvem privada, que permite aplicações organizadas em *pipeline* não elásticas obterem os benefícios da elasticidade em nuvem computacional sem a necessidade de intervenção do usuário. Para prover elasticidade o modelo opera com alocação, consolidação e reorganização de instâncias de máquinas virtuais e máquinas físicas (nós). Como Pipel é dimensionado para processar aplicações organizadas em *pipeline* e cada fluxo de trabalho contempla estágios, a elasticidade é provida/fornecida de forma independente em cada estágio de forma exclusiva. Ou seja, cada Controlador de Estágio (CE) é responsável por receber as requisições de processamento e enviar para as *N* quantidades de máquinas virtuais alocadas para o mesmo. Cada estágio possui um determinado número de máquinas virtuais em operação, distribuindo as tarefas entre elas. O controlador do estágio recebe as requisições, as quais ele é responsável por executar. Estas requisições são colocadas em uma fila e na medida em que são executadas (uma de cada vez), o CE distribui para as máquinas virtuais disponíveis em seu estágio. Cada etapa é monitorada pelo Gerenciador de Elasticidade (GE) em cada estágio e na aplicação de forma geral. De acordo com as regras estabelecidas no GE o Pipel aplica as ações elásticas.

A arquitetura do modelo Pipel é apresentada na Figura 10. A ilustração da arquitetura mostra a organização das máquinas virtuais para que haja o funcionamento do fluxo de trabalho, tornando possível a execução de aplicações no formato *pipeline*. É apresentado também o Controlador da Nuvem (CN) computacional e o Gerenciador de Elasticidade. O CN opera em uma máquina física dentro da nuvem e o GE, que é responsável por monitorar e tomar as decisões sobre as ações de elasticidade no sistema, não possui dependência de localização para a sua execução, sendo possível estar localizado em qualquer ambiente (dentro ou fora da nuvem). Isto é possível através do uso de uma API fornecida pelo *middleware* de nuvem utilizado, o qual faz necessário apenas uma conexão de rede. Por fim, a arquitetura contempla uma área de dados compartilhada entre o Gerenciador de Elasticidade e os Controladores de Estágio. Esta área de dados faz com que a comunicação entre os estágios seja possível.

Aplicações organizadas em *pipeline* recebem dados de entrada para produzir dados de saída em cada estágio de execução. Neste caso, cada estágio de trabalho de entrada opera após a saída da etapa anterior e a saída produzida é alimentada como entrada para a próxima fase no *pipeline* (BUYYA, 1999; BHARATHI et al., 2008). Para que a estrutura de *pipeline* aconteça, deve-se respeitar uma premissa básica: todas as cargas de trabalho devem ter o mesmo número de estágios. Em outras palavras, o processamento dos fluxos de trabalho deve ser o mesmo porém com dados diferentes. No Pipel, o usuário insere os dados que deseja processar. Estes dados seguem a ordem sequencial e devem passar obrigatoriamente por todos os estágios da aplicação.

A Elasticidade é definida pelo grau em que um sistema é capaz de se adaptar à carga de trabalho através de alterações de provisionamento e desprovisionamento de recursos de forma autônoma. Sendo assim, em determinado ponto da linha de tempo da aplicação, os recursos disponíveis correspondem a demanda da carga de trabalho. A elasticidade garante uma melhor utilização de recursos de computação, proporcionando escalabilidade ao sistema, caso haja necessidade (GALANTE; BONA, 2012; HERBST et al., 2013). Pipel utiliza o conceito de elasticidade horizontal, automática e reativa. A elasticidade horizontal foi escolhida pois a elasticidade vertical tem limitações entre as dependências de recursos disponíveis em um único nó e também a maioria dos sistemas operacionais mais utilizados não permitem que sejam agregados recursos em tempo de execução (DUTTA et al., 2012; RIGHI et al., 2015). A elasticidade automática e reativa tem como ponto de partida regras previamente definidas. Estas regras devem ser analisadas, uma vez que envolve uma série de decisões sobre configuração de limites de processamento e de tempo, tipos de recursos, quantidade de coletas sobre o sistema, definições sobre anormalidades pontuais e frequência de monitoramento. Segundo Righi et al. (2015) é normal existir momentos em que regras de elasticidade adquiram resultados positivos para uma determinada aplicação e resultados não tão positivos para outras. A elasticidade transparente refere-se ao fato de que a aplicação não precisa ter seu código fonte alterado, sendo assim, Pipel não evidencia todas as ações de adição/consolidação/reorganização de seus recursos ao programador.

Righi et al. (2015) e Chen et al. (2015), em seus trabalhos, afirmam que aplicações HPC em geral usam o poder de processamento (CPU) de forma intensiva. De acordo com LEE et al. (2011), em seu trabalho são exploradas técnicas para encontrar o melhor desempenho e eficiência em aplicações paralelas. Como resultado é apresentada que a melhor abordagem é utilizar um processador (*core* de CPU de um nó) em cada máquina virtual e também distribuir um processo para cada máquina virtual. Dito isto, o Pipel faz uso desta abordagem. As máquinas virtuais instanciadas ou consolidadas possuem apenas um processador de determinado nó, e para cada máquina virtual só será enviado um processo por vez. Os Controladores de Estágio também são máquinas virtuais instanciadas que controlam as filas de tarefas, dividem as tarefas em subtarefas e as distribuem para as máquinas virtuais alocadas para o estágio em questão. Após a realização do processamento são responsáveis por agrupar e encaminhar o resultado para o próximo estágio da aplicação, mantendo assim a continuidade da aplicação. Cada máquina virtual é alocada para um estágio específico. Uma vez que esta está alocada para o estágio 1, por exemplo, a mesma só irá processar requisições solicitadas pelo controlador do estágio 1 até que ela seja consolidada ou transferida para outro estágio.

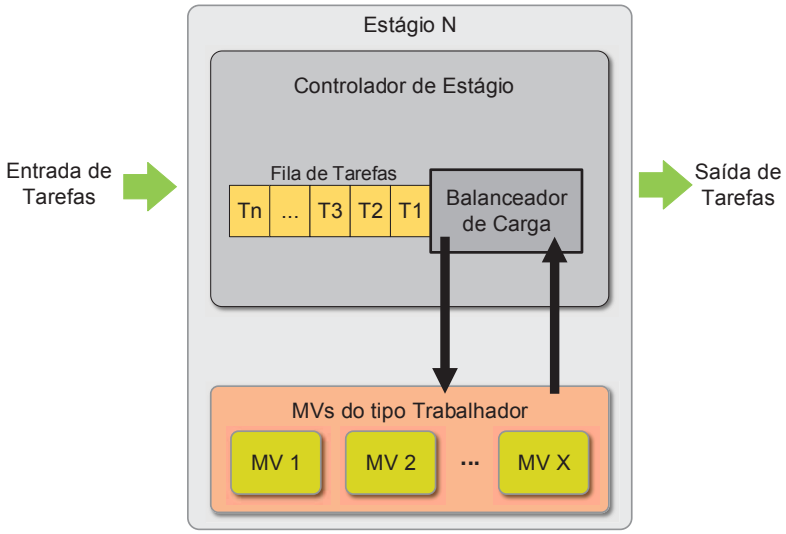
### 4.2.1 Controladores de Estágios e Trabalhadores

Os Controladores de Estágios são responsáveis por gerenciar os estágios aos quais lhes são atribuídos. Em outras palavras, cada CE é alocado em um estágio específico da aplicação e desta forma tem como objetivos: (i) receber as tarefas; (ii) armazená-las em uma fila; (iii) dividir cada tarefa para os *n* Trabalhadores (máquinas virtuais) disponíveis em seu estágio; (iv) aguardar até que todos os Trabalhadores retornem os resultados; (v) concatenar os resultados de retorno de todos Trabalhadores e por fim (vi) encaminhar o resultado da tarefa para o próximo estágio. A Figura 11 ilustra este cenário.

Diversos trabalhos exploram o modelo de computação paralela Mestre/Escravo em ambiente de nuvem computacional (KUMAR et al., 2014; TRAN et al., 2015; RIGHI et al., 2015; GUOXIN; WANLI; LIRONG, 2015). O modelo de computação paralela utilizado em cada estágio é o Mestre/Escravo, onde o CE é o Mestre e os Trabalhadores são os Escravos. O Mestre é responsável por organizar toda a computação realizada em seu estágio, dividindo as tarefas em subtarefas para encaminhar para seus respectivos Escravos, aguardando o retorno e gerando a saída do estágio atual e a entrada para o próximo estágio.

Para prover os recursos autonômicos necessários para o CE de cada estágio, como: elasticidade integrada com balanceamento de carga e o monitoramento tanto das máquinas virtuais existentes quanto das aplicações sendo executadas nestas máquinas. Cada Controlador de Estágio opera em uma instância de máquina virtual a partir de uma imagem específica. Para cada CE ativo também é instanciada pelo menos uma máquina virtual do tipo Trabalhador. O Trabalhador assim como o CE, possui uma imagem específica. A quantidade de estágios da aplicação determina quantas máquinas virtuais do tipo CE serão instanciadas e também o mínimo de ins-

**Figura 11:** Controlador de Estágio e Máquinas Virtuais do tipo Trabalhador, no Estágio N.




Fonte: Desenvolvido pelo autor.

tâncias do tipo Trabalhador. Já que, tanto os CEs quanto os Trabalhadores, possuem imagens específicas (a mesma para cada tipo), podem ser instanciados o número de CEs e Trabalhadores que: (i) o usuário definir no SLA; (ii) que a aplicação necessitar; ou ainda (iii) que a estrutura física (nós) suportar. A quantidade de máquinas virtuais do tipo Controlador de Estágio depende da quantidade de estágios que a aplicação necessita. Sendo assim, é alocado um CE para cada estágio.

Para determinar a quantidade total de máquinas virtuais (*QTMV*) que estão em operação (ativas) no ambiente da nuvem é utilizada a expressão 4.1.

$$QTMV = \sum_{i=1}^{n}(1 + qmvt_i) \tag{4.1}$$

Onde a variável *qmvt* representa a quantidade de máquinas virtuais Trabalhadoras alocadas para o estágio $i$ e a variável $i$ é o índice do somatório que designa os estágios, de 1 até a quantidade de estágio existentes ($n$).

### 4.2.2 Ações de Elasticidade

O modelo proposto contempla um módulo de gerenciamento da utilização de recursos dos Trabalhadores (máquinas virtuais), provendo ações elásticas consideradas necessárias para o instante atual de execução da aplicação sobre a infraestrutura da nuvem computacional. Este módulo monitora e executa ações elásticas em paralelo com a execução da aplicação. Este artifício foi desenvolvido preferencialmente para executar aplicações organizadas em *pipeline*. Neste contexto o usuário pode informar alguns parâmetros no SLA contendo a quantidade Má-

xima e Mínima de máquinas virtuais do tipo Trabalhadores. Caso o usuário não informe um SLA, a aplicação começa com o mínimo de Trabalhadores podendo chegar até o limite do recurso físico caso considere necessário. Lembrando que uma instância de CE e pelo menos uma instância Trabalhador são obrigatórias por estágio. Dessa maneira, a expressão 4.2 define a quantidade mínima de máquinas virtuais (*minMV*) necessárias para a aplicação executar, sendo *qe* a quantidade de estágios da aplicação. A expressão 4.3 define o máximo de máquinas virtuais que a nuvem computacional suporta. A variável *qcd* representa a quantidade de *cores* de processamento disponíveis em cada nó físico. *i* é o índice que representa cada nó, de 0 até a quantidade máxima de nós (*qn*), na infraestrutura de nuvem.

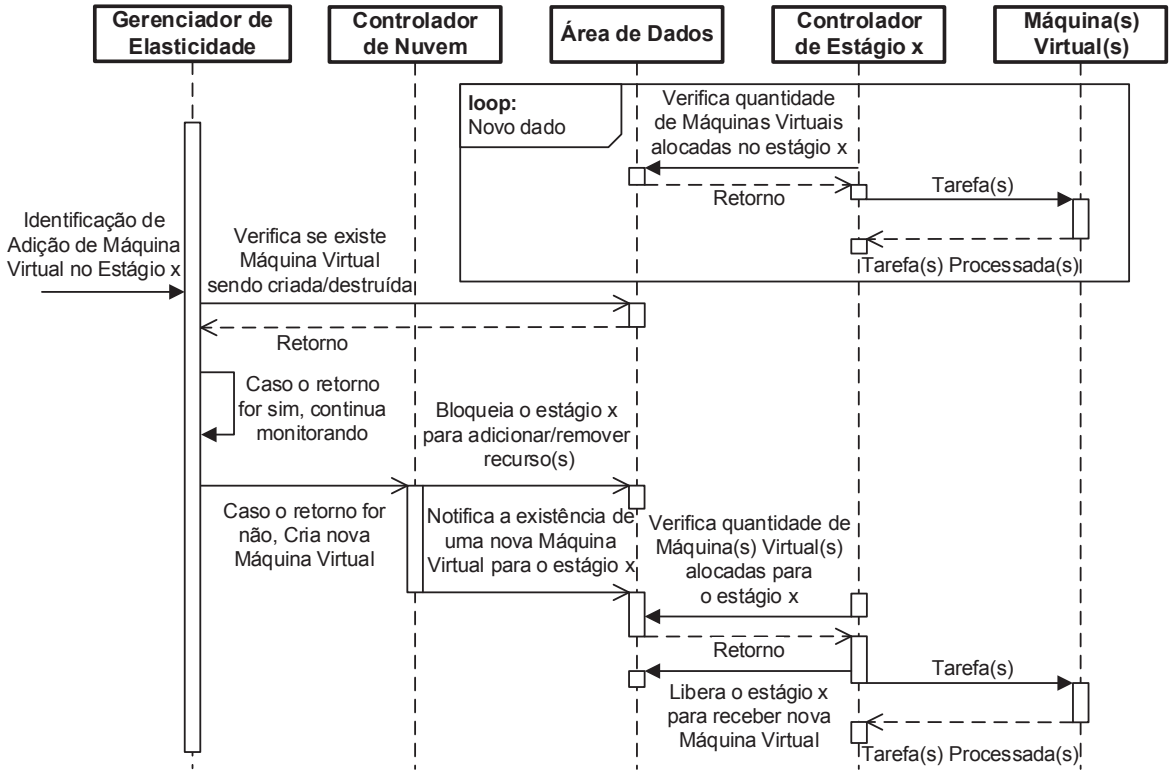
$$minMV = qe \times 2 \tag{4.2}$$

$$maxMV = \sum_{i=0}^{q_n} qcd(i) \tag{4.3}$$

Righi et al. (RIGHI et al., 2015) afirma que a elasticidade assíncrona torna-se viável com a utilização da elasticidade horizontal e isso faz com que não impacte 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. Pipel faz uso desta abordagem. O módulo de gerenciamento monitora e executa as operações de reconfiguração dos recursos da nuvem e o programador da aplicação não necessita dimensioná-la para as ações de elasticidade. Ou seja, ela não participa dessas ações com eventuais paradas de execução. Dessa maneira, para Pipel, a elasticidade é provida em todos os estágios da aplicação de maneira individual, simultânea e transparente. Este comportamento caracteriza o conceito de Múltipla Elasticidade Assíncrona.

Como todos estágios são monitorados de forma individual, cada um pode sofrer ações de elasticidade diferentes. Enquanto um está instanciando mais Trabalhadores, outro pode estar os consolidando. Ainda é feita uma verificação no sistema, caso haja a necessidade de consolidar Trabalhadores: são revisados em todos estágios se algum deles irá instanciar novos Trabalhadores. Caso isso ocorra o estágio que tinha intenção de consolidar seus Trabalhadores, os cede para o estágio que tinha intenção de instanciar novos, ganhando tempo na inicialização de seus sistemas operacionais. O Tempo de *boot* das máquinas virtuais é considerado nesta abordagem. Caso haja necessidade apenas de instanciar novos Trabalhadores os mesmos recebem o comando de inicializar e o módulo de gerenciamento verifica se já existe alguma atividade de alocação ou consolidação no estágio em questão. Caso haja alguma atividade o Gerenciador de Elasticidade continua o monitoramento. Caso não haja ele envia uma mensagem para o Controlador de Nuvem para adicionar mais uma máquina virtual no estágio em questão. Após a nova máquina virtual estar disponível para operação, é adicionada uma notificação na Área de Dados compartilhada notificando para o Controlador de Estágio que mais um recurso encontra-se

**Figura 12:** Diagrama de sequência no instante em que é identificada a necessidade de adicionar uma máquina virtual em algum estágio da aplicação.




Fonte: Desenvolvido pelo autor.

disponível para ele. Antes de cada tarefa que o Controlador de Estágio distribui para os Trabalhadores, ele verifica se existe recursos (máquinas virtuais) para adicionar ao estágio. Se o CE identifica que isso é verdadeiro inclui a máquina virtual na próxima execução e notifica, através da Área de Dados compartilhada, que está disponível para receber ações elásticas novamente. Liberando, assim, novas tomadas de decisões para o Gerenciador de Elasticidade. A operação de identificação e adição de uma nova Máquina Virtual é ilustrada pela Figura 12.

A estratégia de múltipla elasticidade assíncrona é possível em razão da existência de uma região de dados compartilhados entre o módulo de gerenciamento e a nuvem Pipel. Esta abordagem é comum em uso de nuvens privadas (CHEN; ZHU, 2014; SAHHAF et al., 2015; SZABO et al., 2015; RIGHI et al., 2015).

### 4.2.3 Método de Alocação de Recursos

Nos ambientes de computação em nuvem, é importante manter sob controle a alocação de máquinas virtuais nos servidores físicos. Uma alocação adequada implica na redução de custos com *hardware*, energia e refrigeração, além da melhora da qualidade de serviço (MUCHALSKI; MAZIERO, 2014). Para tal, deve-se conhecer a demanda de recursos de cada máquina virtual,

o ambiente de execução e as políticas específicas do *datacenter* para possíveis conflitos. De modo geral, a alocação de uma máquina virtual é dividida em duas etapas: (i) inicialmente é estimada a demanda de recursos da máquina virtual; (ii) em seguida, escolhe-se o servidor onde essa máquina virtual será alocada (MISHRA; SAHOO, 2011).

Considerando um conjunto de servidores físicos em um *datacenter*, a alocação de MVs consiste em, dada uma MV, encontrar um servidor que seja adequado para instanciá-la. O servidor deve possuir recursos suficientes para garantir a execução da MV sem afetar os acordos de nível de serviço. Esse problema representa um desafio algorítmico por sua característica combinatória, onde um conjunto de *y* máquinas virtuais devem ser alocadas em *x* nós computacionais (MUCHALSKI; MAZIERO, 2014). O pseudocódigo 1 mostra uma abordagem de Alocação Sequencial de Recursos (ASR). Nesta abordagem, o gerenciador de elasticidade captura a informação de todos os nós computacionais em formato sequencial. Quando encontra um que tenha disponibilidade de alocação de recursos, pega o ID (*Identity Document*, Documento de Identidade) do nó e envia uma MV para ele. Para identificar qual a Disponibilidade de Alocação de Recursos (*DAR*) que o nó possui é utilizada a Equação 4.4. Ela subtrai a quantidade de *cores* utilizados (*QCU*) da quantidade total de *cores* (*QTC*) do nó *x*. Esta equação tem como resultado a quantidade de máquinas virtuais que o nó está apto a receber.

$$DAR(x) = QTC(x) - QCU(x) \tag{4.4}$$

**Algoritmo 1:** Pseudocódigo da alocação sequencial de recursos na estrutura de nuvem computacional

```
1  início
2      int x=1;
3      enquanto (x <= quantidadeNós) faça
4          se (DAR(x)>=1) então
5              nóDiponível = x;
6              x=quantidadeNós;
7          fim
8          x++;
9      fim
10     alocarMV(nóDisponível);
11 fim
```

Como aplicações organizadas em *pipeline* possuem mais de um estágio e cada estágio pode alocar recursos de forma simultânea, o método sequencial pode não ser o mais adequado. Caso haja a necessidade de alocação de recursos em intervalos de tempo próximos em mais de um estágio, os mesmos podem enviar mais máquinas virtuais do que o nó computacional está dimensionado à suportar. Por exemplo, o nó *w* tem capacidade para executar quatro máquinas virtuais e já possui três em execução, restando apenas disponibilidade para mais uma. Em de-

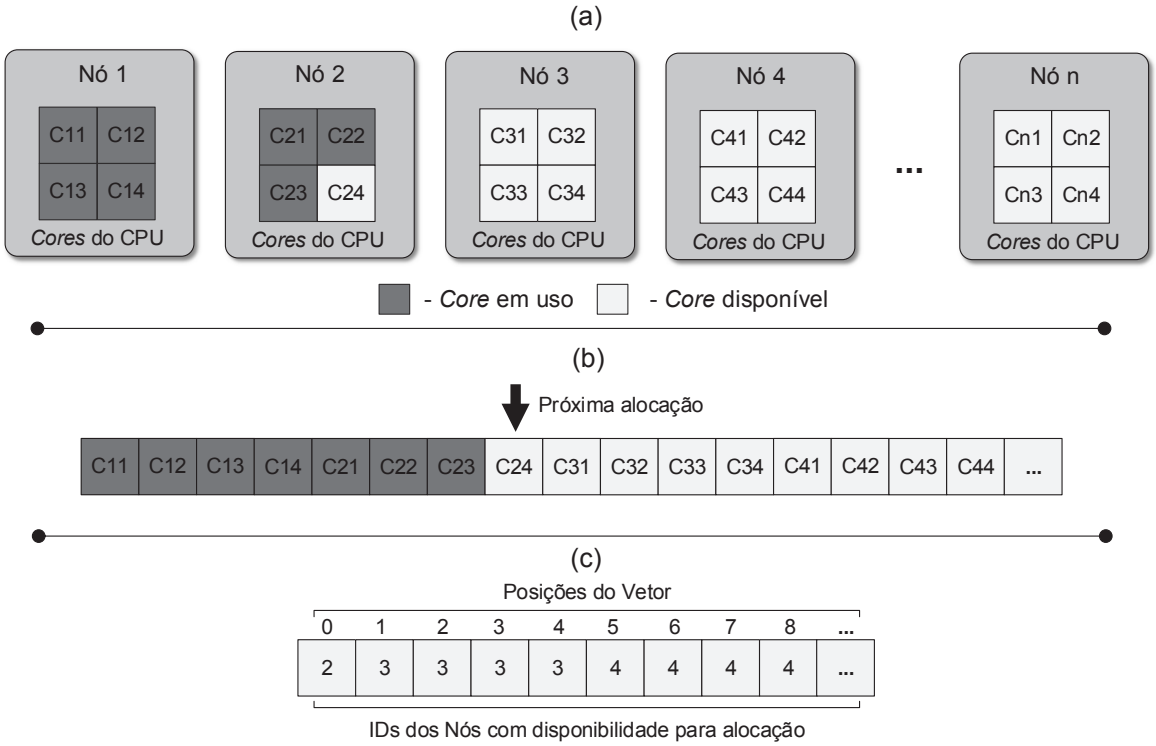
terminado instante da aplicação os estágios 1 e 2 enviam cada um mais uma MV para alocação no mesmo instante (ou em instantes muito próximos) de tempo. Pelo algoritmo sequencial, o nó *w* executaria cinco máquinas virtuais, alocando mais recursos do que deveria, não respeitando o acordo de nível de serviço e não utilizando o *hardware* da maneira mais adequada. Dessa maneira, no presente estudo será utilizado o método chamado de Alocação Aleatória de Recursos (AAR). Este método tem como objetivo evitar a má utilização do *hardware* proveniente da alocação indevida de recursos. O pseudocódigo 2 mostra sua lógica, onde o laço de repetição analisa todos os nós computacionais. Caso haja disponibilidade de instanciar uma máquina virtual no nó atual da análise, o algoritmo aloca o ID do nó em uma posição do Vetor. A verificação de disponibilidade nos nós é feita através da Equação 4.4. Para cada possibilidade de alocação de MV nos nós, é alocado o ID do nó em uma posição do Vetor. Ao término do laço de repetição será possível saber a quantidade de máquinas virtuais que serão possíveis de instanciar e em quais nós. Por fim o algoritmo cria uma número randômico de 0 até o último valor de posição do vetor e aloca a máquina virtual no valor correspondente a essa posição (ID do nó).

**Algoritmo 2:** Pseudocódigo da alocação aleatória de recursos na estrutura de nuvem computacional

```
1  início
2      vetor aloca[];
3      int x=1;
4      int y=0;
5      enquanto (x <= quantidadeNós) faça
6          se (DAR(x)>=1) então
7              int z=1;
8              enquanto (z <= DAR(x)) faça
9                  aloca[y]= idNó(x);
10                 y++;
11                 z++;
12             fim
13         fim
14         x++;
15     fim
16     x = geraNúmeroAleatório(0,y);
17     alocarMV(aloca[x]);
18 fim
```

A Figura 13 ilustra a diferença entre a estratégia de Alocação Sequencial de Recursos e a Alocação Aleatória de Recursos. A Figura 13 (a) apresenta um exemplo de infraestrutura de nuvem homogênea, onde existem diversos nós computacionais. Todos os nós possuem quatro *cores* e conforme citado anteriormente, cada *core* corresponde à alocação de uma máquina

**Figura 13:** Diferença entre a estratégia de Alocação Sequencial de Recursos (b) e a Alocação Aleatória de Recursos (c), na infraestrutura de nuvem computacional (a).




Fonte: Desenvolvido pelo autor.

virtual. Podendo executar quatro máquinas virtuais em cada nó. O nó 1 está completamente utilizado e o nó 2 possui apenas mais uma possibilidade de alocação (três *cores* estão em uso). Os demais nós estão disponíveis para receber alocação de recursos. Utilizando a Alocação Sequencial de Recursos, conforme Figura 13 (b), a próxima alocação seria no nó 2 (*Core* C24), o qual possui apenas mais umas disponibilidade de alocação. Caso, no mesmo instante de tempo, mais de um estágio precise alocar recursos na infraestrutura da nuvem, o nó 2 receberia mais máquinas virtuais que consegue executar de maneira adequada, de acordo com acordo de nível de serviço. Fazendo com que as MVs deste nó não tenham uma execução adequada, atrasando o funcionamento planejado da aplicação. Com a utilização da Alocação Aleatória de Recursos, de acordo com a Figura 13 (c), são analisadas todas as possibilidades de alocação de recursos e os IDs dos nós com disponibilidade para alocação são inseridos em um Vetor. Após isso é gerado um número randômico de 0 (primeira posição do Vetor) até a posição máxima do Vetor. O valor da posição escolhida de maneira aleatória indica o ID do nó no qual será inserida a máquina virtual. Fazendo com que a distribuição seja aleatória e reduzindo as chances do nó 2 (com apenas uma possibilidade de alocação) receber mais de uma alocação de máquina virtual.

## 4.3 Modelo de Elasticidade

Pipel possui um Gerenciador de Elasticidade que monitora periodicamente a métrica de CPU de cada máquina virtual do tipo Trabalhador. O GE colhe valores de carga de processamento (CPU) de cada máquina virtual do tipo Trabalhador, que está executando as subtarefas de seu estágio específico, e aplica um cálculo de séries temporais, considerando também valores coletados anteriormente para adquirir a carga total, neste estudo intitulada de Carga Geral de Processamento (CGP). O monitoramento acontece através de um ciclo de repetições. A cada ciclo o GE captura informações de processamento de todos os Trabalhadores e os armazena. Após ter um número considerável de coletas dos dados aplica uma média sobre estes valores para cada estágio. De acordo com os *thresholds* superior e inferior (LORIDO-BOTRAN; MIGUEL-ALONSO; LOZANO, 2014), previamente parametrizados no SLA, Pipel avalia a necessidade de instanciar ou consolidar máquinas virtuais do tipo Trabalhadores para cada estágio.

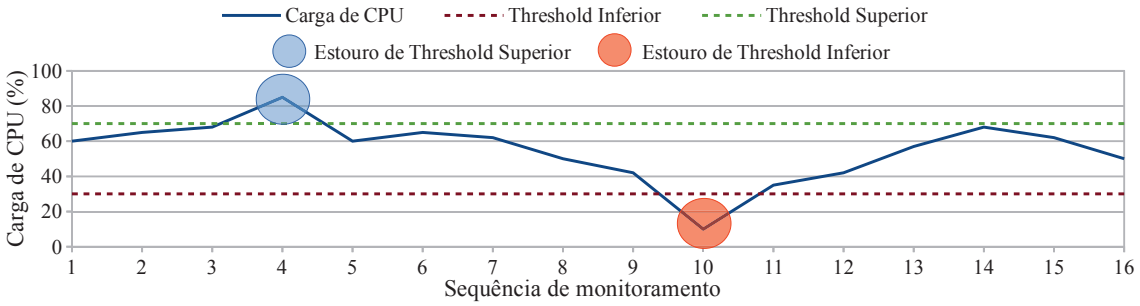
A Carga Geral de Processamento do sistema é obtida através da Equação 4.6 em que a função chamada *CGP(o,e)* calcula a média simples de todas as cargas de processamento das máquinas virtuais na observação *o* para o estágio *e*. Onde $q_e$ representa a quantidade de máquinas virtuais do tipo Trabalhadoras que estão em atividade no estágio *e* e *CPU(i,o,e)* representa a carga de CPU da(s) máquina(s) virtual(s) do tipo Trabalhador *i*, no instante *o* do estágio *e*. As cargas de CPU são obtidas através do método *Simple Exponential Smoothing* (Suavização Exponencial Simples, SES), realizando suavização exponencial simples sobre as cargas de CPU, representada por *SES(o,e)*, de todas as máquinas virtuais Trabalhadoras operantes no estágio *e*, conforme Equação 4.5. Este método é aplicado por estágio, sendo assim, a observação *o* é a mais recente mas também é considerado o histórico de observações sempre. Após obter o valor de CGP, este é utilizado para confrontar com os *thresholds* superior e inferior e disparar as ações de elasticidade.

$$SES(o,e) = \begin{cases} \frac{CPU(o,e)}{2} \text{ se } o = 0 \\ \frac{SES(o-1,e)}{2} + \frac{CPU(o,e)}{2} \text{ se } o \neq 0 \end{cases} \tag{4.5}$$

$$CGP(o,e) = \frac{\sum_{i=0}^{q_e} SES(i,o,e)}{q_e} \tag{4.6}$$

O monitoramento de Pipel opera com o conceito de *Aging* (TANENBAUM, 2008; RIGHI et al., 2015) sobre o método SES, empregando uma suavização exponencial simples, na qual a última métrica capturada tem maior influência sobre o índice de carga. O método *Aging* é utilizado com o intuito de tratar situações de picos de carga de CPU, especialmente para evitar ações resultantes de falsos-positivos ou falsos-negativos, evitando ações desnecessárias de elasticidade. Assim, Pipel evita *thrashing*, utilizando a técnica de suavização exponencial simples sobre as séries temporais com o objetivo de suavizar possíveis ruídos nas observações das cargas.

**Figura 14:** Exemplo da média da carga de CPU monitorada em dado intervalo de tempo no estágio *x*, onde ocorre duas violações de *thresholds* (superior e inferior).




Fonte: Desenvolvido pelo autor.

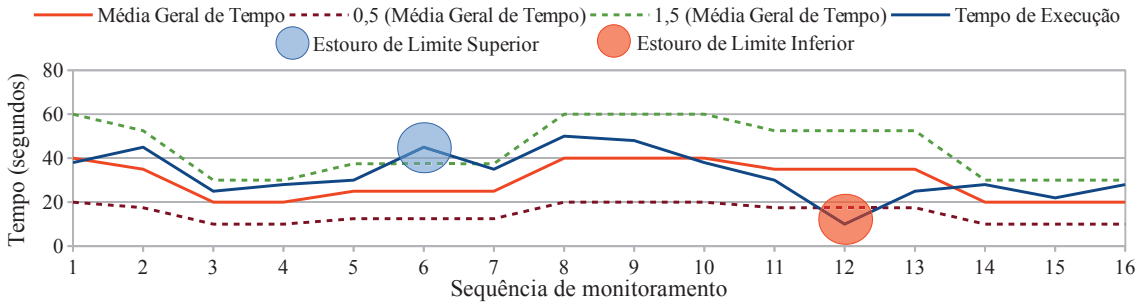
Shishir (BHARATHI et al., 2008) Rajkumar (BUYYA, 1999) afirmam que comunicação entre os estágios de um *pipeline* pode ser completamente assíncrona e que a eficiência desse modelo de paralelismo é diretamente dependente da capacidade de equilibrar a carga entre seus estágios. Além de levar em consideração as cargas de CPU o fator tempo de execução em cada estágio também é considerado para tomada de decisões elásticas. Essa abordagem tem como finalidade a redução de ociosidade dos recursos do estágio por causa de atrasos no processamento dos estágios anteriores.

### 4.3.1 *Thresholds* e Regras de Elasticidade

O uso de técnicas reativas referem-se ao conjunto de métodos que reagem com o estado do sistema atual. As decisões são tomadas com base nos últimos valores capturados a partir de um conjunto de variáveis monitoradas. O uso de *thresholds* (ou políticas de limite) seguem uma abordagem reativa. O número de máquinas virtuais no sistema varia de acordo com um conjunto de regras, geralmente duas: alocação e consolidação. Essas regras podem considerar diversas métricas de desempenho tais como: carga de CPU, taxas de solicitações ou ainda tempo médio de resposta (LORIDO-BOTRAN; MIGUEL-ALONSO; LOZANO, 2014). Pipel faz uso de duas métricas para aplicar ações de elasticidade sobre a aplicação: (i) carga de CPU de todas as máquinas virtuais do tipo Trabalhador e (ii) tempo de execução de cada estágio da aplicação.

Para o uso de *thresholds* nas cargas de CPU, existem dois limites previamente configurados no acordo de nível de serviço: inferior e superior. Caso haja violação destes limites o sistema aplica ações de elasticidade. O monitoramento é periódico e tem como parâmetro a média de carga de CPU calculada de todas as máquinas virtuais presentes em cada estágio. No início de cada ciclo de monitoramento o gerenciador de elasticidade captura todos os valores que representam a carga de CPU de todas as máquinas virtuais do tipo Trabalhador, aplicando a técnica de *Aging* sobre eles. Após isso, aplica a média desses valores para cada estágio, assim cada estágio tem seu próprio resultado, podendo haver variações entre estágios. O fato de

**Figura 15:** Exemplo de tempos de execução monitorados em dado intervalo de tempo no estágio *x*, onde ocorre duas violações de limites de tempo (superior e inferior).




Fonte: Desenvolvido pelo autor.

que cada estágio tem um resultado independente de média de processamento resulta que cada estágio é tratado de forma autônoma. A Figura 14 apresenta dois momentos de ultrapassagem de limites de carga de processamento de CPU (superior e inferior).

O Gerenciador de Elasticidade também monitora os tempos de execução de cada estágio para manter uma proporcionalidade entre eles. A comunicação entre os estágios é realizada de forma assíncrona e faz-se necessário que o GE observe cada estágio, mas também tenha um método que analise a continuidade da aplicação de maneira geral. Para isso Pipel incorpora duas condições: (i) tempo da última execução do estágio maior que a média de tempo de todos os estágios somada com metade dessa média (uma vez e meia da média geral de tempo), de acordo com a Condição 4.7; e (ii) tempo da última execução do estágio menor que metade da média geral de tempo, conforme Condição 4.8. Nas Condições 4.7 e 4.8, $t_e$ representa o tempo do estágio *e* na observação anterior (*o-1*). A média geral de tempo é obtida pelo somatório de tempo de todos os estágios, variando de *i=0* até a quantidade de estágios $q_e$, dividido pela quantidade de estágios. A Figura 15 apresenta um exemplo dessa ideia, com: a média geral de tempo de toda a aplicação *pipeline*, tempos de execução do estágio *e* e os limites de tempo. É possível perceber dois momentos de estouro de limites de tempo (superior e inferior).

$$t_e(o-1) > 1,5 \times (\frac{\sum_{i=0}^{q_e} tempo(e)}{q_e}) \tag{4.7}$$

$$t_e(o-1) < \frac{\frac{\sum_{i=0}^{q_e} tempo(e)}{q_e}}{2} \tag{4.8}$$

Tendo como premissas, a agilidade na execução da carga do *pipeline* e a utilização mínima da infraestrutura da nuvem computacional, o modelo deve balancear essas condições uma vez que a média de todos os estágios muda ao longo da execução da carga de trabalho. A lógica de cada ciclo da rotina de monitoramento da elasticidade está apresentada no Algoritmo 3. As condições e ações da rotina de monitoramento da elasticidade estão descritos na Tabela 2.

**Tabela 2:** Descrição das condições e regras utilizadas na Rotina de Monitoramento do Pipel

| Condição / Ação | Descrição |
|---|---|
| Condição1(x) | x for menor ou igual ao número de estágios da aplicação; |
| Condição2(x) | Média de CPU de todas as máquinas virtuais do tipo Trabalhadoras do estágio x for maior que o Threshold Superior; |
| Condição3(x) | Média de CPU de todas as máquina virtuais do tipo Trabalhadoras do estágio x for menor que o Threshold Inferior; |
| Condição4(x) | Tempo da última execução do estágio x for maior que a média de tempo de todos os estágios somada com metade dessa média (uma vez e meia da média); |
| Condição5(x) | Tempo da última execução do estágio x for menor que metade da média de tempo de todos os estágios; |
| Ação1(x) | Adiciona uma máquina virtual no estágio x; |
| Ação2(x) | Remove uma máquina virtual no estágio x; |

Fonte: Desenvolvido pelo autor.

**Algoritmo 3:** Pseudocódigo da Rotina de Monitoramento do Pipel

```
1  início
2      enquanto (aplicação pipeline estiver executando) faça
3          x=1;
4          enquanto (Condição1(x)) faça
5              coleta_informações_monitoramento();
6              se (Condição2(x) OU Condição4(x)) então
7                  Ação1(x);
8              fim
9              se (Condição3(x) E Condição5(x)) então
10                 Ação2(x);
11             fim
12             x++;
13         fim
14     fim
15 fim
```

## 4.4 Métricas de Avaliação

Para avaliar o desempenho da aplicação organizada em *pipeline* com o uso de elasticidade, serão abordados alguns métodos utilizados por Rosa Righi et al. (2016). Os quais fazem referência à análise de sistemas elásticos nos quesitos desempenho da aplicação e a eficiência na utilização dos recursos. Também foram utilizadas métricas para mensurar o consumo e custo dos recursos utilizados em ambiente de nuvem computacional para a execução da aplicação

*pipline*.

### 4.4.1 *Speedup* e Eficiência

Em aplicações computacionais existe uma métrica chamada de *Speedup* (S). Se trata de uma medida de desempenho utilizada para calcular ganho de tempo entre diferentes execuções de uma determinada aplicação (HENNESSY; PATTERSON, 2011). Normalmente, este cálculo é utilizado para realizar a comparação entre o tempo de execução de uma aplicação em dois cenários: (i) com apenas um processo, de forma sequencial, e (ii) com mais de um processo, representando uma execução paralela. Dito isto, a Equação 4.9 apresenta o *speedup* (S) como uma função do número de processos *p*, os quais *t*(1) e *t*(p) demonstram os tempos de execução sequencial e paralelo, respectivamente. É importante perceber que o *speedup* é uma medida livre de unidades. Além dessa métrica, é possível medir também a eficiência da execução de uma aplicação paralela (KUMMAR, 2002). A Equação 4.10 denota a eficiência de um sistema paralelo, demonstrando a fração de tempo que é utilizada pelos processadores *(*p) para uma determinada computação. Em alguns casos, eficiência é mais pertinente que *speedup* em função de que representa uma maneira mais simples de observar o quanto melhor o sistema paralelo está executando.

$$S(p) = \frac{t(1)}{t(p)} \tag{4.9}$$

$$E(p) = \frac{S(p)}{p} \tag{4.10}$$

Tendo como objetivo observar o ganho da elasticidade em nuvem computacional para aplicações organizadas em *pipeline*, serão utilizadas duas métricas como extensão dos conceitos de *speedup* e eficiência: *Speedup* Elástico (SE) e Eficiência Elástica (EE) (ROSA RIGHI et al., 2016). Estas métricas são exploradas de acordo com a elasticidade horizontal, em que instâncias de máquinas virtuais podem ser adicionadas ou consolidadas durante o processamento da aplicação, alterando a quantidade de processos (CPUs) disponíveis. Para avaliar o sistema, é considerado um ambiente homogêneo em que cada instância de máquina virtual consiga executar em 100% de um *core* de processamento do nó computacional. SE é calculado pela função SE(*q*, *i*, *s*) conforme Equação 4.11, na qual *q* representa a quantidade inicial de máquinas virtuais enquanto *s* e *i* representam os limites superior e inferior para a quantidade de máquinas virtuais definidos pelo SLA. $t_{ne}$ e $t_e$ fazem referência aos tempos de execução da aplicação executada em cenários com elasticidade e sem elasticidade. Dessa forma, $t_{ne}$ é interpretado com o menor número de máquinas virtuais possíveis (*i*).

$$SE(q, i, s) = \frac{t_{ne}(i)}{t_e(q, i, s)} \tag{4.11}$$

A função EE(*q*, *i*, *s*), representada pela Equação 4.12, faz o cálculo da eficiência elástica. Os

parâmetros dessa função são os mesmo da função SE. A eficiência representa o quanto eficaz é o uso dos recursos, sendo este posicionado como denominador da fórmula. Diferente da Equação 4.10, os recursos nesta são flexíveis e por causa disso foi criado um mecanismo para alcançar um único valor de forma coerente. Para isso, EE assume a execução de um sistema de monitoramento que captura o tempo gasto em cada configuração de máquinas virtuais. A Equação 4.13 apresenta a métrica *Recursos* utilizada no cálculo de EE indicando o uso de recursos da aplicação em que $pt_e(j)$ é o intervalo de tempo em que a aplicação foi executada com $j$ máquinas virtuais. Caso não ocorram operações de elasticidade, $pt_e(j)$ e $t_e(q, i, s)$ resultarão no mesmo valor para $j$=$q$. A Equação 4.12 apresenta o parâmetro $q$ no numerador, que está fazendo a multiplicação com o *speedup* elástico. Ao contrário de E($p$) em que o menor número de recursos é 1 (execução sequencial) servindo como base para comparação, neste caso esse valor é igual a $q$. Isso acontece pois SE($q$, $q$, $q$) é igual a 1 e *Recursos*($q$, $q$, $q$) é igual a $q$, sendo assim $q$ no numerador retorna uma eficiência elástica de 100%.

$$EE(q, i, s) = \frac{SE(q, i, s) \times q}{Recursos(q, i, s)} \tag{4.12}$$

$$Recursos(q, i, s) = \sum_{j=i}^{s} (j \times \frac{pt_e(j)}{t_e(q, i, s)}) \tag{4.13}$$

### 4.4.2 Modelo de Energia e Custo

Para complementar a análise de consumo de recursos, não somente a Eficiência Elástica será considerada mas também uma forma mais fácil de mensurar os recursos consumidos durante a execução da aplicação, chamada de *Energia*. Essa métrica é baseada na relação próxima entre consumo energético e consumo de recursos realizada por Orgerie, Assuncao e Lefevre (2014). A Equação 4.14 o índice da utilização de recursos, com $i$ e $s$ representando também a menor e maior quantidade de máquinas virtuais permitida. Neste trabalho é empregada a mesma ideia do modelo utilizado pela Amazon e Microsoft: é considerada a quantidade de máquinas virtuais em cada unidade de tempo, que no caso dessas empresas é de normalmente uma hora. A variável $pt_e(j)$ representa o intervalo de tempo em que a aplicação foi executada com $j$ instâncias de máquinas virtuais. Dessa maneira a unidade de tempo depende do valor de $pt_e$ (segundos, minutos ou horas) em que a estratégia é somar a quantidade de máquinas virtuais utilizadas em cada unidade de tempo. Sendo assim, a *Energia* mensura o uso de $i$ até $s$ instâncias de máquinas virtuais, considerando o tempo de execução parcial em cada organização dos recursos utilizados. O uso de energia se torna pertinente para realizar comparações entre diferentes aplicações que utilizam elasticidade variando de $i$ até $s$.

$$Energia(i, s) = \sum_{j=l}^{s} (j \times pt_e(j)) \tag{4.14}$$

Com o intuito de estimar a viabilidade da elasticidade em diversas situações, foi adotada uma métrica que calcula o custo da execução através da multiplicação do tempo total de execução da aplicação organizada em *pipeline* pela energia utilizada para a execução, de acordo com a Equação 4.15. Esta ideia é uma adaptação do custo de uma computação paralela, com uso em cenários elásticos. Alguns autores como Rosa Righi et al. (2016) e Baliga et al. (2011) comentam que o consumo de energia é proporcional ao uso de recursos. O objetivo é de obter um custo menor na utilização da elasticidade em comparação com a execução da aplicação em ambientes com recursos estáticos (fixos). Resumindo, uma configuração pode ser considerada ruim, se é capaz de reduzir o tempo de execução total pela metade do *pipeline* com elasticidade, mas gasta seis vezes mais, elevando os custos.

$$Custo = Tempo\_pipeline \times Energia \tag{4.15}$$

$$Custo_{\text{ce}} \leq Custo_{\text{se}} \tag{4.16}$$

Dessa forma, considerando os valores da função de custo em ambientes elásticos e não elásticos, o objetivo é preservar a verdade da Desigualdade 4.16. O $Custo_{ce}$ representa o custo da execução do *pipeline* com uso de elasticidade e $Custo_{se}$ o custo da execução com recursos fixos (sem elasticidade).

# 5 METODOLOGIA

Este capítulo apresenta a metodologia adotada para avaliar o modelo Pipel, o protótipo desenvolvido e também os recursos de ambiente de nuvem computacional. Para realizar a avaliação do modelo, foi desenvolvida uma aplicação organizada em *pipeline* com três estágios definidos e diferentes configurações de comportamentos de carga. Não só o tempo da aplicação será avaliado neste capítulo mas também o modelo de energia e custo com o uso de elasticidade reativa. A Seção 5.1 apresenta como o protótipo foi desenvolvido e como está configurada a infraestrutura de nuvem computacional. A Seção 5.2 explica o funcionamento da aplicação organizada em *pipeline*. Por fim, a Seção 5.3 descreve alguns parâmetros utilizados nos testes e em quais cenários foram executados.

## 5.1 Protótipo e Ambiente Computacional

Para realizar os testes em ambiente de nuvem computacional, foi desenvolvido um protótipo com o uso de Java para o modelo Pipel. Este protótipo utilizou as técnicas de *thresholds* para elasticidade reativa, a técnica de *Aging* para suavização de picos de cargas e também a elasticidade assíncrona. Por fim, foi utilizado um ambiente de nuvem privada para a execução dos experimentos.

### 5.1.1 Desenvolvimento do Protótipo

Para a realização da avaliação do modelo Pipel e ter uma análise geral das funcionalidades foi desenvolvido um protótipo do gerenciador de elasticidade. O *middleware* utilizado como controlador de nuvem foi o OpenNebula[1]. No desenvolvimento, a linguagem de programação Java foi utilizada devido a compatibilidade com o protocolo XML-RPC[2] incorporado pelo *middleware* Opennebula. XML-RPC é uma API que permite a interoperabilidade entre *softwares* executados em diferentes sistemas operacionais, rodando em ambientes diferentes para fazer chamadas de procedimento através do protocolo HTTP (*Hypertext Transfer Protocol*, Protocolo de Transferência de Hipertexto). Esta API faz uso de requisições do gerenciador de elasticidade para o controlador da nuvem por meio de arquivos XMLs. Através dessas requisições o gerenciador de elasticidade do Pipel consegue: (i) solicitar informações do estado atual do sistema para a rotina de monitoramento e (ii) adicionar ou remover recursos (máquinas virtuais) nos estágios desejados, de acordo com as condições e regras estabelecidas no SLA.

O *middleware* dispõe de uma interface gráfica intitulada de OpenNebula-Sunstone[3]. Essa interface é destinada à administradores de nuvem e tem o objetivo de simplificar operações de gestão da infraestrutura. Com ela foi possível adicionar os nós computacionais no sistema,

[1] www.opennebula.org
[2] http://xmlrpc.scripting.com/
[3] http://archives.opennebula.org/documentation:archives:rel2.2:sunstone

carregar as imagens do tipo Controlador de Estágio e do tipo Trabalhador, configurar a rede entre os computadores (físicos e virtuais) e liberar as devidas permissões de acesso entre todos os agentes do sistema.

Para o funcionamento de Pipel, se faz necessário o uso de uma região de dados compartilhada utilizando NFS (*Network File System*, Sistema de Arquivos de Rede). Esta área de dados fica disponível para que o gerenciador de elasticidade e os controladores de estágios consigam se comunicar. Também tem o objetivo de armazenar informações do histórico da execução da aplicação como: tempos de execuções dos estágios, quais máquinas virtuais estão alocadas para cada estágio, dados da aplicação e *logs* de monitoramento. Nesta região de dados ainda ficam informações sobre o SLA como: quantidade mínima/máxima de máquinas virtuais em cada estágio no início da aplicação, valores dos *thresholds* e intervalo de monitoramento. Caso nenhum parâmetro desses sejam setados no início da execução da aplicação, são utilizados valores padrões: (i) uma máquina virtual do tipo Trabalhador por estágio, *threshold* inferior sendo 30 e superior sendo 70 e o intervalo de monitoramento a cada 5 segundos.

### 5.1.2 Ambiente de Nuvem

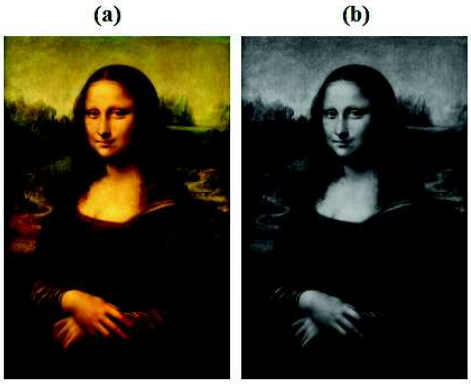
O modelo proposto foi executado no laboratório 10 da Escola Estadual de Ensino Profissional de Estrela localizada no município de Estrela (Rio Grande so Sul, Brasil). Este laboratório possui computadores com processadores *core i5* (dois núcleos físicos e quatro virtuais) e 2GB de memória *ram*. Todos os computadores têm a mesma configuração, sendo assim considerado um ambiente homogêneo de computação. Para a realização dos testes foram selecionados 8 computadores. O primeiro é o controlador da nuvem (*Front-End*), e segundo é o gerenciador de elasticidade e os outros 6 ficaram disponíveis para executar a aplicação. Dessa maneira, a aplicação deteve a capacidade de executar com até 24 instâncias de máquinas virtuais. Lembrando que para a execução mínima da aplicação *pipeline* é necessário uma máquina virtual do tipo Controlador de Estágio e pelo menos uma do tipo Trabalhador para cada estágio, totalizando 6 máquinas virtuais para sua execução mínima (sequencial) e com a capacidade de estender para mais 18 máquinas virtuais (com elasticidade). Todos os computadores (nós) são interconectados com uma rede *Fast Ethernet* (100MBit/s). O *middleware* utilizado foi o OpenNebula na versão 4.14[4] (OPENNEBULA, 2015) sobre o sistema operacional Ubuntu (64 *bits*) na versão 14.04[5].

## 5.2 Aplicação *Pipeline*

Para validar o modelo Pipel será utilizada uma aplicação organizada em *pipeline* que tem como finalidade processamento de grandes quantidades de imagens. Essa aplicação segue a

[4]http://opennebula.org/software/release/
[5]http://releases.ubuntu.com

**Figura 16:** Exemplo de processamento de imagem executado no estágio 1: (a) imagem original [quadro Mona Lisa de Leonardo da Vinci]; (b) imagem processada com tons de cinza.




Fonte: Desenvolvido pelo autor.

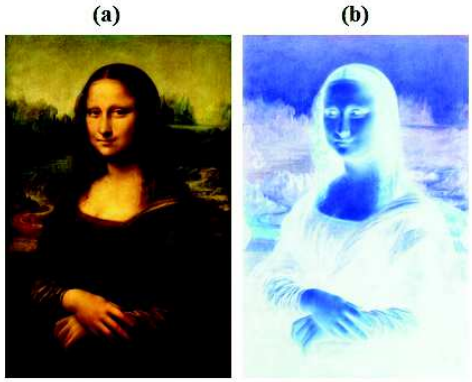
ideia do trabalho de Li et al. (2010) que tem como objetivo processamento sequencial de imagens extraídas de satélites em grandes volumes. Li et al. (2010) e seu grupo de pesquisa fazem o carregamento dos dados a partir de repositórios específicos, reprojetam as imagens em diferentes resoluções, executam algoritmos para buscar variáveis de relevância científica e gerar tabelas com todas informação, por fim entregam essas imagens e seus resultados para os cientistas analisarem.

No caso deste estudo foi criada uma aplicação similar, com três estágios de processamento de imagens definidos:

- **Tons de cinza** (Estágio 1): Uma cor no modelo de cores RGB pode ser descrita pela indicação da quantidade de vermelho, verde e azul (RGB, *reb, blue, green*) que contém. Cada uma pode variar entre o mínimo (0) e máximo (255) criando uma combinação de três valores, estes resultando em uma cor final. Quando todas as cores estão no mínimo (0,0,0), o resultado é preto. Se todas estão no máximo, o resultado é branco (255,255,255) (NISHIDATE et al., 2008). Alguns outros exemplos: (i)Azul - RGB (0,0,255); (ii) Vermelho - RGB (255,0,0); (iii) Verde - RGB (0,255,0); e (iv) Amarelo - RGB (255,255,0). A imagem que passa por essa etapa é analisada *pixel* a *pixel* e os valores de R, G e B têm seus valores alterado para o mesmo. Exemplo: (45,45,45). Dessa maneira ela é considerada em tons de cinza (PINHO, 2016). A Figura 16 mostra um exemplo do processamento executado neste estágio;

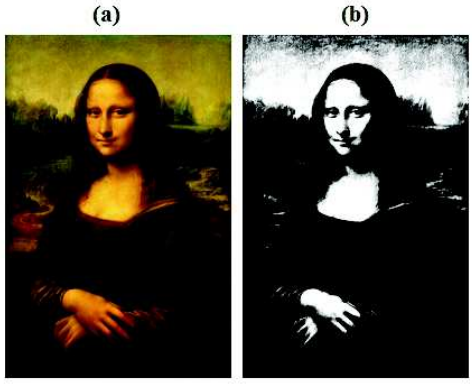
- **Inversão de cores** (Estágio 2): É uma função de mapeamento linear inversa, ou seja, o contraste ocorre de modo que as áreas escuras (baixos valores de nível de cinza) tornam-se claras (altos valores de nível de cinza) e vice-versa. O negativo de uma imagem nas bandas R, G e B da imagem introduz alterações intensas de matizes, com a substituição das cores por suas complementares (verde por magenta, azul por amarelo, vermelho por cíano, branco por preto)(BATISTA, 2016). A Figura 17 apresenta um exemplo do processamento executado neste estágio;

**Figura 17:** Exemplo de processamento de imagem executado no estágio 2: (a) imagem original [quadro Mona Lisa de Leonardo da Vinci]; (b) imagem processada com inversão de cores.




Fonte: Desenvolvido pelo autor.

**Figura 18:** Exemplo de processamento de imagem executado no estágio 3: (a) imagem original [quadro Mona Lisa de Leonardo da Vinci]; (b) imagem processada com limiarização.

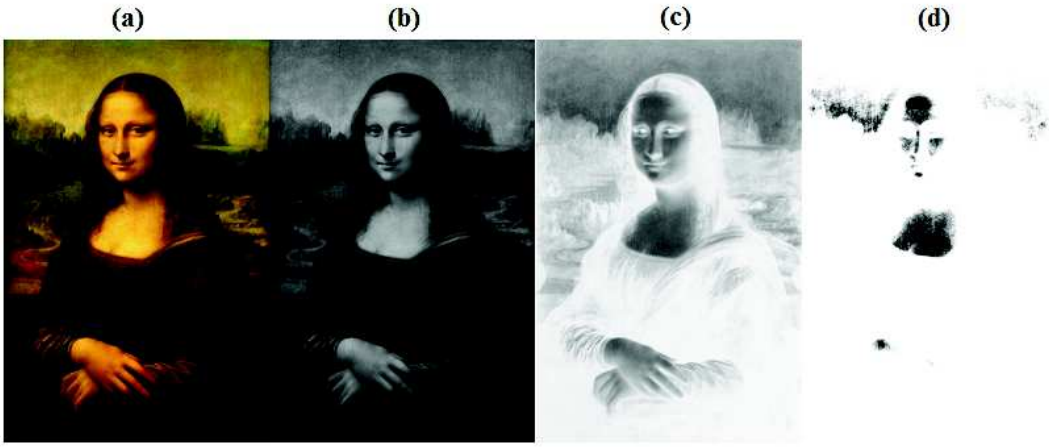



Fonte: Desenvolvido pelo autor.

- **Limiarização** (Estágio 3): A imagem nesta etapa tem cada ponto (RBG) analisado. É determinado um limiar (limite) entre 0 e 255, aqui foi utilizado o valor de 150. Cada ponto é verificado, se estiver abaixo desse limiar é alterado para (0,0,0) (preto) e se estiver acima é alterado para (255,255,2550) (branco). Assim, alterando a imagem para branco e preto a partir de um parâmetro de análise (limiar) (PINHO, 2016). A Figura 18 demonstra um exemplo do processamento executado neste estágio.

A aplicação, neste caso, utiliza os dados (imagens) de forma sequencial nos três estágios. Isso quer dizer que a imagem passa pelo processamento do primeiro estágio, após isso o resultado desta etapa serve de entrada para o segundo estágio e por fim o resultado do segundo serve de entrada para o terceiro estágio produzindo o resultado geral da aplicação *pipeline*. Dessa maneira a imagem original passa pelos três estágios do *pipeline* sofrendo alterações sequenciais e produzindo um resultado (imagem final). A Figura 19 apresenta o processamento total de uma imagem, onde é apresentada imagem original (Figura 19(a)) e suas subsequentes transformações (Figura 19(b), Figura 19(c) e Figura 19(d)).

**Figura 19:** Visão geral do processamento gáfico gerado pela aplicação *pipeline*: (a) imagem original [quadro Mona Lisa de Leonardo da Vinci]; (b) imagem processada com tons de cinza ; (c) imagem processada com inversão de cores; (d) imagem processada com limiarização.



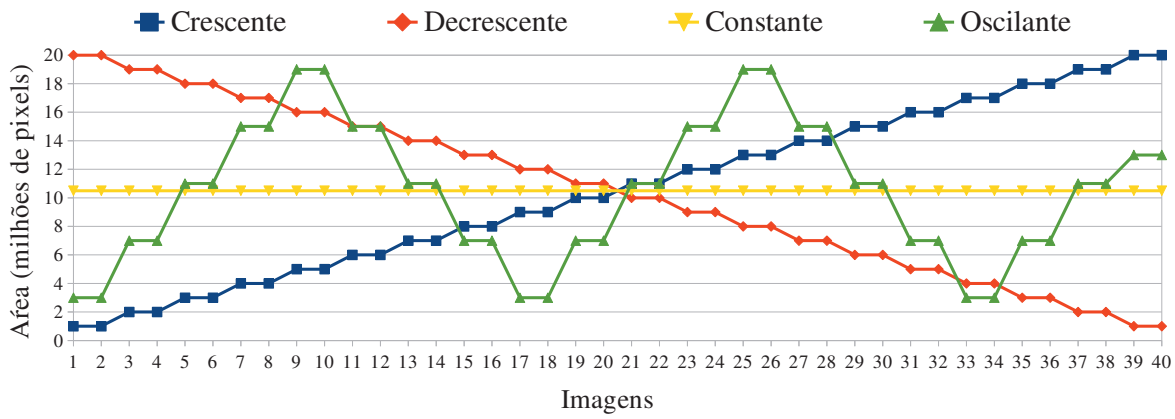

Fonte: Desenvolvido pelo autor.

Com o intuito de avaliar o desempenho do modelo Pipel, foi executado um fluxo de trabalho de processamento de imagens. Onde cada estágio faz um processamento diferente. Considerando que o processamento aplicado sobre as imagens é de paralelismo de tarefas, é possível distribuir o processamento de tarefas entre as máquinas virtuais do tipo Trabalhador. O fluxo de entrada para aplicações que utilizam padrões *pipeline* pode ser intenso, inconstante ou irregular. Para testar o sistema em diversos cenário, foram executadas cargas de processamento com diferentes intensidades: Crescente, Decrescente, Constante e Oscilante. Para isso foram utilizadas quarenta imagens em cada carga de dados. Para criar as variações de carga, o tamanho da área das imagens foi distribuído de maneira que ao longo de sua execução force o sistema a ter diferentes configurações de processamento, forçando as diferentes reconfigurações de elasticidade de acordo com o monitoramento. A Figura 20 mostra distribuição das quarenta imagens imagens e suas diferentes configurações.

O somatório de todas as imagens em cada configuração de carga resulta no mesmo valor (420.000.000 de *pixels*), fazendo com que em todos testes, a quantidade de dados processados pela aplicação seja a mesma. Existem sites como HUBBLE[6] e NASA[7] que disponibilizam imagens de telescópio e de satélites em alta resolução. Pode-se considerar que nos testes realizados pela aplicação *pipeline* foram processados aproximadamente cinco imagens com área de 81.000.000 *pixels* (alta resolução) retiradas de um desses sites, anteriormente citados. A Tabela 3 mostra todas as configurações das cargas, o tamanho da área de cada imagem processada e o somatório das aréas de todas as imagens em cada cenário.

[6]http://hubblesite.org/gallery/album/
[7]http://visibleearth.nasa.gov/

**Figura 20:** Diferentes configurações de cargas de dados.




Fonte: Desenvolvido pelo autor.

**Tabela 3:** Diferentes configurações de cargas de dados com seus respectivos somatórios.

| Configuração de Carga | Crescente | Decrescente | Constante | Oscilante |
|---|---|---|---|---|
| **Área da imagem** (milhões de *pixels*) | 1 | 20 | 10,5 | 3 |
| | 1 | 20 | 10,5 | 3 |
| | 2 | 19 | 10,5 | 7 |
| | 2 | 19 | 10,5 | 7 |
| | 3 | 18 | 10,5 | 11 |
| | 3 | 18 | 10,5 | 11 |
| | 4 | 17 | 10,5 | 15 |
| | 4 | 17 | 10,5 | 15 |
| | 5 | 16 | 10,5 | 19 |
| | 5 | 16 | 10,5 | 19 |
| | 6 | 15 | 10,5 | 15 |
| | 6 | 15 | 10,5 | 15 |
| | 7 | 14 | 10,5 | 11 |
| | 7 | 14 | 10,5 | 11 |
| | 8 | 13 | 10,5 | 7 |
| | 8 | 13 | 10,5 | 7 |
| | 9 | 12 | 10,5 | 3 |
| | 9 | 12 | 10,5 | 3 |
| | 10 | 11 | 10,5 | 7 |
| | 10 | 11 | 10,5 | 7 |
| | 11 | 10 | 10,5 | 11 |
| | 11 | 10 | 10,5 | 11 |
| | 12 | 9 | 10,5 | 15 |
| | 12 | 9 | 10,5 | 15 |
| | 13 | 8 | 10,5 | 19 |
| | 13 | 8 | 10,5 | 19 |
| | 14 | 7 | 10,5 | 15 |
| | 14 | 7 | 10,5 | 15 |
| | 15 | 6 | 10,5 | 11 |
| | 15 | 6 | 10,5 | 11 |
| | 16 | 5 | 10,5 | 7 |
| | 16 | 5 | 10,5 | 7 |
| | 17 | 4 | 10,5 | 3 |
| | 17 | 4 | 10,5 | 3 |
| | 18 | 3 | 10,5 | 7 |
| | 18 | 3 | 10,5 | 7 |
| | 19 | 2 | 10,5 | 11 |
| | 19 | 2 | 10,5 | 11 |
| | 20 | 1 | 10,5 | 13 |
| | 20 | 1 | 10,5 | 13 |
| **Somatório das Áreas** (milhões de *pixels*) | **420** | **420** | **420** | **420** |

## 5.3 Cenários de Testes

O modelo Pipel tem como objetivo proporcionar uma solução que adeque os recursos disponíveis sob demanda em tempo de execução, para assim aproveitar de forma mais eficiente estes recursos para executar aplicações organizadas em *pipeline*. Tornando-se um sistema que gerencia a elasticidade de maneira horizontal, automática e reativa.

Como resultado, espera-se que o modelo desenvolvido consiga identificar pontos de sobrecarga ou de ociosidade das máquinas virtuais, através do Gerenciador de Elasticidade, e tome ações a partir de regras definidas previamente (mecanismo regra-condição-ação). Estas regras utilizam limiares pré-definidos baseados na carga atual sobre a capacidade total de CPU obtidos a partir do monitoramento. A partir desses valores executa rotinas de alocação ou consolidação de recursos quando o limites forem ultrapassados, sempre respeitando os valores máximos e mínimos também previamente definidos no SLA. Os valores escolhidos para os *thresholds* foram de 30 para inferior e 70 para superior em razão de serem utilizados comumente em aplicações paralelas distribuídas (DAWOUD; TAKOUNA; MEINEL, 2011; AL-HAIDARI; SQALLI; SALAH, 2013; RIGHI et al., 2015). O intervalo de tempo de monitoramento configurado para o controlador de nuvem será de 5 segundos, valor padrão do SLA, conforme citado na Seção 5.1. Para tornar possível os cálculos para avaliação citados na Seção 4.4, faz-se necessário executar a aplicação *pipeline* de maneira simples, sem elasticidade para conseguir confrontar com a execução com elasticidade e analisar os resultados. Assim sendo, foram criados dois cenários de testes:

- **Cenário 1** (C1.SE) - Sem Elasticidade : A aplicação é executada com todas as diferentes cargas de processamento sem a utilização das funcionalidades do Gerenciador de Elasticidade. Dessa forma, as alocações e consolidações de recursos (máquinas virtuais do tipo Trabalhador) não são habilitadas. Neste cenário utilizam-se três Controladores de Estágios (um para cada estágio) e três Trabalhadores (um para cada estágio), totalizando 6 máquinas virtuais.

- **Cenário 2** (C2.CE) - Com Elasticidade: A aplicação é executada com todas as diferentes cargas de processamento com a utilização das funcionalidades do Gerenciador de Elasticidade. Dessa forma, as alocações e consolidações de recursos (máquinas virtuais do tipo Trabalhador) encontram-se habilitadas. Neste cenário a aplicação inicia com três Controladores de Estágios (um para cada estágio) e três Trabalhadores (um para cada estágio), totalizando 6 máquinas virtuais. Podendo expandir para mais 18 máquinas virtuais (limite físico) do tipo Trabalhador, de acordo com a necessidade.

Cada comportamento de carga deve ser testado nos dois cenários, podendo obter as métricas de execução da aplicação com o objetivo de comparar uma execução não elástica com um elástica.

Conforme citado no Capítulo 4, na Subseção 4.2.3, o método de alocação de recursos utilizado no modelo Pipel é de Alocação Aleatória de Recursos. Todos os testes que envolvem elasticidade utilizarão este método. Entretanto será feita uma análise comparativa entre os dois métodos mencionados: Alocação Aleatória de Recursos e Alocação Sequencial de Recursos.

# 6 RESULTADOS

Este capítulo apresenta os resultados provenientes dos experimentos realizados nos dois cenários de testes e com as quatro configurações de carga de processamento diferentes: Carga Crescente, Carga Decrescente, Carga Constante e Carga Oscilante. Resultando assim, execuções em variadas situações em diversas configurações de cargas. Este capítulo está organizado em cinco seções. Na Seção 6.1 é realizada uma análise das métricas de Tempo, Energia e Custo. Na Seção 6.2 são avaliadas as métricas *Speedup* e Eficiência com o uso de elasticidade para aplicação *pipeline*. Na Seção 6.3 é realizado um mapeamento da utilização dos recursos nos diversos cenários e as variações de alocação de máquinas virtuais nos estágios da aplicação. Na Seção 6.4, através da métrica Energia, são feitas considerações a respeito do consumo e perfis de alocação dos recursos. Na Seção 6.5 é mostrado o impacto do emprego da técnica *Aging* no gerenciamento da elasticidade. Por fim, na Seção 6.6 é realizada uma comparação entre os métodos de alocação de recursos na infraestrutura de nuvem computacional.

## 6.1 Análise de Tempo, Energia e Custo

Para analisar as métricas Tempo, Energia e Custo, é apresentado na Tabela 4 os valores obtidos na execuções dos testes nos dois diferentes cenários: (C1.SE) execução da aplicação com a quantidade mínima de recursos e sem utilização da elasticidade; (C2.CE) execução da aplicação com utilização de recursos variáveis, ou seja, com uso de elasticidade. Os dois cenários foram executados com todas as variações de cargas de processamento descritas no Capítulo 5, Seção 5.2. Esta tabela também apresenta os resultados parciais por estágio em cada cenário. Os resultados apresentados foram calculados com as equações disponibilizados na Seção 4.4.

Cada vez que uma máquina virtual é adicionada em algum estágio, após iniciar sua atividade, aguarda o controlador de estágio incluir ela com as demais MVs do mesmo estágio. Essa fatia de tempo depende do termino da execução dos dados atuais no estágio em questão. Como a máquina virtual está ativa nesses instantes, e consequentemente "custando", foi calculada uma média desse tempo em todas as execuções realizadas. O valor calculado dessa média de tempo de inatividade por máquina virtual adicionada no sistema foi de 10 segundos. A Tabela 4 considera a Energia e Custo dessa adição de recursos nos estágios, para o cenário C2.CE. Este intervalo de tempo em que a MV é adicionada, representou um acréscimo de 0,7%, 0,7%, 0,6%, 0,8% no Custo na execução das cargas Crescente, Decrescente, Constante e Oscilante, respectivamente.

Avaliando apenas a métrica Tempo, é possível afirmar que no cenário C1.SE, a execução mais rápida foi com a configuração de carga de processamento Decrescente (1747 segundos) e que a mais demorada foi com a configuração Crescente (1870 segundos). Descendo para o nível de estágios, todos os estágios 1 foram os mais rápidos em suas configurações específicas e os estágios 3 os mais demorados, isso porque o estágio 3 fica ativo em 100% da aplicação. O

**Tabela 4:** Análise das métricas de Tempo, Energia e Custo nos quatro comportamentos de cargas, para os seguintes cenários: (C1.SE) execução da aplicação com a quantidade mínima de recursos e sem utilização da elasticidade; (C2.CE) execução da aplicação com utilização de elasticidade.

| Cenários | Estágios | Crescente | | | Decrescente | | | Constante | | | Oscilante | | |
|---|---|---|---|---|---|---|---|---|---|---|---|---|---|
| | | Tempo | Energia | Custo | Tempo | Energia | Custo | Tempo | Energia | Custo | Tempo | Energia | Custo |
| C1.SE | 1 | 1767 | 3534 | 6244 | 1715 | 3430 | 5882 | 1729 | 3458 | 5978 | 1786 | 3572 | 6379 |
| | 2 | 1817 | 3634 | 6602 | 1722 | 3444 | 5930 | 1739 | 3478 | 6048 | 1805 | 3610 | 6516 |
| | 3 | 1870 | 3740 | 6993 | 1743 | 3486 | 6076 | 1777 | 3554 | 6315 | 1829 | 3658 | 6690 |
| | **Total** | **1870** | **10908** | **20397** | **1747** | **10360** | **18057** | **1777** | **10490** | **18640** | **1829** | **10840** | **19826** |
| C2.CE | 1 | 1080 | 4120 | 4449 | 1198 | 3460 | 5208 | 1142 | 3950 | 4510 | 1109 | 4109 | 4556 |
| | 2 | 1129 | 4201 | 4742 | 1222 | 4018 | 4909 | 1177 | 3644 | 4288 | 1133 | 3575 | 4050 |
| | 3 | 1154 | 4259 | 4914 | 1328 | 4348 | 5774 | 1184 | 3572 | 4229 | 1172 | 3603 | 4222 |
| | **Total** | **1154** | **12580** | **14517** | **1328** | **12319** | **16359** | **1184** | **11166** | **13220** | **1172** | **11287** | **13228** |

Fonte: Desenvolvido pelo autor.

estágio 1 que foi mais rápido com relação às outras configurações de carga se destaca o da carga Decrescente (1715 segundos). No cenário C2.CE, a configuração com o tempo de execução mais rápido foi o da carga Crescente (1154 segundos) e o mais lento foi da carga Decrescente (1328 segundos). Assim como na configuração sem elasticidade os tempos dos estágios 1 são menores, e os dos estágios 3 são os maiores em cada configuração. O estágio 1 que teve o menor tempo de execução foi da carga Crescente (1080 segundos). Fazendo uma relação entre os dois cenários afirma-se que, os tempos de execução total da aplicação reduziram 38% (Crescente), 24% (Decrescente), 33% (Constante), e 35% (Oscilante) com o uso de elasticidade. Por fim, o estágio que teve a maior redução de tempo de execução com o uso de elasticidade em relação ao cenário sem elasticidade foi o estágio 1 da carga Crescente (38% menor) e o estágio com a menor redução foi o estágio 3 da carga Decrescente (23% menor).

Observando a métrica de Energia do cenário C2.CE em relação C1.SE, afirma-se que aumentou 15% na carga Crescente, 18% na carga Decrescente, 6% na carga Constante e 4% na carga Oscilante. No geral todas as execuções com uso de elasticidade utilizaram mais energia do que sem uso de elasticidade, com exceção dos estágios 2 e 3 da carga Oscilante, que reduziram em 1% e 1,5%, respectivamente. Isso porque o acréscimo de recursos compensou o tempo de execução nestes estágios.

Mesmo que a energia teve aumento do cenário 1 para o cenário 2, a métrica de Custo mantem conformidade com a Inequação 4.16 na Seção 4.4 do Capítulo 4, que diz que o custo com uso de elasticidade deve ser menor ou igual do que sem sua utilização. No cenário C1.SE a configuração com o menor custo é a carga Decrescente, enquanto que a configuração com o maior custo é da carga Crescente. No cenário C2.CE, a configuração que teve menor custo foi a carga Constante e a configuração com maior custo foi da carga Decrescente. Fazendo a relação do cenário C2.CE com o cenário C1.SE, pode-se afirmar que o Custo diminuiu 24% na carga Crescente, 33% na carga Decrescente, 35% na carga Constante e 33% na carga Oscilante.

**Tabela 5:** Análise das métricas de Recursos (RE), *Speedup* Elástico (SE) e Eficiência Elástica (EE) nos quatro comportamentos de cargas, para os dois cenários: (C1.SE) execução da aplicação sem uso de elasticidade; (C2.CE) execução da aplicação com utilização de elasticidade.

| Cenários | Estágios | Crescente | | | Decrescente | | | Constante | | | Oscilante | | |
|---|---|---|---|---|---|---|---|---|---|---|---|---|---|
| | | *RE* | *SE* | *EE* | *RE* | *SE* | *EE* | *RE* | *SE* | *EE* | *RE* | *SE* | *EE* |
| C1.SE | 1 | 1 | 1 | 1 | 1 | 1 | 1 | 1 | 1 | 1 | 1 | 1 | 1 |
| | 2 | 1 | 1 | 1 | 1 | 1 | 1 | 1 | 1 | 1 | 1 | 1 | 1 |
| | 3 | 1 | 1 | 1 | 1 | 1 | 1 | 1 | 1 | 1 | 1 | 1 | 1 |
| | **Total** | **3** | **1** | **1** | **3** | **1** | **1** | **3** | **1** | **1** | **3** | **1** | **1** |
| C2.CE | 1 | 3,81 | 2,68 | 0,70 | 3,63 | 2,05 | 0,56 | 3,46 | 2,29 | 0,66 | 3,71 | 2,59 | 0,70 |
| | 2 | 3,72 | 2,59 | 0,70 | 3,29 | 1,99 | 0,60 | 3,10 | 2,18 | 0,71 | 3,16 | 2,54 | 0,80 |
| | 3 | 3,69 | 2,63 | 0,71 | 3,27 | 1,72 | 0,53 | 3,02 | 2,25 | 0,75 | 3,07 | 2,44 | 0,79 |
| | **Total** | **10,9** | **2,63** | **0,72** | **9,28** | **1,73** | **0,56** | **9,43** | **2,25** | **0,72** | **9,63** | **2,44** | **0,76** |

Fonte: Desenvolvido pelo autor.

## 6.2 Análise de *Speedup* e Eficiência

A Tabela 5 apresenta a comparação das métricas de *Speedup* Elástico (SE) e Eficiência Elástica (EE). Além disto é informada a métrica de Recursos (RE), utilizada para obter os valores de EE. Conforma citado na Seção 4.4 do Capítulo 4, no cálculo de EE é informada a quantidade de máquinas virtuais com que a aplicação inicia. Pelo fato da variação de recursos ocorrer apenas nas máquinas virtuais do tipo Trabalhador e que em cada estágio a aplicação inicia com uma MV deste tipo, considera-se uma MV inicial para o cálculo de EE nos estágio e três MVs para o mesmo cálculo na aplicação total (com todos os estágios).

No cenário C1.SE todos os resultados de SE e EE resultaram no número 1. Isto porque o cálculo considera a relação entre o cenário C1.SE para o mesmo cenário, resultando esses valores. Analisando o cenário C2.CE, afirma-se que a métrica SE que obteve melhor resultado foi com a configuração de carga Crescente e o pior resultado com a configuração de carga Decrescente. Essas configurações também tiveram o melhor e pior resultado na avaliação Tempo na Tabela 4. O mesmo ocorre com a métrica EE e a métrica Energia, demonstrando relação direta entre essas métricas. De maneira geral, os resultados demonstram que um melhor desempenho equivale a uma eficiência menor e um pior desempenho equivale a uma eficiência maior. Isso acontece porque, como a aplicação utilizada é intensiva quanto ao uso de CPU, na medida em que são incluídas máquinas virtuais, essas vão ficando cada vez menos utilizados em razão da carga ser mais espalhada entre elas (maior quantidade de recursos). Entretanto, mesmo com a subutilização de recursos obtém-se desempenho porque as execuções não acontecem por uma única unidade de processamento (sequencialmente).

## 6.3 Análise do Histórico de Carga *versus* Alocação de Recursos

A Tabela 4 apresentou todos os resultados obtidos, em todos os cenários e com as diversas variações de cargas de execuções. As Figuras 21, 24, 25 e 26 mostram os comportamentos desses cenários e das diversas configurações realizadas nos experimentos. Em todos os casos

serão mostradas: (a) as execuções sem elasticidade, (b) as execuções com elasticidade e (b.1, b.2 e b.3) as execuções de cada estágio com o uso de elasticidade. No caso das execuções por estágio, são mostrados os recursos sendo adicionados ou removidos. As figuras geradas utilizam os eixos nas mesmas escalas, possibilitando a comparação visual entre cada uma das execuções. É possível perceber que, com o uso de elasticidade nas aplicações, o tempo foi menor em todos os casos do que as execuções sem a utilização de elasticidade.

É importante destacar que apesar das configurações do OpenNebula estarem marcadas para realizar o monitoramento a cada 5 segundos, o monitoramento de fato não respeita esse intervalo perfeitamente. Ou seja, o tempo de intervalo de monitoramento não é sempre igual. E ainda, é comum que o monitoramento retorne valores exatamente iguais aos anteriores, já avaliados. Para contornar isso foi adotada uma estratégia que valida o ID (chave específica de cada pacote) do pacote com as informações disponibilizadas pelo *middleware* com a finalidade de descartar a informação, caso ela já tenha sido utilizada. Dessa maneira os intervalos de monitoramento são parcialmente prejudicados. A seguir cada comportamento de carga é analisado individualmente.

### 6.3.1 Comportamento da Carga Crescente

Na Figura 21, na execução sem elasticidade (a), destacam-se os *thresholds* superior e inferior. Entretanto, neste caso, estão informados para manter o padrão das ilustrações pois não são utilizados. Esta execução é marcada por picos (para cima e para baixo) de cargas de processamento. Isso ocorre porque na aplicação *pipeline* trafegam diversos dados (imagens, neste caso). E entre cada dado, as máquinas do tipo Trabalhador, as quais são monitoradas, reduzem seu processamento quase que de forma nula. E quando estão processando elevam o nível de CPU. Ainda, ressalta-se que o primeiro estágio é o primeiro a parar sua execução, depois o segundo e por fim o terceiro. Para fins de organização, na sequência do texto, esse comportamento será chamado de “Padrão de Execução *Pipeline* sem Elasticidade”.

Ainda na mesma figura, na execução com elasticidade (b), não destacam-se todos os níveis de processamentos baixos (troca de dados dos estágios) em razão de que ao longo da execução são adicionadas mais recursos e OpenNebula aumenta um pouco o tempo entre monitoramentos. O gráfico é prejudicado porque a fatia de tempo que o dado troca de estágio é de alguns segundos. Podendo não ser capturado pelo monitoramento. Entretanto essa informação (troca de dados entre os estágios) não compromete a execução do monitoramento da aplicação *pipeline*. Nota-se que em relação à execução sem elasticidade (a), a execução com elasticidade (b) tem redução de tempo de 38% (de 1870 para 1154 segundos) e nota-se que o nível de processamento tende a ficar dentro das faixas dos *thresholds* superior e inferior ao longo da execução.

Com relação ao uso dos recursos nos estágios, percebe-se que ao longo da execução da carga Crescente apenas ocorreu a ação de alocação de recursos em todos os estágios. Isso acontece porque a carga de dados aumenta, logo, o processamento também e dessa maneira nenhuma máquina virtual é consolidada. O estágio 1 (b.1) começa com uma máquina virtual e

ao longo da execução são adicionadas mais três MVs. O mesmo ocorre no estágio 2 (b.2) e 3 (b.3). Ainda é possível identificar que no estágio 1 e estágio 3 ocorrem um estouro de *threshold* superior próximo ao fim da execução. Nestes casos as máquinas virtuais foram adicionadas ao sistema, porém não chegaram a iniciar suas execuções devido ao tempo de entrega delas pela rede e tempo de *boot* do sistema operacional.

A Figura 22 mostra os seis *hosts* com quinze máquinas virtuais alocadas, tendo a possibilidade de expandir para mais nove MVs próximo ao instante 730 segundos da execução com elasticidade (b). A Figura 23 apresenta os recursos (máquinas virtuais) utilizados, onde percebe-se que as MVs 1, 2 e 3 são os controladores de estágios, as MVs 152, 154 e 159 são as que iniciam com a aplicação, as MVs 160, 162 ,162 , 163, 164 e 167 estão executando a aplicação, as MVs 165 e 166 estão em processo de *boot* para os estágios 1 e 2 e a MV 168 está sendo enviada pela rede (*status*=PROLOG) para ser utilizada no estágio 3.

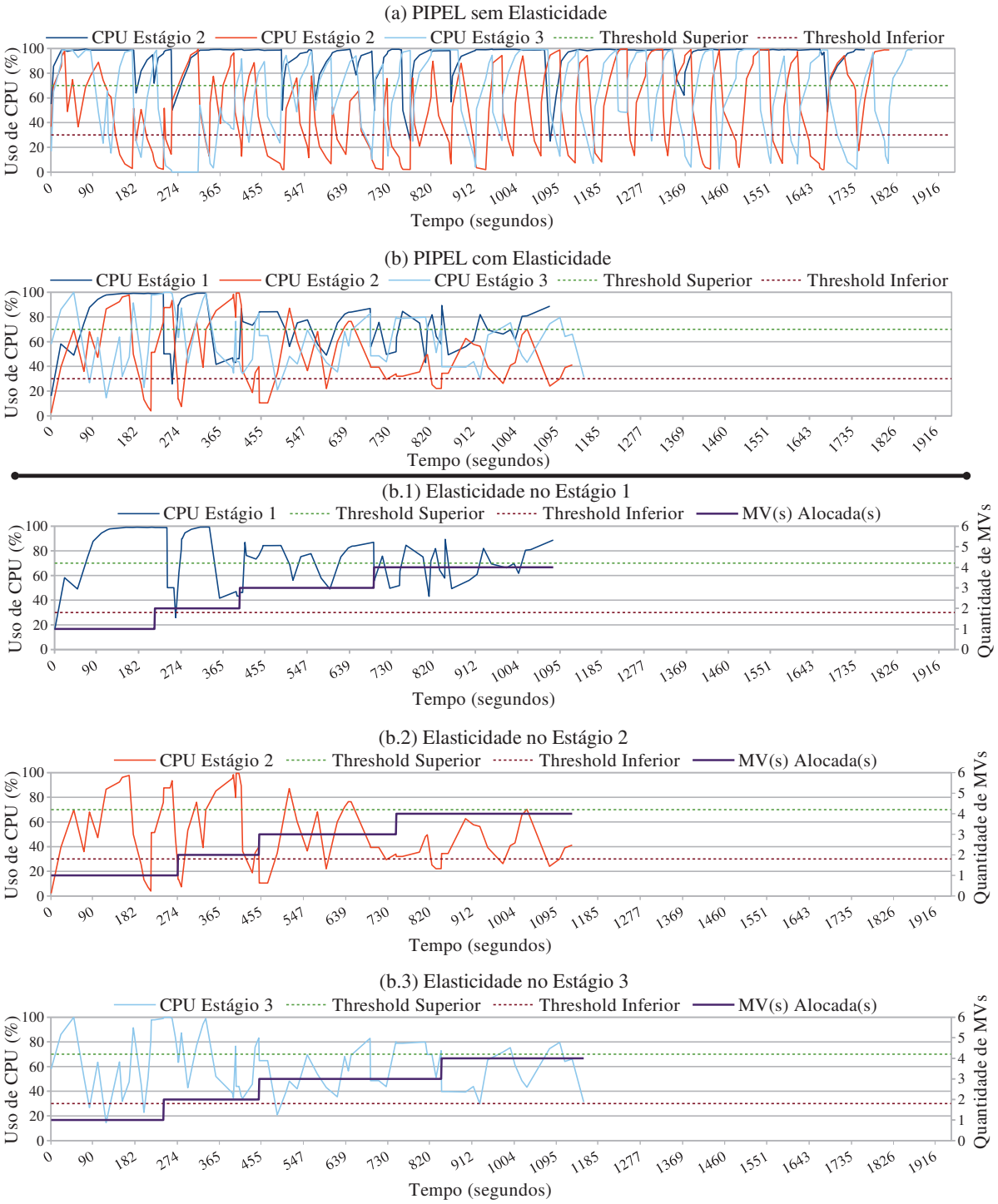
### 6.3.2 Comportamento da Carga Decrescente

Analisando a Figura 24, a execução sem elasticidade (a) segue o padrão de execução *pipeline* sem elasticidade. Este comportamento obteve o menor tempo de execução dentre as execuções sem elasticidade devido à distribuição dos dados ao longo dos estágios da aplicação. No início da execução os dados (imagens) são maiores e no fim diminuem de tamanho. Nesse caso a aplicação começa com elevados índices de processamento e portanto o gerenciador de elasticidade julga necessário alocar recursos desde o princípio da execução.

Através da mesma figura, na execução com elasticidade (b), nota-se que o tempo de execução com relação à execução sem elasticidade obteve uma redução de tempo de 24%. Dentre todos os comportamentos de carga este foi o que obteve a menor redução de tempo. Percebe-se também que é o comportamento de carga que tem maior diferença de tempo entre o início do processamento dos estágios, isso deve-se ao fato de que os dados que trafegam pelo *pipeline* começam com dados maiores e portanto gera maior tempo de processamento entre os estágios.

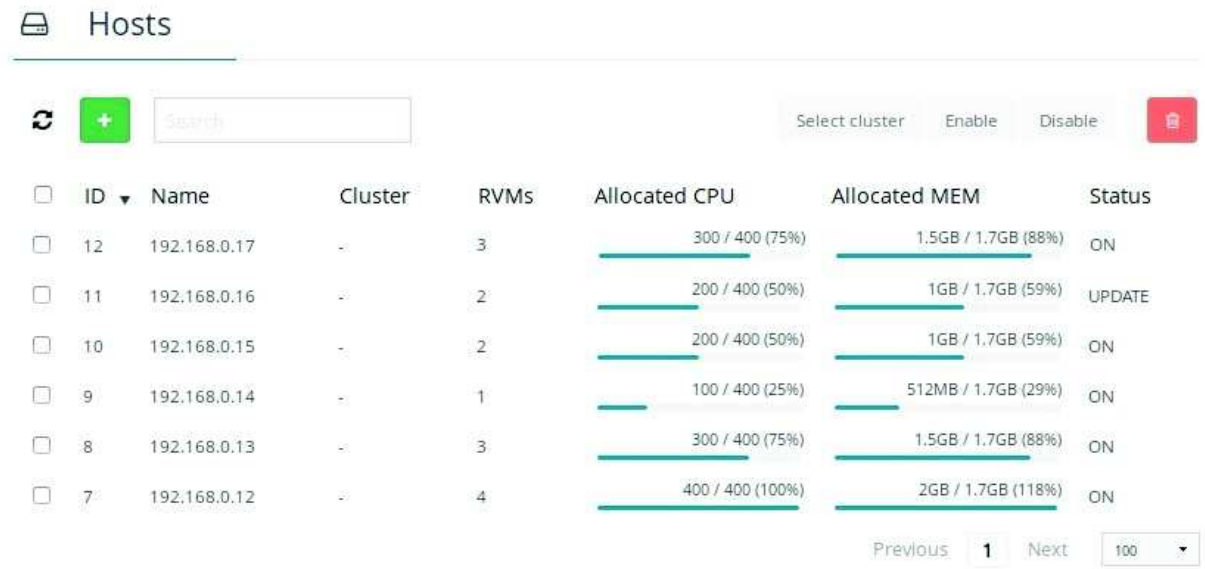
Analisando o uso dos recursos nos estágios, nota-se que no início da execução a reação do sistema é de alocação de recursos devido ao intenso uso de CPU. Entretanto, chega em um momento que os dados diminuem de tamanho e o gerenciador de elasticidade consolida recursos. Isso ocorre no estágio 1 (b.1) e no estágio 3 (b.3). No estágio 1, a consolidação ocorre no instante 826 segundos da execução. Já, no instante 876 segundos, o gerenciador de elasticidade percebe que o *threshold* superior foi ultrapassado e no instante 1076 segundos aloca novamente mais uma máquina virtual para o estágio. No estágio 3, no instante 1087 segundos e no instante 1152 segundos, a consolidação de recursos ocorre, consolidando assim duas máquinas virtuais. No Estágio 2 (b.2), ocorrem duas alocações de recursos, no instante 280 segundos e no instante 620 segundos. Por fim, no instante 1003 segundos é ultrapassado o *threshold* superior e inicia-se o processo de alocação de recursos, entretanto a aplicação finaliza antes da máquina virtual ser alocada.

**Figura 21:** Resultado da execução da carga Crescente: (a) PIPEL sem ações elásticas; (b) PIPEL com ações elásticas; Uso de CPU *versus* quantidade de máquinas virtuais alocadas para o estágio 1 (b.1), estágio 2 (b.2) e estágio 3 (b.3).



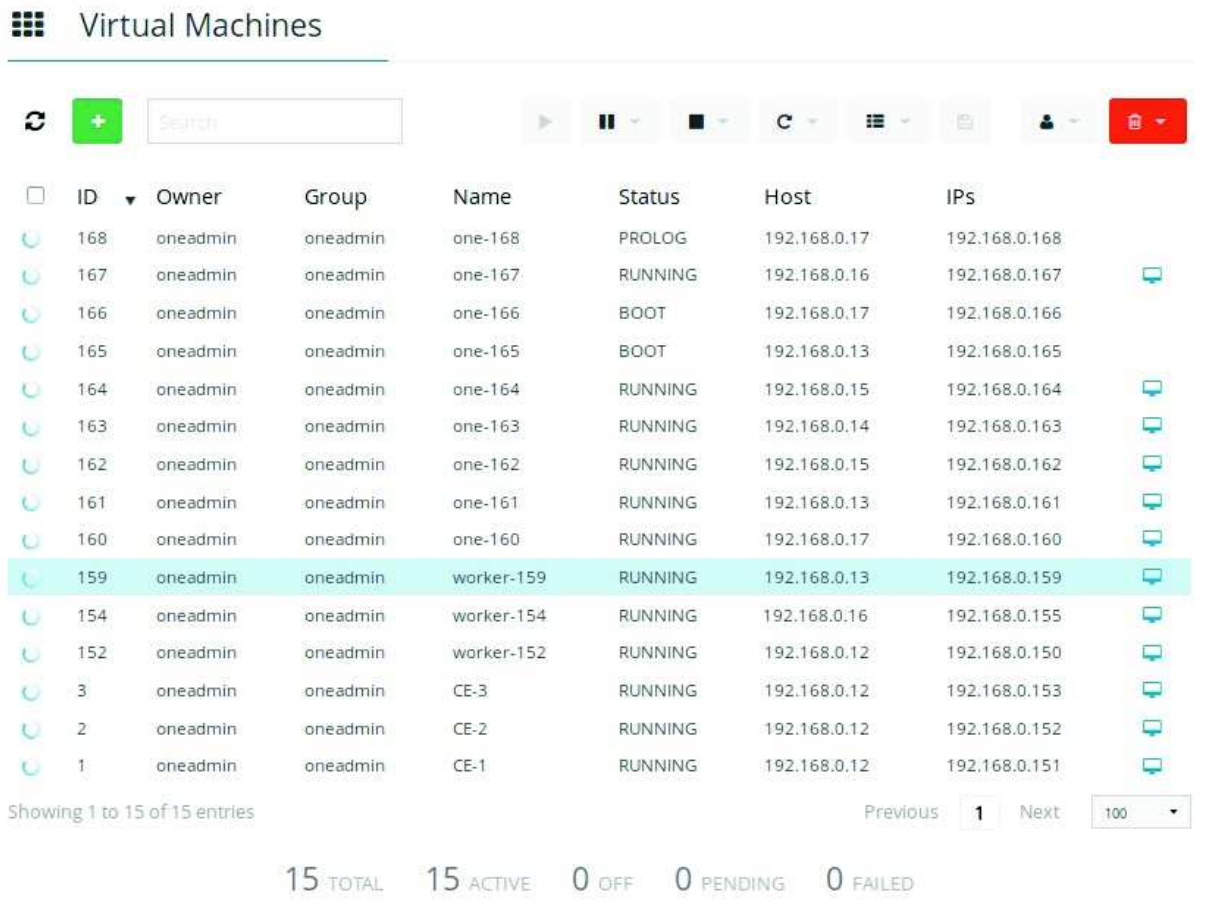

Fonte: Desenvolvido pelo autor.

**Figura 22:** Captura de tela, realizada próximo ao instante 730 segundos da carga Crescente com uso de elasticidade, de *hosts* disponíveis no OpenNebula.



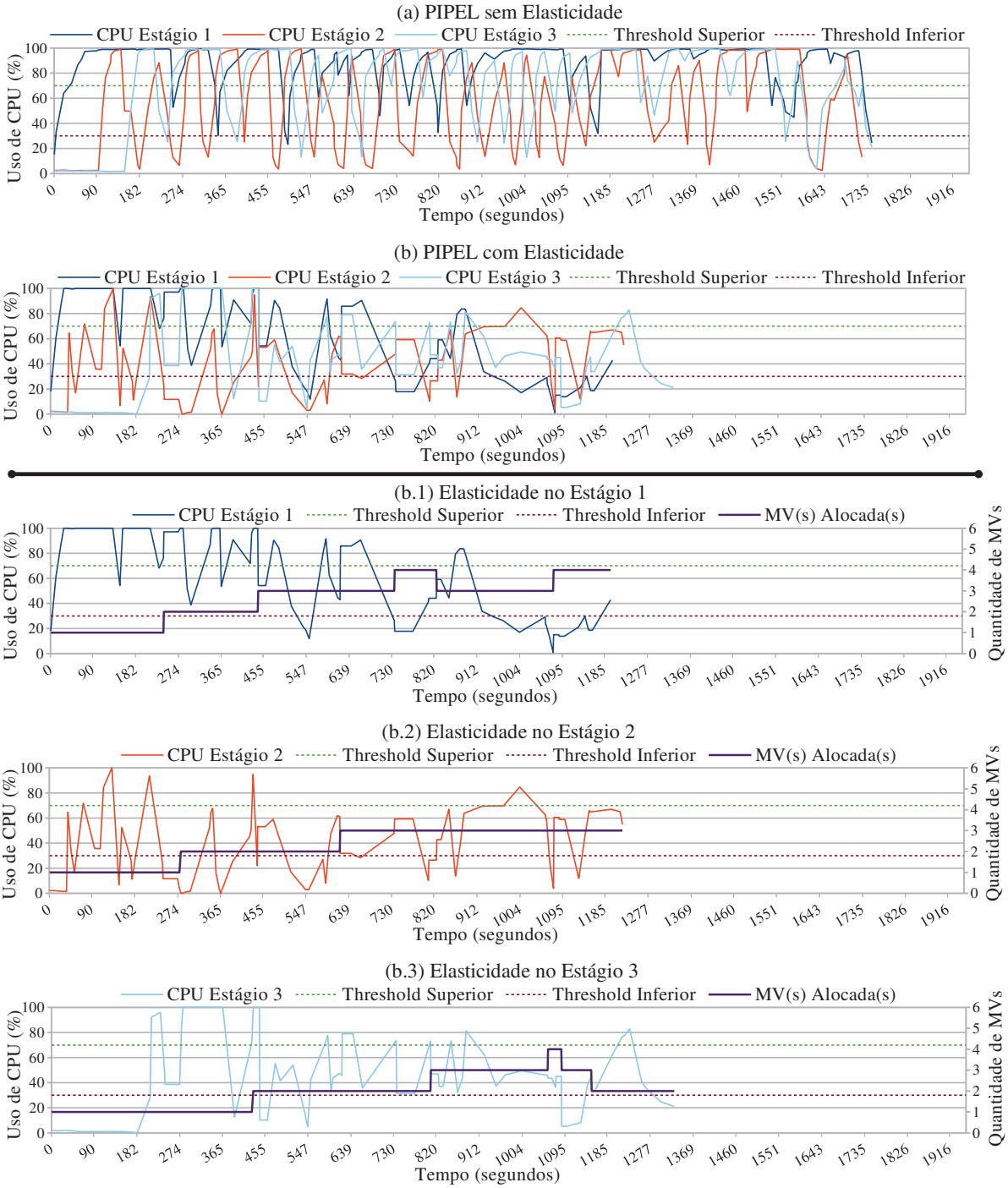

Fonte: Desenvolvido pelo autor.

**Figura 23:** Captura de tela, realizada próximo ao instante 730 segundos da carga Crescente com uso de elasticidade, das máquinas virtuais em execução no OpenNebula.




Fonte: Desenvolvido pelo autor.

**Figura 24:** Resultado da execução da carga Decrescente: (a) PIPEL sem ações elásticas; (b) PIPEL com ações elásticas; Uso de CPU *versus* quantidade de máquinas virtuais alocadas para o estágio 1 (b.1), estágio 2 (b.2) e estágio 3 (b.3).




Fonte: Desenvolvido pelo autor.

### 6.3.3 Comportamento da Carga Constante

Analisando a Figura 25, a execução sem elasticidade (a) segue o padrão de execução *pipeline* sem elasticidade. Comparando a execução sem (a) e com (b) elasticidade é possível afirmar que na execução com elasticidade o tempo total teve redução de 33%. Nesta configuração de carga, os dados são todos iguais, portanto o tempo de execução de cada dado com o uso dos mesmos recursos deve ser equivalente para cada estágio. Essa configuração de carga é a mais uniforme de todas. Isso faz com que, com o uso de elasticidade, os níveis de processamento tendem a ficar entre os *thresholds* definidos ao longo da execução.

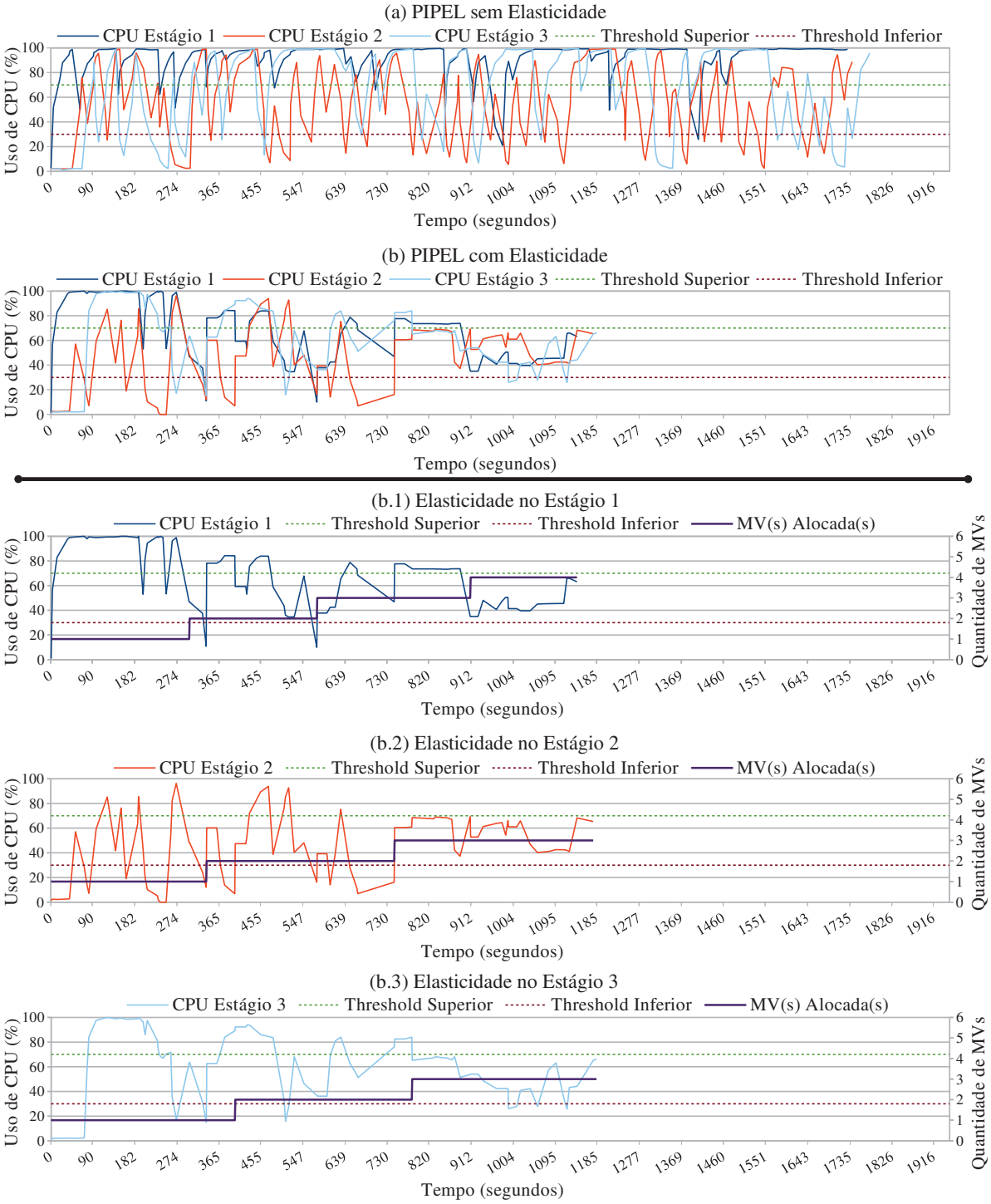
Considerando os estágios de forma individual, no estágio 1 (b.1), nos instantes 13 segundos, 338 segundos e 686 segundos foram detectadas as necessidades de alocação de recursos. O mesmo ocorre no estágio 2 (b.2) nos instantes 122 e segundos 431 segundos. No estágio 3 (b.3) a adição de máquinas virtuais ocorrem nos instantes 119 segundos e 377 segundos. Totalizando 4 MVs para o estágio 1 e 3 MVs para os estágios 2 e 3. No comportamento da execução da aplicação *pipeline* com a configuração de carga Constante não considerou-se necessário a consolidação de recursos porque as condições exigidas para isso não foram atingidas.

### 6.3.4 Comportamento da Carga Oscilante

A ideia, neste comportamento de carga, é gerar um fluxo de dados em formato de onda, simulando uma execução inconstante ou irregular. Analisando a Figura 26, a execução da aplicação sem elasticidade (a) segue o padrão de execução *pipeline* sem elasticidade. Comparando a execução sem elasticidade (a) com a execução com elasticidade (b), percebe-se que na execução com elasticidade o tempo total reduziu 35%. Por se tratar da configuração de carga mais irregular nos experimentos executados, pode-se afirmar que obteve o segundo melhor desempenho com relação ao tempo de execução, não destoando das demais cargas testadas.

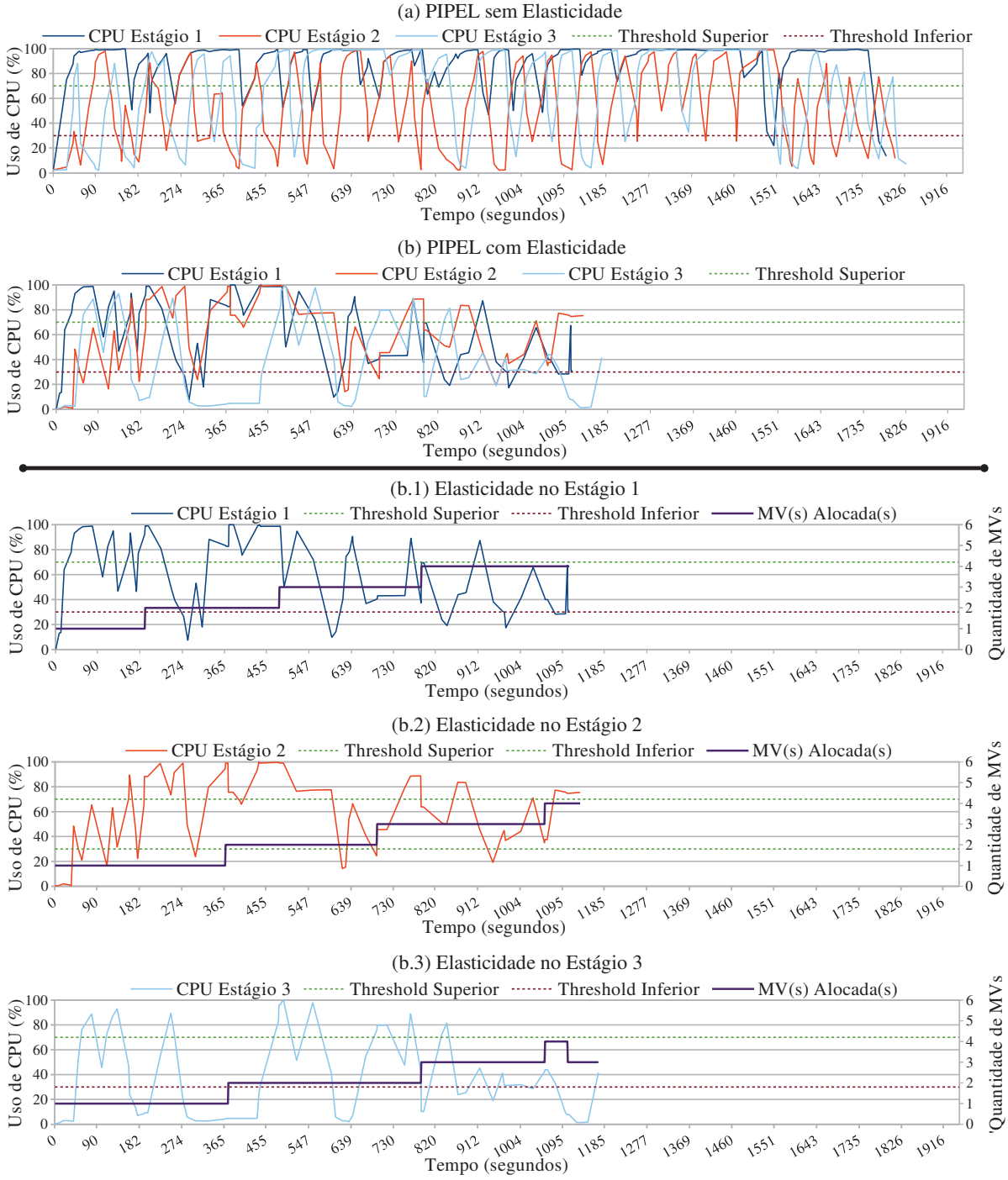
Na análise dos estágios, de maneira individual, nota-se que: (i) o estágio 1 (b.1) sofreu apenas alocações de recursos nos instantes 193 segundo, 483 segundos e 790 segundos; (ii) o estágio 2 (b.2) também teve apenas alocações de recursos nos instantes 368 segundos, 695 segundos e 1057 segundos; (iii) o estágio 3 (b.3) sofreu alocações de recursos nos instantes 374 segundos, 790 segundos e 1062 segundos; (iv) e que no estágio 3, o gerenciador de elasticidade julgou necessário a consolidação de uma máquina virtual no instante 1106 segundos, estabilizando assim, os recursos pela carga de processamento. Por fim, no estágio 1 (b.1), no instante 916 segundos, o gerenciador de elasticidade inicia o procedimento de inclusão de recursos, motivado por um estouro de *threshold* superior. Não chegando a ser alocado de fato, em razão de que a aplicação finaliza sua execução antes da inclusão dessa MV.

**Figura 25:** Resultado da execução da carga Constante: (a) PIPEL sem ações elásticas; (b) PIPEL com ações elásticas; Uso de CPU *versus* quantidade de máquinas virtuais alocadas para o estágio 1 (b.1), estágio 2 (b.2) e estágio 3 (b.3).



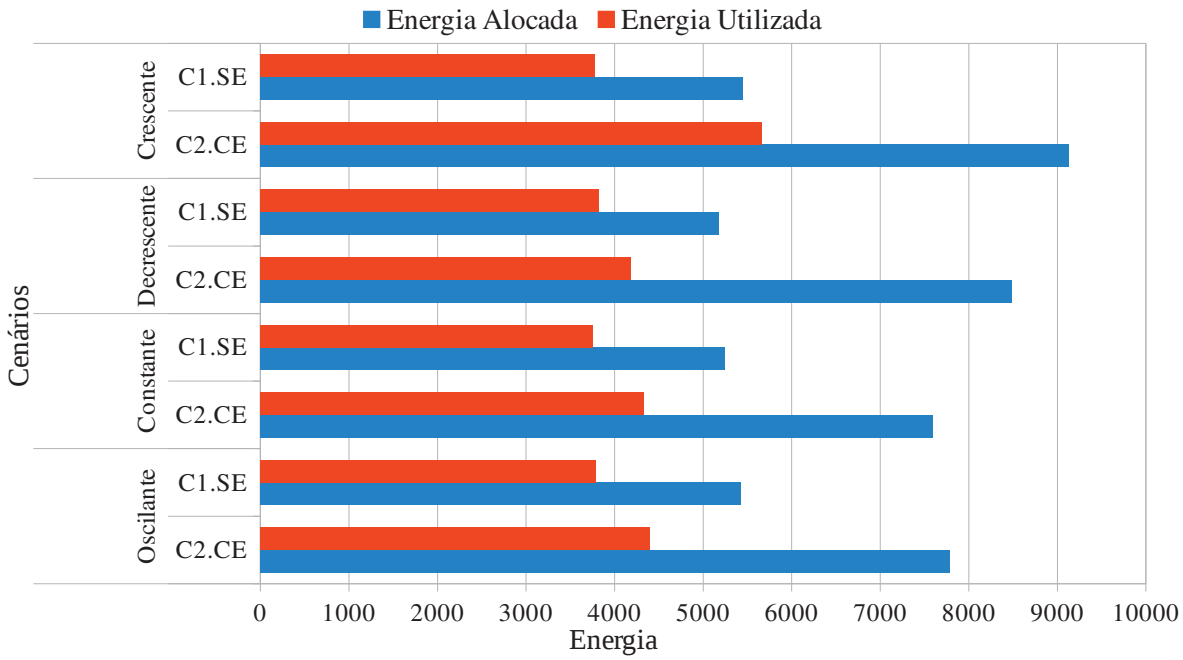

Fonte: Desenvolvido pelo autor.

**Figura 26:** Resultado da execução da carga Oscilante: (a) PIPEL sem ações elásticas; (b) PIPEL com ações elásticas; Uso de CPU *versus* quantidade de máquinas virtuais alocadas para o estágio 1 (b.1), estágio 2 (b.2) e estágio 3 (b.3).




Fonte: Desenvolvido pelo autor.

**Figura 27:** Análise da Alocação *versus* Consumo (Energia) dos Recursos nos dois cenários de testes: (C1.SE) execução da aplicação sem o uso de elasticidade e (C2.CE) execução da aplicação com a utilização de elasticidade.



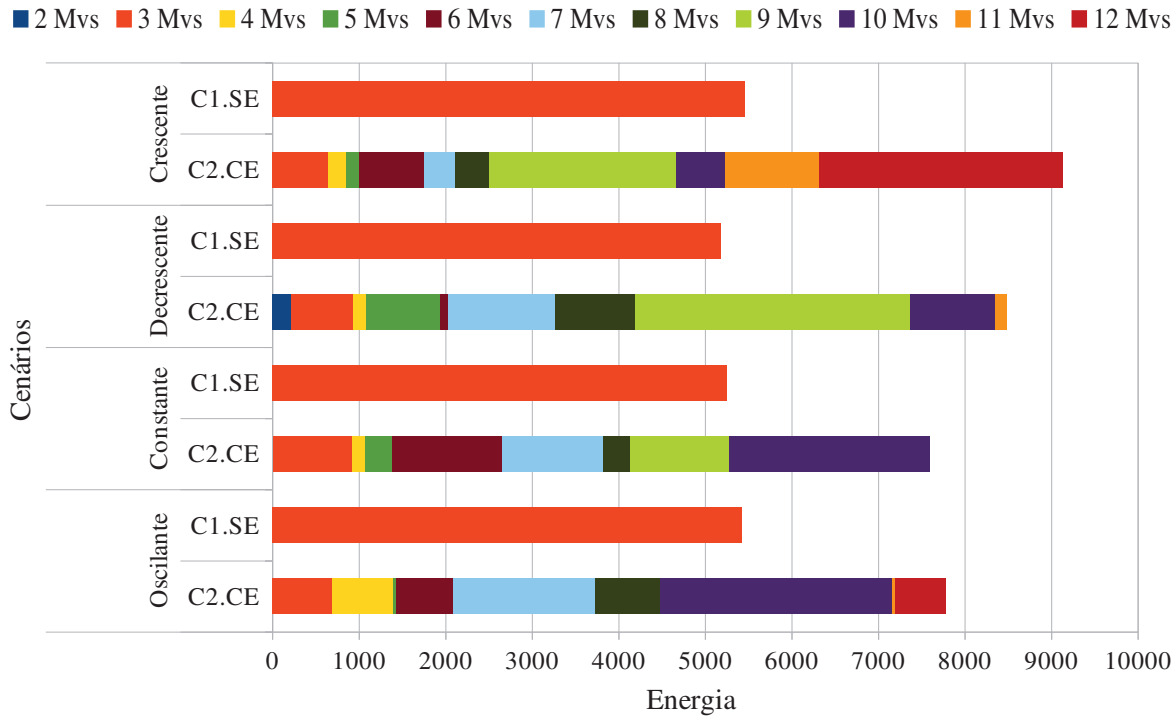

Fonte: Desenvolvido pelo autor.

## 6.4 Análise da Alocação *versus* Consumo dos Recursos

A Figura 27 demonstra a avaliação da estimativa da métrica Energia obtida em todos os cenários testados, considerando todos os comportamentos de carga. Essa métrica representa a quantidade total de recursos que foram utilizados ao longo da execução da aplicação *pipeline*. Como nesta seção, o intuito é fazer a comparação de alocação de recursos, foram consideradas apenas as máquinas virtuais do tipo Trabalhador. Isso porque as máquinas virtuais do tipo Controladores de Estágio não são monitoradas durante a execução da aplicação. Entretanto, na Seção 6.2, as análises de Eficiência contemplam a Energia utilizada pelos Controladores de Estágios.

Nos cenários testados, é adicionada a informação de quanto realmente foi utilizado da capacidade disponibilizada. Como é observada a quantidade de recursos utilizados a cada ciclo de monitoramento, também é obtido o valor da carga de processamento total no mesmo momento. Um exemplo disso é: se cinco máquinas virtuais estão em execução, e o gerenciador de elasticidade obtém o valor da carga de processamento de todos os recursos sendo 70%, o consumo de recursos aponta o valor de 3,5 máquinas virtuais para esse instante. Essa mesma ideia é usada para calcular a energia consumida a cada ciclo de monitoramento. A figura realiza uma comparação de alocação de consumo de recursos das execuções da cada comportamento de carga nos diferentes cenários: (C1.SE) execução da aplicação sem o uso de elasticidade e

**Figura 28:** Distribuição de consumo de recursos nos diferentes cenários: (C1.SE) execução da aplicação sem a utilização de elasticidade e (C2.CE) execução da aplicação com uso de elasticidade.




Fonte: Desenvolvido pelo autor.

(C2.CE) execução da aplicação com a utilização de elasticidade. As figuras correspondentes a cada comportamento de carga foram organizadas de maneira a usar o eixo x com a mesma escala auxiliando na comparação vertical entre eles. Primeiramente, observando o consumo dos recursos, percebe-se que os índices de execuções com o uso de elasticidade apresentadas nos comportamentos Decrescente, Constante e Oscilante atingem um valor muito próximo (entre 4180 e 4394), apresentando uma variação de 4,8% com base no maior valor. O comportamento Crescente destoou um pouco desse padrão, apresentando uma variação de 26% do menor valor. Isso porque, este comportamento de carga é o que mais tempo fica executando, logo utiliza durante mais tempo os recursos. Analisando o consumo dos recursos nas execuções sem o uso da elasticidade, percebe-se que em todas as configurações de carga, o consumo dos recursos atinge índices muito próximos, com variação máxima de 1,7%. A execução sem elasticidade possui os menores valores para alocação e consumo, pois nesses cenários a execução da aplicação é realizada com a menor quantidade de recursos possíveis durante 100% do tempo. Também é importante perceber que existe a indireta proporcionalidade entre Tempo e Energia. Para a execução da aplicação de maneira mais rápida se tornam necessários maior quantidade de recursos. A priorização de economia de alocação de recursos possivelmente resultará penalidades de tempo na execução da aplicação.

A Figura 28 expõe os diferentes perfis de quantidade de recursos alocados e sua contribuição

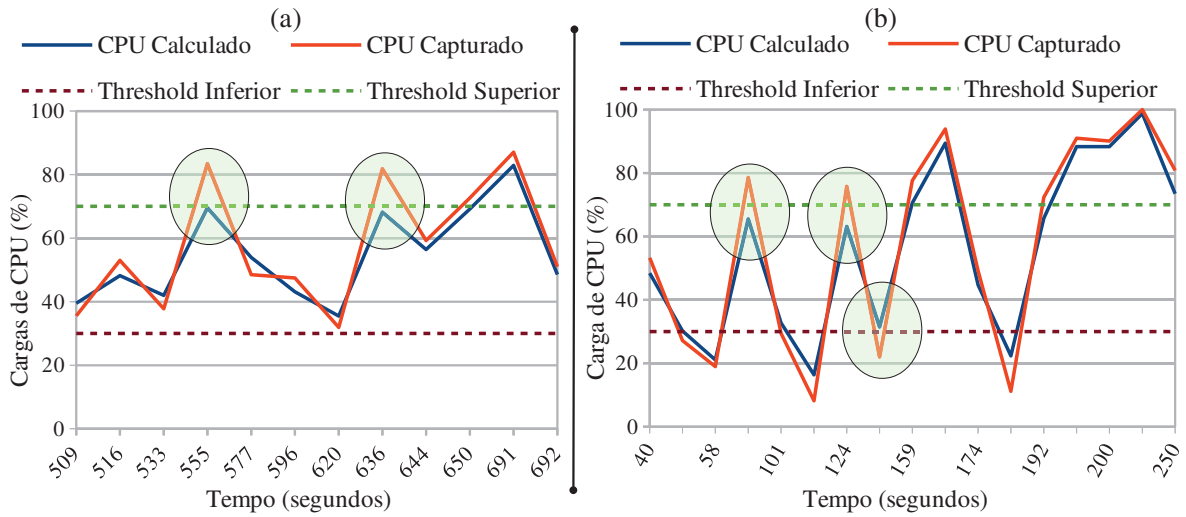
para a métrica Energia nos cenários: (C1.SE) execução sem elasticidade e (C2.CE) execução da aplicação com elasticidade. Observando as diferentes cargas é possível identificar que os menores índices de energia foram obtidos pelas execuções em que menores quantidades de recursos foram utilizadas. Se observados os cenários sem elasticidade, nota-se que foram utilizadas apenas 3 máquinas virtuais do tipo Trabalhador em suas execuções. Entretanto, os maiores resultados da métrica Energia foram obtidos pelas execuções que dispõem de uma maior quantidade de máquinas virtuais. Que é o caso do cenário que utiliza elasticidade na configuração de carga Crescente. Chegando a empregar o maior número de recursos (12 máquinas virtuais) durante a maior quantidade de tempo se comparado com os demais experimentos. Ainda, nos cenários com elasticidade, é pertinente observar que a utilização dos menores índices de recursos acontece em intervalos de tempo que o controlador da nuvem está em intensa atividade, ou seja, executando a alocação de um ou mais recursos ao sistema. Outra observação importante a ser destacada é que apenas as configurações de carga Crescente e Oscilante chegaram a utilizar 12 máquinas virtuais do tipo Trabalhador simultaneamente e que em todas as configurações de carga, pelo menos 10 MVs do tipo Trabalhador foram utilizadas.

## 6.5 Análise da Técnica *Aging* no Gerenciamento da Elasticidade

A Figura 29 apresenta uma comparação entre os dados de CPU capturados e o valores calculados pelo gerenciador de elasticidade que utiliza a técnica de suavização *Aging*. A Figura 29 (a) mostra um trecho da execução da aplicação *pipeline* do estágio 3 na configuração de carga Crescente e a Figura 29 (b) apresenta um pedaço da execução da aplicação do estágio 2 na configuração de carga Oscilante. Nesta ilustração, no estágio 3 na carga Crescente (a), percebe-se que em dois momentos (555 e 636 segundos) a utilização da suavização impossibilitou que o gerenciador de elasticidade alocasse recursos para este estágio, fazendo com que dois falsos-positivos fossem descartados. Na imagem, no estágio 2 na carga Oscilante (b), apresenta também: (i) dois momentos (79 e 124 segundos) em que a suavização de carga evita com que o *threshold* superior seja ultrapassado; e (ii) um momento (134 segundos) em que haja estouro de *threshold* inferior. Como a regra de consolidação de recursos determina que deva haver estouro de *threshold* inferior e satisfazer as regras de tempo de execução de dados por estágio (de acordo com a Seção 4.3.1 do Capítulo 4), neste caso não houve consolidação de máquinas virtuais. Através da figura, nota-se que a carga de CPU Calculada e a carga de CPU Capturada possuem traços próximos, porém na carga Capturada acontecem mais variações entre as observações. Através do cálculo, com a utilização da técnica *Aging*, estas variações são suavizadas para o cálculo da carga do ambiente que é de fato utilizada para que o gerenciador de elasticidade tome as devidas decisões. Desta maneira, percebe-se que a utilização da técnica *Aging* posterga, em alguns casos, a alocação e consolidação de recursos e mantém a conformidade entre as cargas de CPU capturadas nos estágios da aplicação.

Os picos de carga de CPU apresentados na figura ocorrem nos momentos em que novos

**Figura 29:** Comparação entre os dados de CPU capturados e o CPU calculados pelo gerenciador de elasticidade com a utilização da técnica de suavização *Aging*: (a) trecho da execução do estágio 3 na configuração de carga Crescente; (b) pedaço da execução do estágio 2 na configuração de carga Oscilante.




Fonte: Desenvolvido pelo autor.

dados são inseridos em cada estágio da aplicação. Isso ocorre porque, quando o dado sai do estágio existe um tempo de sincronização feito pelos Controladores de Estágio. Nestes breves intervalos de tempo de sincronização, as máquinas virtuais do tipo Trabalhador diminuem seu processamento e as respectivas máquinas virtuais causam um pico na carga de CPU Capturada. Quando novos dados entram no estágio, o processamento novamente aumenta de forma rápida, causando outro pico no consumo de CPU. A utilização da técnica de *Aging* é essencial para aplicações *pipeline*, pois ao invés de utilizar a captura de dados pontuais, ela faz uso de uma média suavizada. Caso não fosse utilizada a suavização da média, o gerenciador de elasticidade poderia ter a falsa ilusão de que, em alguma captura da carga de CPU, o processamento esteja dentro dos limites (*thresholds*), mas na realidade ele poderia ultrapassar estes limites, resultando em ações elásticas.

## 6.6 Análise do Método de Alocação de Recursos

A Tabela 6 demonstra a comparação realizada entre o método de Alocação Sequencial de Recursos (ASR) e o método de Alocação Aleatória de Recursos (AAR). Nesta tabela é possível perceber a distribuição de máquinas virtuais nos seis nós computacionais, disponíveis no ambiente de infraestrutura de nuvem, em cada execução da aplicação *pipeline*. Foram executadas, para cada carga de processamento, os dois métodos de alocação de recursos. Cada nó computacional, utilizado na realização dos experimentos deste estudo, tem a possibilidade de alocar o número máximo de quatro máquinas virtuais. Isto acontece em razão dos nós possuírem quatro *cores* de processamento. Os valores da tabela foram coletados no momento em que a execução

**Tabela 6:** Comparação de alocação de recursos realizada entre o método de Alocação Sequencial de Recursos (ASR) e o método de Alocação Aleatória de Recursos (AAR).

| Carga | Crescente | | Decrescente | | Constante | | Oscilante | |
|---|---|---|---|---|---|---|---|---|
| Método de Alocação | ASR | AAR | ASR | AAR | ASR | AAR | ASR | AAR |
| nó 1 | 4/4 | 4/4 | 4/4 | 4/4 | 4/4 | 4/4 | 4/4 | 4/4 |
| nó 2 | **6/4** | 3/4 | **5/4** | 3/4 | 4/4 | 3/4 | **6/4** | 3/4 |
| nó 3 | 4/4 | 1/4 | **6/4** | 1/4 | **6/4** | 2/4 | **6/4** | 3/4 |
| nó 4 | 2/4 | 2/4 | 2/4 | 3/4 | 2/4 | 1/4 | 2/4 | 2/4 |
| nó 5 | 0/4 | 2/4 | 0/4 | 2/4 | 0/4 | 2/4 | 0/4 | 1/4 |
| nó 6 | 0/4 | 3/4 | 0/4 | 1/1 | 0/4 | 1/1 | 0/4 | 2/4 |
| Total | 16/24 | 15/24 | 17/24 | 14/24 | 16/24 | 13/24 | 18/24 | 15/24 |

Fonte: Desenvolvido pelo autor.

da aplicação registrou o número máximo de máquinas virtuais em utilização no experimento. Isso porque existe a possibilidade de consolidação de recursos durante sua execução. Portanto as máquinas virtuais que foram consolidadas não são consideradas.

Em todas as execuções realizadas com o método AAR, todos os nós foram utilizados. Isso acontece porque a distribuição de recursos é feita de maneira randômica, ou seja, é sorteada entre os nós com disponibilidade de alocação de recursos. Dessa forma, a aleatoriedade da distribuição é variada entre todos os nós, e pelo menos uma máquina virtual foi alocada em cada nó. Nota-se também que todos os experimentos feitos com o método ASR sofreram alocações indevidas, por conta das alocações simultâneas de recursos feitas pelos diferentes estágios da aplicação *pipeline*. Estas alocações, em todos os casos, foram realizadas por mais de um estágio, de forma concorrente, em períodos de tempos muito próximos. Na tabela são destacados (em negrito) essas alocações indevidas. Na carga crescente, observa-se que o nó 2, em determinado momento da aplicação, recebeu seis máquinas virtuais, alocando duas a mais que o ideal permitido. Na carga decrescente, o nó 2 alocou uma máquina a mais que o permitido e no nó 3 foram instanciadas duas máquinas virtuais a mais que o permitido. Na carga constante, o nó 3 recebeu duas MVs a mais do que o máximo permitido. Na carga Oscilante, o nó 2 e o nó 3 instanciaram duas máquinas virtuais cada, a mais do que o permitido pelo acordo de nível de serviço. Todos os experimentos que fizeram uso do método ASR foram prejudicados no tempo de execução. Com relação ao método AAR, a carga crescente teve um acréscimo de 12% (1293 segundos), a carga decrescente teve acréscimo de 22% (1621 segundos), a carga constante teve acréscimo de 17% (1386 segundos) e a carga oscilante teve acréscimo de 27% (1488 segundos). Em função das alocações indevidas de recursos nos *hardwares*, o processamento das máquinas virtuais do tipo trabalhador foram prejudicados. Não só o tempo de execução aumentou, mas também a quantidade de recursos foi maior com o método ASR do que com o método AAR. Esse aumento do uso de recursos está intimamente ligado com o aumento do tempo de execução. Isso porque, com mais tempo de execução da aplicação, com níveis de processamentos

elevados (não ideais), o gerenciador de elasticidade decidiu alocar mais recursos nos estágios da aplicação.

# 7 CONCLUSÃO

O uso de ambientes paralelos e distribuídos trouxe grandes vantagens para o processamento de grandes quantidades de tarefas e fluxos de trabalhos. Porém, para se conseguir melhor desempenho destes sistemas, é necessário que as aplicações executadas nestes ambientes aproveitem ao máximo suas capacidades. Esta Dissertação de Mestrado apresentou um modelo de gerenciamento da elasticidade para aplicações organizadas em *pipeline*, chamado de Pipel. O Pipel faz uso da elasticidade de forma reativa, automática e transparente. A elasticidade automática e reativa tem como ponto de partida regras previamente definidas. Estas regras analisam a carga de processamento do sistema e comparam com *thresholds* e regras de tempo de execução de dados, servindo de base nas tomadas de decisões de elasticidade. A elasticidade transparente refere-se ao fato de que a aplicação não precisa ter seu código fonte alterado, sendo assim, Pipel não evidencia todas as ações de adição e consolidação das máquinas virtuais para o seu usuário. Para avaliar o modelo desenvolvido, foram realizados experimentos com uma aplicação *pipeline*, utilizando variadas cargas de processamento, com dois cenários sobre o *middleware* OpenNebula.

No decorrer do estudo foram apresentados conceitos de Computação em Nuvem, Aplicações com suas Organizações no Formato *Pipeline*, conceitos de Computação Paralela e Análise de Desempenho com utilização de Séries Temporais. Foi feita uma análise bibliográfica em um acervo de periódicos nacionais e internacionais sobre o estado da arte de computação em nuvem aliada a aplicações organizadas no formato *pipeline* e o gerenciamento dos seus recursos computacionais. Concluiu-se que existia uma oportunidade de pesquisa com relação ao gerenciamento da elasticidade entre os estágios da aplicação, de maneira individual por estágio.

Para a realização dos experimentos foram criados dois cenários de testes: (i) sem uso de elasticidade e (ii) com uso da elasticidade. Estes cenários foram testados com diferentes configurações de cargas: (i) Carga Crescente; (ii) Carga Decrescente; (iii) Carga Constante e (iv) Carga Oscilante. Foram apresentadas métricas de avaliação de desempenho e consumo de recursos da aplicação: *Speedup*, Eficiência, Energia e Custo para a avaliação dos resultados. Através da análise de resultados, foi possível demonstrar o impacto do uso de diferentes cargas de processamentos, nos diferentes cenários, para executar ações de elasticidade no desempenho e recursos da aplicação.

Na Seção 1.2 do Capítulo 1, foi realizada a seguinte questão de pesquisa: *O quanto pode melhorar o desempenho de aplicações organizadas em pipeline utilizando um modelo de elasticidade automática e reativa em ambiente de nuvem computacional?* Respondendo à esta pergunta, os resultados demonstraram que foi possível obter de 23% (pior caso) até 38% (melhor caso) de redução no tempo da execução da aplicação com o uso de elasticidade. Apesar da Energia utilizada durante a execução da aplicação aumentar com a utilização da elasticidade, a métrica de Custo manteve conformidade com a Inequação 4.16 na Seção 4.4 do Capítulo 4, que diz que o custo com uso de elasticidade deve ser menor ou igual do que sem sua utilização.

Atingindo redução de Custo máxima de 35% e mínima de 24% na execução dos testes. No geral, os resultados demonstram que um melhor desempenho equivale a uma eficiência menor e um pior desempenho equivale a uma eficiência maior. Isso ocorre pois a aplicação utiliza intensivo uso de CPU e conforme são incluídas máquinas virtuais, as mesmas tendem a ficar cada vez menos utilizadas em razão da carga de processamento ser mais espalhada entre elas.

## 7.1 Contribuições

O modelo Pipel buscou preencher a lacuna identificada nas pesquisas dos trabalhos relacionados que abordam aplicações organizadas em *pipeline* e sua elasticidade. Assim sendo, o desenvolvimento do modelo resultou em contribuições científicas que apresentam adição ao estado da arte que contempla pesquisa de programação paralela, elasticidade em ambiente de computação em nuvem e aplicações organizadas em *pipeline*. As contribuições da pesquisa são:

- **Arcabouço para Aplicações Organizadas em *Pipeline* mais Elasticidade Automática e Individual por Estágios**: A arquitetura de Pipel, em conjunto com toda infraestrutura de nuvem computacional, permite que as aplicações organizadas em *pipeline* sejam executadas com o uso da elasticidade em nuvem. Com o auxílio de um gerenciador de elasticidade, que monitora e reorganiza os recursos, é possível que as máquinas virtuais sejam alocados ou consolidados de acordo com a necessidade de processamento da aplicação. Executando em nível de *PaaS*, Pipel oferece um *middleware* que transforma uma aplicação não elástica em uma aplicação elástica de maneira transparente, ou seja, sem a intervenção do usuário;

- **Múltipla Elasticidade Assíncrona para Aplicações *Pipeline***: com a utilização da elasticidade automática, um desafio é não causar nenhum impacto à execução da aplicação durante a reorganização de recursos. Com esta finalidade, a múltipla elasticidade assíncrona na aplicação *pipeline* possibilita que o Gerenciador de Elasticidade do Pipel consiga realizar todas as operações de elasticidade utilizando uma área de dados compartilhada para gerar notificações que, em união com o OpenNebula, permitem disponibilizar estes recursos para a aplicação;

- **Alocação Aleatória de Recursos**: em aplicações organizadas em *pipeline*, é importante manter a alocação de máquinas virtuais na infraestrutura de nuvem sob controle. Alocações adequadas implicam na redução de custo e tempo de execução da aplicação. Para melhorar o desempenho da aplicação, a adequação ideal dos recursos torna-se uma estratégia interessante. O nó computacional que recebe as máquinas virtuais deve estar dimensionado para executá-las da melhor maneira possível. Assim sendo, é possível afirmar que a utilização da alocação aleatória de recursos trouxe vantagens para o modelo de elasticidade.

A Tabela 7 apresenta a comparação de todos as características consideradas importantes na avaliação dos trabalhos relacionados, conforme Capítulo 3. De acordo com estes trabalhos considerados relevantes para o presente estudo, percebeu-se que existia uma lacuna a ser preenchida, no que se refere a aplicações organizadas em *pipeline* e a elasticidade fornecida por estágios da aplicação de forma individual. Na última linha da tabela, foram inseridas as características de Pipel, salientando na coluna "Elasticidade por estágio(s) em *Pipeline*" que a elasticidade é fornecida individualmente por estágios da aplicação *pipeline*. Dessa maneira, a lacuna encontrada foi preenchida .

**Tabela 7:** Comparação entre as principais características dos trabalhos relacionados e o modelo Pipel. (NI - Não Informado)

| **Trabalho** | **Modelo de Serviço** | **Método de Elasticidade** | **Modelo(s) de Elasticidade** | **Elasticidade por estágio(s) em *Pipeline*** | **Modelo de Computação Paralela** | **Carga de Trabalho** |
|---|---|---|---|---|---|---|
| (LI et al., 2010) | *PaaS* | Horizontal | Manual | Não fornece | *Pipeline* | Redução de Imagens de Satélites |
| (RAJAN; CANINO; IZAGUIRRE, 2011) | *PaaS* | Horizontal | Manual | Não fornece | Mestre Escravo | Troca Molecular Dinâmica |
| (MAO; HUMPHREY, 2011) | *IaaS* | Horizontal | Automático Reativo e Proativo | Não fornece | *Pipeline*, Paralelo e Híbrido | Aplicações no modelo *Workflow* |
| KingFisher (SHARMA et al., 2011) | *IaaS* | Horizontal | Automático Reativo e Proativo | Não fornece | *Bag of Task* | Saturação de E-commerce |
| (APOSTOL et al., 2011) | *IaaS* | Horizontal | Automático Reativo | Não fornece | NI | NI |
| Cloud Operation System (IMAI; CHESTNA; VARELA, 2012) | *PaaS* | Horizontal | Automático Proativo | Não fornece | NI | Problema de Difusão de Calor |
| Scattered (ZHANG et al., 2012) | *IaaS* | Horizontal | Automático Proativo | Não fornece | *Workflows* e *Bag of Pipeline* | Aplicação Comercial Real |
| (RODRIGUEZ; BUYYA, 2015) | *IaaS* | Horizontal | Automático Reativo e Proativo | Não fornece | *Pipeline*, *Bag of Task* e *Bag of Pipeline* | Aplicações no modelo *Workflow* |
| Auto-Elastic (RIGHI et al., 2015) | *PaaS* | Horizontal | Automático Reativo | Não fornece | Mestre Escravo | Cálculo Diferencial |
| Pipel | *PaaS* | Horizontal | Automático Reativo | **Fornece** | *Pipeline* | Processamento de Imagens |

Fonte: Desenvolvido pelo autor.

## 7.2 Trabalhos Futuros

Após a realização deste estudo, percebeu-se alguns pontos que podem ser encarados como oportunidades de extensões para trabalhos futuros:

- Acrescentar um método que avalie de forma proativa as decisões de elasticidade e comparar com o modelo atual;
- Avaliar o uso de elasticidade vertical para executar Pipel em ambientes heterogêneos;
- Desenvolver um método que monitore o "tempo de vida" de cada máquina virtual e considere esse parâmetro durante a execução da aplicação a fim de reduzir custos com aloca-

ção de recursos. Ao invés de instanciar uma nova MV, pode ser interessante realocar as já existentes.

# REFERÊNCIAS


AKHANI, J.; CHUADHARY, S.; SOMANI, G. Negotiation for resource allocation in IaaS cloud. **COMPUTE '11 Proceedings of the Fourth Annual ACM Bangalore Conference**, New York, NY, USA, n. 15, p. 1–7, Mar. 2011.

AL-HAIDARI, F.; SQALLI, M.; SALAH, K. Impact of CPU Utilization Thresholds and Scaling Size on Autoscaling Cloud Resources. **2013 IEEE 5th International Conference on Cloud Computing Technology and Science**, Bristol, p. 256–261, Dec. 2013.

APOSTOL, E.; BALUTA, I.; GORGOI, A.; CRISTEA, V. Efficient manager for virtualized resource provisioning in Cloud Systems. **2011 IEEE International Conference on Intelligent Computer Communication and Processing (ICCP)**, Cluj-Napoca, p. 511–517, Aug. 2011.

BALDO, L.; BRENNER, L.; FERNANDES, L. G.; FERNANDES, P.; SALES, A. Performance Models For Master/Slave Parallel Programs. **Proceedings of the First International Workshop on Practical Applications of Stochastic Modelling (PASM 2004)**, Royal Society, London, n. 128, p. 101–121, Apr. 2005.

BALIGA, J.; AYRE, R. W. A.; HINTON, K.; TUCKER, R. S. Green Cloud Computing: balancing energy in processing, storage, and transport. **Proceedings of the IEEE**, IEEE, p. 149–67, Jan. 2011.

BATISTA, L. V. **Introdução ao Processamento Digital de Imagens**. Acessado em 22/06/2016, `http://www.di.ufpb.br/leonardo/pdi/PDI2005.pdf`.

BERNSTEIN, P. A. Middleware: a model for distributed system services. **Magazine Communications of the ACM**, New York, NY, USA, p. 86–98, Feb. 1996.

BHARATHI, S.; CHERVENAK, A.; DEELMAN, E.; MEHTA, G.; SU, M.-H.; VAHI, K. Characterization of scientific workflows. **WORKS 2008. Third Workshop on Workflows in Support of Large-Scale Science, 2008.**, Austin, TX, p. 1–10, Nov. 2008.

BOX, G. E. P.; JENKINS, G. M.; REINSEL, G. C. **Time series analysis**: forecasting and control. IEEE: . 4. ed. Upper Saddle River, NJ, USA, 2003.

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

CHAVAN, P.; KULKARNI, P. P. G.; SUTAR, R.; BELSARE, S. IaaS Cloud Security. **2013 International Conference on Machine Intelligence and Research Advancement (ICMIRA)**, Katra, p. 549 – 553, Dec. 2013.

CHEN, H. bung; GRIDER, G.; INMAN, J.; FIELDS, P.; KUEHN, J. A. . An empirical study of performance, power consumption, and energy cost of erasure code computing for HPC cloud storage systems. **Architecture and Storage (NAS), 2015 IEEE International Conference on Networking**, Boston, MA, p. 71–80, 2015.

CHEN, Y.; ZHU, A. Implementation of Linux centralized user authentication and cloud storage in teaching. **2014 International Conference on Information Science, Electronics and Electrical Engineering (ISEEE)**, Sapporo, n. 1, p. 626–628, Apr. 2014.

CHILIPIREA, C.; LAURENTIU, G.; POPESCU, M.; RADOVENEANU, S.; CERNOV, V.; DOBRE, C. A Comparison of Private Cloud Systems. **2016 30th International Conference on Advanced Information Networking and Applications Workshops (WAINA)**, IEEE, p. 139–143, Dec. 2016.

COSTA, F.; OLIVEIRA, D. de; OCALA, K.; OGASAWARA, E.; DIAS, J.; MATTOSO, M. Handling Failures in Parallel Scientific Workflows Using Clouds. **High Performance Computing, Networking, Storage and Analysis (SCC), 2012 SC Companion**, Salt Lake City, UT, p. 129–139, 2012.

DAVIDSON, S. B.; FREIRE, J. Provenance and scientific workflows: challenges and opportunities. **2008 ACM SIGMOD international conference on Management of data**, New York, NY, USA, p. 1345–1350, 2008.

DAWOUD, W.; TAKOUNA, I.; MEINEL, C. Elastic VM for Cloud Resources Provisioning Optimization. **Advances in Computing and Communications: First International Conference, ACC 2011**, Kochi, India, p. 431–435, July 2011.

DEELMAN, E.; GANNON, D.; SHIELDS, M.; AND, I. T. Workflows and e-Science: an overview of workflow system features and capabilities. **Future Generation Computer Systems**, Amsterdam, n. 25, p. 528–540, May 2009.

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

EVANGELINOS, C.; HILL, C. N. Cloud Computing for parallel Scientific HPC Applications: feasibility of running coupled atmosphere-ocean climate models on amazon's ec2. **Workshop on Cloud Computing and its Applications (CCA)**, n/a, n. 1, 2008.

F. GUIRADO A. RIPOLL, C. R. A. H. E. L. Exploiting Throughput for Pipeline Execution in Streaming Image Processing Applications. **Euro-Par 2006 Parallel Processing**, Springer Berlin Heidelberg, p. 1095–1105, 2006.

FRINCU, M.; GENAUD, S.; GOSSA, J. Comparing Provisioning and Scheduling Strategies for Workflows on Clouds. **2013 IEEE 27th International Parallel and Distributed Processing Symposium Workshops PhD Forum (IPDPSW)**, Cambridge, MA, p. 2101–2110, May 2013.

GALANTE, G.; BONA, L. de. A Survey on Cloud Computing Elasticity. **2012 IEEE Fifth International Conference on Utility and Cloud Computing (UCC)**, Chicago, IL, p. 263 – 270, Nov. 2012.

GUOXIN, L.; WANLI, Z.; LIRONG, W. Application of OPC to Realize the Communications between WinCC and Master-Slave PLC in the PROFIBUS Network. **2015 IEEE Fifth International Conference on Big Data and Cloud Computing (BDCloud)**, Dalian, China, p. 227–230, Aug. 2015.

HARTLEY, T. D. R.; FASIH, A. R.; BERDANIER, C. A.; OZGUNER, F. Investigating the use of GPU-accelerated nodes for SAR image formation. **2009 IEEE International Conference on Cluster Computing and Workshops**, New Orleans, LA, p. 1–8, 2009.

HENDRIX, V.; FOX, J.; GHOSHAL, D.; RAMAKRISHNAN, L. Tigres Workflow Library: supporting scientific pipelines on hpc systems. **2016 16th IEEE/ACM International Symposium on Cluster, Cloud and Grid Computing (CCGrid)**, Cartagena, Colombia, p. 146–155, May 2016.

HENNESSY, J. L.; PATTERSON, D. A. **Computer Architeture, Fifth Edition**. San Francisco, CA, USA: 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. **CICPE '13 Proceedings of the 4th ACM/SPEC International Conference on Performance Engineering**, New York, NY, USA, p. 187–198, Mar. 2013.

HOROWITZ, E.; ZORAT, A. Divide-and-Conquer for Parallel Processing. **IEEE Transactions on Computers**, IEEE, p. 582–585, June 1983.

IMAI, S.; CHESTNA, T.; VARELA, C. A. Elastic Scalable Cloud Computing Using Application-Level Migration. **2012 IEEE Fifth International Conference on Utility and Cloud Computing (UCC)**, Chicago, IL, p. 91–98, Nov. 2012.

JADEJA, Y.; MODI, K. Cloud computing - concepts, architecture and challenges. **2012 International Conference on Computing, Electronics and Electrical Technologies (ICCEET)**, Kumaracoil, p. 877–880, Mar. 2012.

KUMAR, V.; RAVIKUMAR, K.; ARAVINTH, S. S.; RAJKUMAR, B. A message passing interface to support fast data access in distributed cloud environment along with master and slave communication. **2014 2nd International Conference on Current Trends in Engineering and Technology (ICCTET)**, Coimbatore, p. 309–312, July 2014.

KUMMAR, V. **Introduction to Parallel Computing**. Boston, MA, USA: 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. **2011 38TH ANNUAL INTERNATIONAL SYMPOSIUM ON COMPUTER ARCHITECTURE (ISCA)**, San Jose, CA, p. 129–140, 2011.

LI, J.; HUMPHREY, M.; AGARWA, D.; INGEN, C. van; JACKSON, K.; RYUL, Y. eScience in the cloud: a modis satellite data reprojection and reduction pipeline in the windows azure platform. **2010 IEEE International Symposium on Parallel Distributed Processing (IPDPS)**, Atlanta, GA, p. 1–10, 2010.

LIN, C.; LU, S. Scheduling Scientific Workflows Elastically for Cloud Computing. **2011 IEEE International Conference on Cloud Computing (CLOUD)**, Washington, DC, p. 746–747, July 2011.

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, p. 559–592, 2014.

MALAWSKI, M.; JUVE, G.; DEELMAN, E.; NABRZYSKI, J. Cost- and deadline-constrained provisioning for scientific workflow ensembles in IaaS clouds. **Proceeding SC '12 Proceedings of the International Conference on High Performance Computing, Networking, Storage and Analysis**, Los Alamitos, CA, USA, n. 22, 2012.

MANDAL, A.; KENNEDY, K.; KOELBEL, C.; MARIN, G.; MELLOR-CRUMMEY, J.; LIU, B.; JOHNSSON, L. Scheduling strategies for mapping application workflows onto the grid. **14th IEEE International Symposium on High Performance Distributed Computing, 2005. HPDC-14. Proceedings.**, IEEE, p. 125 – 134, July 2005.

MAO, M.; HUMPHREY, M. Auto-scaling to minimize cost and meet application deadlines in cloud workflows. **SC '11 Proceedings of 2011 International Conference for High Performance Computing, Networking, Storage and Analysis**, New York, NY, USA, n. 49, 2011.

MATTOSO, M.; WERNER, C.; TRAVASSOS, G. H.; BRAGANHOLO, V.; OGASAWARA, E.; OLIVEIRA, D. de; CRUZ, S. M. S. da; MARTINHO, W.; MURTA, L. Towards Supporting the Life Cycle of Largescale Scientific Experiments. **International Journal of Business Process Integration and Management**, New York, NY, USA, n. 1, p. 79–92, Mar. 2010.

MELL, P.; GRANCE, T. Definition of Cloud Computing. **National Institute of Standards and Technology (NIST)**, Gaithersburg, MD, 2011.

MILOJICIC, D.; LLORENTE, I. M.; MONTERO, R. S. OpenNebula: a cloud management tool. internet computing. **IEEE**, New York, NY, USA, n. 2, p. 11–14, Mar. 2011.

MISHRA, M.; SAHOO, A. On theory of VM placement: anomalies in existing methodologies and their mitigation using a novel vector based approach. **In IEEE 4th International Conference on Cloud Computing**, Washington, DC, p. 275–282, Dec. 2011.

MUCHALSKI, F. J.; MAZIERO, C. A. Alocação de Máquinas Virtuais em Ambientes de Computação em Nuvem Considerando o Compartilhamento de Memória. **Anais do XII Workshop de Computação em Clouds e Aplicações - WCGA 2014**, Florianópolis, 2014.

NISHIDATE, I.; SASAOKA, K.; YUASA, T.; NIIZEKI, K.; MAEDA, T.; ; AIZU, Y. Visualizing of skin chromophore concentrations by use of RGB images. **OPTICS LETTERS**, New York, NY, USA, p. 2263–2265, Oct. 2008.

OPENNEBULA. **Release Cycle**. Acessado em 29/11/2015, `http://opennebula.org/software/release/`.

ORGERIE, A.-C.; ASSUNCAO, M. D. de; LEFEVRE, L. A survey on techniques for improving the energy efficiency of large-scale distributed systems. **ACM Computing Surveys (CSUR)**, New York, NY, USA, p. 187–198, Apr. 2014.

PINHO, M. S. **Computação Gráfica**: manipulação de imagens. Acessado em 21/06/2016, `http://www.inf.pucrs.br/ pinho/CG/Aulas/Img/IMG.htm`.

PRAJAPATI, H.; VIJ, S. Analytical study of parallel and distributed image processing. **2011 International Conference on Image Information Processing (ICIIP)**, Himachal Pradesh, p. 1–6, Nov. 2011.

RAJAN, D.; CANINO, A.; IZAGUIRRE, J. A. Converting a High Performance Application to an Elastic Cloud Application. **2011 IEEE Third International Conference on Cloud Computing Technology and Science (CloudCom)**, Athens, p. 383–390, Dec. 2011.

RAJARAMAN, V. Cloud computing. **Resonance**, Springer India, n. 19, p. 242–258, Mar. 2014.

RAMANATH, R.; SNYDER, W. E.; YOO, Y.; DREW, M. S. Color image processing pipeline. **IEEE Signal Processing Magazine**, IEEE Signal Processing Society, p. 34–43, Mar. 2005.

RAMGOVIND, S.; ELOFF, M.; SMITH, E. The management of security in Cloud computing. **Information Security for South Africa (ISSA), 2010**, Sandton, Johannesburg, p. 1–7, Aug. 2010.

RIGHI, R.; RODRIGUES, V.; COSTA, C. A. da; GALANTE, G.; BONA, L.; FERRETO, T. AutoElastic: automatic resource elasticity for high performance applications in the cloud. **IEEE Transactions on Cloud Computing**, IEEE, Apr. 2015.

RIMAL, B.; CHOI, E.; LUMB, I. A Taxonomy and Survey of Cloud Computing Systems. **Fifth International Joint Conference on INC, IMS and IDC, 2009. NCM '09.**, Seoul, p. 44 –51, Aug. 2009.

RIMAL, B. P.; CHOI, E.; LUMB, I. A Taxonomy, Survery, and Issues of Cloud Computing Ecosystems. **Cloud Computing: principles, systems and applications**, Springer London, p. 21–46, 2010.

RODRIGUEZ, M. A.; BUYYA, R. A Responsive Knapsack-based Algorithm for Resource Provisioning and Scheduling of Scientific Workflows in Clouds. **2015 International Conference on Parallel Processing**, Beijing, China, Sept. 2015.

ROSA RIGHI, R. da; COSTA, C. A. da; RODRIGUES, V. F.; ROSTIROLLA, G. Joint-analysis of performance and energy consumption when enabling cloud elasticity for synchronous HPC applications. **Concurrency and Computation: Practice and Experience**, n/a, p. n/a–n/a, 2015. cpe.3710.

ROSA RIGHI, R. da; COSTA, C. A. da; RODRIGUES, V. F.; ROSTIROLLA, G. Joint-analysis of performance and energy consumption when enabling cloud elasticity for synchronous HPC applications. **CONCURRENCY AND COMPUTATION: PRACTICE AND EXPERIENCE**, Arequipa, p. 1548–1571, Apr. 2016.

SADIQ S., O. M. Modelling and Verification of Workflow Graphs, Technical Report, nº 386. **Department of Computer Science, University of Queensland**, Queensland, Australia, 1996.

SAHHAF, S.; TAVERNIER, W.; CZENTYE, J.; SONKOLY, B.; SKOLDSTROM, P.; JOCHA, D.; GARAY, J. Scalable Architecture for Service Function Chain Orchestration. **2015 Fourth European Workshop on Software Defined Networks (EWSDN)**, Bilbao, Spain, p. 19–25, Oct. 2015.

SCHNEIDER, S.; ANDRADE, H.; GEDIK, B.; BIEM, A. Elastic scaling of data parallel operators in stream processing. **Parallel Distributed Processing, 2009. IPDPS 2009. IEEE International Symposium on**, Rome, p. 1–12, May 2009.

SCHNEIDER, S.; HIRZEL, M.; GEDIK, B.; WU, K.-L. Safe Data Parallelism for General Streaming. **IEEE Transactions on Computers**, IEEE, p. 504–517, Jan. 2015.

SEFRAOUI, O.; AISSAOUI, M.; ELEULDJ, M. OpenStack: toward an open-source solution for cloud computing. **International Journal of Computer Applications**, NY,USA, n. 3, p. 38–42, Oct. 2012.

SHARMA, U.; SHENOY, P.; SAHU, S.; SHAIKH, A. Kingfisher: cost-aware elasticity in the cloud. **INFOCOM, 2011 Proceedings IEEE**, Shanghai, p. 206–210, Apr. 2011.

SHUANG, K.; ZHANG, T.; DONG, Z.; XU, P. Impact of HTTP Pipelining Mechanism for Web Browsing Optimization. **2015 IEEE International Conference on Mobile Services (MS)**, New York, NY, p. 415–422, July 2015.

SZABO, R.; KIND, M.; WESTPHAL, F. J.; WOESNER, H.; JOCHA, D.; CSASZAR, A. Elastic network functions: opportunities and challenges. **IEEE Network**, IEEE, p. 15–21, May 2015.

TANENBAUM, A. **Computer Networks**. NJ, USA: 4th. ed. Hoboken, NJ, USA: John Wiley Sons, Ltd., 2008.

TRAN, G. S.; TCHANA, A.; HAGIMONT, D.; PALMA, N. de. Cooperative Resource Management in a Iaas. **2015 IEEE 29th International Conference on Advanced Information Networking and Applications (AINA)**, Gwangiu, p. 611–618, Mar. 2015.

VAQUERO, L. M.; RODERO-MERINO, L.; CACERES, J.; LINDNER, M. A break in the clouds: towards a cloud definition. **ACM SIGCOMM Computer Communication Review**, New York, NY, USA, n. 39, p. 50–55, Jan. 2009.

WALKER, J.; BORGO, R.; JONES, M. W. TimeNotes: a study on effective chart visualization and interaction techniques for time-series data. **IEEE Transactions on Visualization and Computer Graphics**, IEEE, p. 549–558, Aug. 2015.

WANG, B.; LI, M.; WANG, H. Geometric Range Search on Encrypted Spatial Data. **IEEE Transactions on Information Forensics and Security**, Crans-Montana, p. 704–719, Mar. 2016.

WANG, G.; NG, T. The Impact of Virtualization on Network Performance of Amazon EC2 Data Center. **INFOCOM, 2010 Proceedings IEEE**, San Diego, CA, p. 1–9, Mar. 2010.

WU, C.-J.; ; KU, C.-F.; HO, J.-M.; CHEN, M.-S. A Novel Pipeline Approach for Efficient Big Data Broadcasting. **IEEE Transactions on Knowledge and Data Engineering**, IEEE, p. 17–28, Aug. 2016.

ZHANG, X.; JAVAID, H.; SHAFIQUE, M.; PEDDERSEN, J.; HENKEL, J.; PARAMESWARAN, S. OE-pipeline: elastic hardware/software pipelines on a many-core fabric. **2015 Design, Automation Test in Europe Conference Exhibition (DATE)**, Grenoble, p. 363–368, Mar. 2016.

ZHANG, X.; SHAE, Z.-Y.; ZHENG, S.; JAMJOOM, H. Virtual Machine Migration in an Over-committed Cloud. **Network Operations and Management Symposium (NOMS)**, Maui, HI, p. 196–203, Apr. 2012.

ZOUNMEVO, J. A.; KIMPE, D.; ROSS, R.; AFSAHI, A. Using MPI in high-performance computing services. **Proceedings of the 20th European MPI Users' Group Meeting**, New York, NY, USA, p. 43–48, 2013.