UNIVERSIDADE FEDERAL DE SANTA CATARINA
DEPARTAMENTO DE INFORMÁTICA E ESTATÍSTICA

Rodrigo Pedro Marques

DESENVOLVIMENTO DE UMA HEURÍSTICA DE BALANCEAMENTO DE MEMÓRIA RAM EM AMBIENTES COMPUTACIONAIS EM NUVEM

Florianópolis

2017

Rodrigo Pedro Marques

DESENVOLVIMENTO DE UMA HEURÍSTICA DE BALANCEAMENTO DE MEMÓRIA RAM EM AMBIENTES COMPUTACIONAIS EM NUVEM

TCC submetido ao Programa de Graduação em Ciências da Computação para a obtenção do Grau de Bacharelado em Ciências da Computação.
Orientador
Prof. Dr. Carlos Becker Westphall: Universidade Federal de Santa Catarina
Coorientador
Me. Gabriel Beims Bräscher: Universidade Federal de Santa Catarina

Florianópolis

2017

Ficha de identificação da obra elaborada pelo autor através do
Programa de Geração Automática da Biblioteca Universitária da UFSC.

A ficha de identificação é elaborada pelo próprio autor

Maiores informações em:
http://portalbu.ufsc.br/ficha

Rodrigo Pedro Marques

DESENVOLVIMENTO DE UMA HEURÍSTICA DE BALANCEAMENTO DE MEMÓRIA RAM EM AMBIENTES COMPUTACIONAIS EM NUVEM

Este TCC foi julgado aprovado para a obtenção do Título de "Bacharelado em Ciências da Computação", e aprovado em sua forma final pelo Programa de Graduação em Ciências da Computação.

Florianópolis, 01 de junho 2017.

Universidade Federal de Santa Catarina
Orientador
Prof. Dr. Carlos Becker Westphall

Banca Examinadora:

Universidade Federal de Santa Catarina
Coorientador
Me. Gabriel Beims Bräscher

Universidade Federal de Santa Catarina
Prof.ª Dr.ª Carla Merkle Westphall

Dedico este trabalho à minha futura esposa Mariana Felipe, aos meus pais e à minha irmã Priscilla, que, com muito carinho e apoio, não mediram esforços para que eu chegasse até esta etapa de minha vida.

AGRADECIMENTOS

Gostaria de agradecer primeiramente à instituição e ao Laboratório de Redes e Gerência pelo ambiente criativo e amigável que me proporcionaram.

Agradecer ao Prof. Dr. Carlos Becker Westphall pela oportunidade e apoio na elaboração deste trabalho.

Agradecer ao meu coorientador Gabriel Beims Bräscher, por todo o empenho dedicado à elaboração deste trabalho, assim como todo o suporte que me deu no pouco tempo que lhe coube. Sempre me auxiliou realizando suas correções e incentivos.

Agradeço principalmente a minha futura esposa Mariana Felipe, que acompanhou todo o percurso da minha formação. Ela que sempre esteve lá para me apoiar, consolidar e ajudar em todos os momentos que precisei. Te amo muito, paixão!

Agradeço aos meus pais pelo incentivo e apoio que me proporcionaram.

Meus agradecimentos aos meus amigos e colegas de curso, que fizeram parte da minha formação e que vão continuar presentes em minha vida.

Sonhos determinam o que você quer. Ação determina o que você conquista.

(Aldo Novak)

RESUMO

Devido ao grande crescimento do uso de ambientes computacionais em nuvem, a complexidade do gerenciamento destes ambientes tem aumentado significativamente, tornando o gerenciamento manual impraticável. Como consequência disto, a automatização deste processo se mostra importante pois auxilia a agilizar o gerenciamento, melhorando o desempenho quando feito de forma correta. Neste trabalho, é apresentado o desenvolvimento de uma heurística de balanceamento de memória RAM nestes ambientes, buscando aproveitar melhor este recurso entre *hosts* e *clusters*. Esta heurística facilitará o gerenciamento deste recurso em ambientes computacionais em nuvem.

Palavras-chave: Computação em nuvem. Heurística de balanceamento. Balanceamento de recursos.

ABSTRACT

Due to the large growing of cloud computing, the management complexity of these environments has grown, turning the manual management impractical. As a consequence, the automation of these environments takes an important part in making the management faster, improving the environment performance when implemented correctly. This work presents the development of a RAM memory balancing heuristic directed to cloud computing environment. This heuristic aims at making a better usage of this resource among hosts and clusters, improving the RAM memory management in cloud computing environments.

Keywords: Cloud computing. Balancing heuristic. Resources balancing

LISTA DE FIGURAS

Figura 1 Infraestrutura do CloudStack, figura traduzida de (FOUNDATION, 2016b). 30
Figura 2 Arquitetura Básica do OpenStack (OPENSTACK, 2016d). 31
Figura 3 Arquitetura conceitual do HPE Helion Eucalyptus (PACKARD, 2016a). 32
Figura 4 Infraestrutura do OpenNebula (OPENNEBULA, 2016c). . 34
Figura 5 Ilustração do problema a ser corrigido. 44
Figura 6 Ilustracao da proposta deste trabalho. 45
Figura 7 Arquitetura da orquestração em nuvem com o *framwork Autonomiccs* (WEINGÄRTNER; BRÄSCHER; WESTPHALL, 2016). . . . 45
Figura 8 Resultado de Alocação de RAM x Tempo das heurísticas 54

LISTA DE ABREVIATURAS E SIGLAS

QoS *Quality of Service* 21
VPN *Virtual Private Networks* 25
SO Sistema Operacional 25
VMM *Virtual Machine Monitor* 27
SaaS *Software as a Service* 27
PaaS *Plataform as a Service* 27
IaaS *Infrastructure as a Service* 27
API *Application Programming Interface* 29
AWS *Amazon Web Services* 32
UI *User Interface* 32
CLC *Cloud Controller* 32
SOS *Scalable Object Storage* 33
CC *Cluster Controller* 33
SC *Storage Controller* 33
NC *Node Controller* 33
QoS Quality of Service 35

SUMÁRIO

1 INTRODUÇÃO . . . 21
1.1 MOTIVAÇÃO . . . 22
1.2 OBJETIVOS . . . 23
1.2.1 Objetivo Geral . . . 23
1.2.2 Objetivos Específicos . . . 23
1.3 ORGANIZAÇÃO DO TEXTO . . . 23
2 FUNDAMENTAÇÃO TEÓRICA . . . 25
2.1 VIRTUALIZAÇÃO . . . 25
2.2 HYPERVISOR . . . 27
2.3 COMPUTAÇÃO EM NUVEM . . . 27
2.4 ORQUESTRAÇÃO DE AMBIENTES DE COMPUTAÇÃO EM NUVEM . . . 28
2.4.1 Apache CloudStack . . . 29
2.4.2 OpenStack . . . 31
2.4.3 HPE Helion Stackato . . . 32
2.4.4 OpenNebula . . . 33
2.5 QUALIDADE E DEGRADAÇÃO DE SERVIÇO . . . 35
2.6 GERÊNCIA DE AMBIENTES DE COMPUTAÇÃO EM NUVEM . . . 35
2.6.1 Balanceamento . . . 35
2.7 SISTEMAS COMPUTACIONAIS AUTÔNOMOS . . . 36
2.7.1 Sistemas Autônomos x Sistemas Automatizados . . . 36
2.8 PLATAFORMA AUTONOMICCS . . . 37
3 TRABALHOS RELACIONADOS . . . 39
4 PROPOSTA . . . 43
4.1 ILUSTRAÇÃO DO PROBLEMA . . . 43
4.2 ARQUITETURA DO *FRAMEWORK* AUTONOMICCS . 45
4.3 ALGORITMO . . . 46
5 DESENVOLVIMENTO E ANÁLISE . . . 51
5.1 AJUSTES PRELIMINARES E IMPLEMENTAÇÃO . . . 51
5.2 RESULTADOS E ANÁLISE . . . 53
6 CONCLUSÃO E CONSIDERAÇÕES FINAIS . . . 55
REFERÊNCIAS . . . 57
APÊNDICE A – Código da Heurística Desenvolvida 63
APÊNDICE B – Artigo do SBC deste Trabalho de Conclusão de Curso . . . 69

1 INTRODUÇÃO

Computação em nuvem é um modelo que possibilita o acesso conveniente e sob-demanda a recursos computacionais (por exemplo, redes, servidores, armazenamento, aplicações e serviços) que podem ser alocados e liberados com um mínimo de esforço de gerenciamento ou interação com o provedor de serviço (MELL; GRANCE, 2011).

Devido ao crescimento da demanda por serviços ofertados por ambientes computacionais, a estrutura necessária para hospedar estes ambientes aumenta em tamanho e complexidade, impactando diretamente no gerenciamento destes ambientes (WEINGÄRTNER; BRÄSCHER; WESTPHALL, 2015) e (GERONIMO et al., 2013). Devido à complexidade destas infraestruturas, a gerência de ambientes de computação em nuvem requer o auxílio de sistemas computacionais autônomos, como o modelo autônomo proposto por (KEPHART; CHESS, 2003).

Weingärtner, Bräscher e Westphall (2016) desenvolveram um *framework* distribuído autônomo para o *CloudStack* (FOUNDATION, 2016a) que avalia a infraestrutura e, com base em determinadas heurísticas[1], toma decisões para migrar máquinas virtuais[2] e ligar/desligar servidores. Com a finalidade de demonstrar o potencial do framework desenvolvido, Weingärtner, Bräscher e Westphall (2016) implementaram uma heurística que busca a redução do consumo energético em ambientes de computação em nuvem, por meio de técnicas de consolidação[3] de máquinas virtuais. Porém, a solução apresentada por Weingärtner, Bräscher e Westphall (2016) possui carência de outras heurísticas, como uma heurística que balanceasse as máquinas virtuais entre os *hosts* com o melhor desempenho e aproveitamento possível.

Técnicas de gerência e otimização de ambientes em nuvem podem trazer benefícios como o controle da qualidade de serviços (QoS) e redução de gastos energéticos. Porém, diferentes técnicas podem trazer consequências, como por exemplo, o desligamento de servidores para economizar recursos energéticos pode vir a sobrecarregar servidores e prejudicar diretamente os clientes.

Apesar de muitos trabalhos desenvolverem técnicas de balancea-

---

[1] Uma heurística é um método (ou estratégia) prático suficiente para atingir um objetivo. Elas podem ser utilizadas para acelerar o processo para atingir o objetivo.

[2] Máquinas virtuais são emulações de sistemas operacionais sobre um sistema hospedeiro, por exemplo, um sistema operacional ou *hardware*.

[3] A consolidação é uma técnica aplicada em ambientes virtualizados onde a sua implementação produz resultados significativos na eficiência do uso de recursos computacionais.

mento de máquinas virtuais, poucos consideram o dinamismo, escala e heterogeneidade de ambientes em nuvem. Motivado por tal carência, este trabalho tem como objetivo estender o *framework* desenvolvido por Weingärtner, Bräscher e Westphall (2016) apresentando uma proposta de heurística que tem como princípio distribuir máquinas virtuais que utilizam mais memória RAM em diferentes *hosts*, diminuindo a concorrência entre elas por este recurso. Isto evita que alguns servidores se encontrem subutilizados ou super utilizados, e também diminui a concorrência pelos mesmos recursos computacionais entre as máquinas virtuais. Assim, a degradação de serviço diminui, melhorando a qualidade de serviço oferecida pelo ambiente.

1.1 MOTIVAÇÃO

Segundo (FORUM, 2013), a utilização de computação em nuvem tem crescido bastante. Em virtude deste amplo crescimento dos ambientes de computação em nuvem, a complexidade de gerenciamento aumenta significativamente como apresentado por Weingärtner, Bräscher e Westphall (2015).

Atualmente, os ambientes computacionais em nuvem possuem poucas ou nenhuma heurística de balanceamento dos recursos computacionais utilizados pelas máquinas virtuais hospedadas. Nathuji, Kansal e Ghaffarkhah (2010) mencionam que a virtualização nos servidores não garante o isolamento de desempenho entre as máquinas virtuais. Por exemplo, uma aplicação que esteja utilizando um núcleo de um processador com vários núcleos, pode sofrer redução de desempenho quando outra aplicação estiver rodando simultaneamente em um núcleo adjacente. Tal comportamento ocorre devido ao aumento na taxa de falta (do inglês, *miss rate*) no último nível de *cache*. Além disso, a competição por recursos computacionais também agrava a degradação de serviço, afetando a qualidade de serviço entregue ao cliente.

Motivado pela importância de evitar a degradação de serviço nestes ambientes e o aproveitamento inteligente dos recursos computacionais em ambientes de computação em nuvem, junto da necessidade de gerenciar estes ambientes de forma eficiente e autônoma, notou-se a possibilidade do desenvolvimento de uma heurística para balancear de forma eficiente a carga computacional utilizada por estes ambientes, com o mínimo possível de intervenção humana. Esta nova heurística pode ser uma alternativa a heurística proposta em (WEINGÄRTNER; BRÄSCHER; WESTPHALL, 2016).

1.2 OBJETIVOS

Esta seção apresenta o objetivo geral e objetivos específicos deste trabalho.

1.2.1 Objetivo Geral

Este trabalho tem como objetivo criar uma heurística de balanceamento de carga computacional em ambientes computacionais em nuvem. Esta heurística tem como princípio distribuir máquinas virtuais que utilizam mais memória RAM em diferentes *hosts*, diminuindo a concorrência entre essas máquinas por este recurso. Assim, é possível melhorar a qualidade de serviço entregue por estes ambientes.

1.2.2 Objetivos Específicos

Os objetivos específicos deste trabalho são:

- Modelar uma heurística de balanceamento;
- Implementar a heurística;
- Testar e avaliar a nova heurística em um simulador de ambientes computacionais em nuvem.

1.3 ORGANIZAÇÃO DO TEXTO

O texto é organizado nos seguintes capítulos:

- Capítulo 1 - Introdução: apresenta o trabalho, sua motivação, justificativa, objetivos e organização do texto;
- Capítulo 2 - Fundamentação Teórica: apresenta os conceitos e fundamentação teórica necessários para a compreensão deste trabalho;
- Capítulo 3 - Trabalhos Relacionados: apresenta o estado da literatura referente ao balanceamento de máquinas virtuais, analisando e comparando com este trabalho;

- Capítulo 4 - Proposta: apresenta o problema e uma solução ao mesmo tendo como base as referências bibliográficas, trabalhos relacionados e ferramentas apresentadas. Esta seção também descreve o algoritmo para implementar a proposta;
- Capítulo 5 - Desenvolvimento e Análise: descreve como foi realizado o processo para implementar o algoritmo proposto e são apresentados os resultados e análises;
- Capítulo 6 - Conclusão: apresenta as conclusões finais e trabalhos futuros.

2 FUNDAMENTAÇÃO TEÓRICA

Neste capítulo são apresentadas as tecnologias e conceitos utilizados durante o desenvolvimento deste trabalho.

2.1 VIRTUALIZAÇÃO

Carissimi (2008) cita que virtualização é a técnica que permite particionar um único sistema computacional em vários outros denominados de máquinas virtuais. Cada máquina virtual oferece um ambiente completo muito similar a uma máquina física. Com isso, cada máquina virtual pode ter seu próprio sistema operacional, aplicativos e serviços de rede. É possível ainda interconectar virtualmente cada uma dessas máquinas através de interfaces de redes, *switches*, roteadores e *firewalls* virtuais, além do uso já bastante difundido de *Virtual Private Networks* (VPN).

Uma de suas principais vantagens é a possibilidade de hospedar diferentes máquinas virtuais e cada uma executa um sistema operacional. Assim, a falha de uma máquina virtual não influencia nas outras. Porém, uma consolidação de servidores como esta pode se tornar catastrófica caso todas as máquinas virtuais estejam em um único servidor e ocorra alguma falha nele. Com a virtualização, é possível salvar o estado das máquinas virtuais a cada momento e também é possível migrá-las para outros servidores (TANENBAUM, 2007).

Segundo Thorpe (2012), um virtualizador é uma das várias técnicas de virtualização que permite múltiplos sistemas operacionais, denominado convidados (*guests*), que executam em um computador hospedeiro (*host*). Thorpe (2012) cita que o virtualizador apresenta ao sistema operacional convidado uma plataforma virtual operacional e ele monitora a execução deste sistema operacional. Múltiplas instâncias de vários tipos de sistemas operacionais podem compartilhar os recursos de hardware virtualizados. O virtualizador é instalado em um servidor e sua única tarefa é executar sistemas operacionais convidados.

(ARCHER et al., 2010) e (TANENBAUM, 2007) apresentam alguns tipos de virtualização utilizadas neste trabalho. São elas:

- Virtualização por contêiner: a camada de virtualização é implementada criando-se instâncias de um sistema operacional (SO) base. Estas instâncias são ponteiros que apontam para os endereços das rotinas SO base e estes ponteiros residem em uma

memória protegida dentro do contêiner. Isto possibilita um menor consumo de memória RAM e uma ampla densidade de contêineres. Porém, o cliente deste tipo de virtualização possui somente a opção de criar instâncias com SOs idênticos ao SO base. Um exemplo de contêiner é o *LXC* (CANONICAL, 2017).

- Virtualização do tipo 1: este tipo de virtualização tem acesso direto ao hardware e pode ou não ser implementada sobre ele, sem a necessidade de um SO base para funcionar. O virtualizador executa em modo *kernel* enquanto as máquinas virtuais executam em modo usuário. Isto possibilita ao virtualizador criar abstrações de hardware para hospedar SOs distintos e intermediar a comunicação entre hardware e máquinas virtuais. Assim, o cliente pode instalar diferentes tipos de SOs em suas máquinas virtuais. Porém, como cada máquina virtual terá seu próprio SO, isto gera um maior consumo de memória do sistema se comparado ao contêiner (visto que as instâncias virtuais partilham de um mesmo sistema operacional base). Alguns virtualizadores do tipo 1 são *XenServer* (SYSTEMS, 2017) e *KVM* (ALLIANCE, 2017).

- Virtualização do tipo 2: os virtualizadores deste tipo são conhecidos como *virtualizadores hospedados*. Isto se deve ao fato de que o virtualizador necessita de um SO base onde ele é implantado. Todas as instruções sensíveis, como chamadas de sistema, são alteradas para rotinas onde o virtualizador emula estas instruções. Assim, nenhuma instrução sensível é executada pelo SO hospedeiro diretamente sobre o hardware. Este tipo de virtualização é semelhante à virtualização do tipo 1, onde é possível haver máquinas virtuais com SOs distintos. Porém, como o virtualizador se encontra sobre a camada de software, ele não possui acesso direto ao hardware. Um exemplo de virtualização do tipo 2 é a ferramenta *Virtualbox* (ORACLE, 2017).

Como é possível observar, cada tipo de virtualização tem suas vantagens e desvantagens. A virtualização por contêiner utiliza menos memória RAM, porém limita o usuário a um tipo de sistema operacional. As virtualizações de tipo 1 e tipo 2, permitem ao usuário escolher o seu sistema operacional. Entretanto, há um maior consumo de memória visto que cada máquina virtual possui um sistema operacional diferente. A virtualização do tipo 2 pode ter um maior consumo de memória pois terá o consumo de memória do sistema operacional hospedeiro e das máquinas virtuais hospedadas pelo virtualizador, porém é uma solução

interessante para o desenvolvimento de ferramentas e aprendizado, devido sua praticidade.

2.2 HYPERVISOR

O *hypervisor* é responsável pela virtualização num servidor, compartilhando os recursos físicos necessários e gerenciando as máquinas virtuais. Ele cria um *virtual machine monitor* (VMM), em português monitor de máquina virtual, para cada instância virtual, permitindo a separação lógica destas máquinas virtuais (WEINGÄRTNER; BRÄSCHER; WESTPHALL, 2016).

Segundo VMware (2007), o *virtual machine monitor* é um componente do *hypervisor* com responsabilidade na gerência, criação e destruição de uma máquina virtual, gerando o ambiente virtualizado necessário para a execução desta, atua configurando os recursos a serem disponibilizados pelo *hypervisor*.

O *hypervisor* é utilizado nas virtualizações do tipo 1 e 2. Na virtualização do tipo 1, o *hypervisor* é implantado diretamente sobre o *hardware*, permitindo um bom desempenho pois as requisições geradas pelo *hypervisor* não passam por nenhuma outra camada. Já na virtualização do tipo 2, ele é implantado em cima de um SO base. Desta forma, toda requisição gerada pelo *hypervisor* precisa passar pela camada do SO, afetando o desempenho da virtualização.

2.3 COMPUTAÇÃO EM NUVEM

Computação em Nuvem é um modelo que permite acesso onipresente, conveniente e sob-demanda à um conjunto de recursos computacionais configuráveis (por exemplo, redes, servidores, armazenamento, aplicações e serviços) que podem ser provisionados e liberados rapidamente com um mínimo de esforço de gerenciamento ou interação com o provedor de serviço (MELL; GRANCE, 2011).

A computação em nuvem atualmente pode oferecer diversos tipos de serviços em diferentes modelos. Segundo Mell e Grance (2011) os serviços de computação em nuvem podem ser classificados em três categorias: *Software as a Service* (SaaS - Software como um Serviço), *Platform as a Service* (PaaS - Plataforma como Serviço) e *Infrastructure as a Service* (IaaS - Infraestrutura como Serviço). Elas podem ser descritas da seguinte maneira:

- *Software as a Service*: capacidade provida ao consumidor de usar as aplicações do provedor executando em um ambiente de computação em nuvem. As aplicações são acessíveis a vários dispositivos de clientes através de uma interface, como navegadores web, ou de um programa. O consumidor não gerencia ou controla a infraestrutura do ambiente de computação em nuvem tal como a rede, servidores, sistemas operacionais, armazenamento, entre outros;

- *Platform as a Service*: capacidade provida ao consumidor de implantar na estrutura de computação em nuvem aplicações criadas ou adquiridas utilizando linguagens de programação, bibliotecas, serviços e ferramentas suportadas pelo provedor. O consumidor não gerencia ou controla a infraestrutura do ambiente de computação em nuvem tal como a rede, servidores, sistemas operacionais ou armazenamento, porém, ele tem o controle sobre a aplicação implantada e possivelmente as configurações do ambiente do hospedeiro da aplicação;

- *Infrastructure as a Service*: capacidade provida ao consumidor de suprir processamento, armazenamento, rede, e outros recursos computacionais fundamentais onde o consumidor pode implantar e executar softwares arbitrários, os quais podem incluir sistemas operacionais e aplicações. O consumidor não gerencia ou controla a infraestrutura física do ambiente de computação em nuvem, porém, ele tem o controle sobre os sistemas operacionais, armazenamento, e aplicações implantadas; e possivelmente pode ter um controle limitado de componentes de rede (por exemplo, o *firewall* da máquina hospedeira).

2.4 ORQUESTRAÇÃO DE AMBIENTES DE COMPUTAÇÃO EM NUVEM

Ambientes de computação em nuvem, são complexos e heterogêneos, possuindo inúmeros componentes e variáveis dinâmicas que devem ser interligadas e gerenciadas. Essas características, tornam a gerência manual destes ambientes humanamente impossível. Sendo assim, é necessária a adoção de ferramentas capazes de orquestrar a infraestrutura, auxiliando na implementação, manutenção e otimização da infraestrutura, sem o uso de um administrador (WEINGÄRTNER; BRÄSCHER; WESTPHALL, 2015) e (WEINGÄRTNER; BRÄSCHER; WESTPHALL, 2016).

Segundo Weingärtner, Bräscher e Westphall (2016), orquestração de ambientes computacionais em nuvem é o processo de criar estes ambientes. Orquestradores de ambientes computacionais em nuvem tal como o *Apache CloudStack* (FOUNDATION, 2016a) e *OpenStack* (OPENSTACK, 2016d), conectam diferentes tipos de sistemas computacionais (por exemplo, armazenamento, rede e *hypervisors*), criando a camada de abstração *IaaS* para os usuários e administradores.

Ferramentas de orquestração de ambientes computacionais em nuvem surgiram para auxiliar na gerência e orquestração destes ambientes. Elas podem auxiliar na visibilidade do ambiente (e.g estados dos *hosts* e máquinas virtuais); podem facilitar o trabalho de instanciar uma nova máquina virtual, podendo deixar o cliente fazer isso sem a ajuda de um administrador; diminuem a necessidade de intervenção física em diversas tarefas; entre outros benefícios. Durante este trabalho, foram estudadas algumas ferramentas de orquestração de ambientes computacionais em nuvem. Elas são apresentadas nas próximas subseções junto de suas arquiteturas que são explicadas brevemente.

2.4.1 Apache CloudStack

Apache CloudStack é um *software* desenvolvido para implantar e gerenciar uma grande rede de máquinas virtuais com alta disponibilidade e plataforma de computação em nuvem no modelo *IaaS* altamente escalável. Esta plataforma é utilizada por provedores de serviços para oferecer serviços públicos de computação em nuvem, ou ser parte de uma solução híbrida em computação em nuvem. Além disso, ela oferece orquestração computacional, *Network-as-a-Service*, gerenciamento de usuários e contas, *Application Programming Interface* (API)[1] nativa aberta e interface ao usuário (FOUNDATION, 2016a).

O *Apache CloudStack* possui suporte à uma vasta variedade de *hypervisors*, podendo assim, ter distintos *hypervisors* no mesmo ambiente computacional em nuvem. Além disso, ele pode gerenciar milhares de servidores físicos instalados em centros de dados geograficamente distribuídos (FOUNDATION, 2016b). Isso tudo oferece uma maior liberdade e alternativas de configurações aos provedores de serviços e usuários.

De acordo com Foundation (2016b) e como apresentado na figura 1, a infraestrutura do *CloudStack* é composta por sete recursos

[1] Uma API é um conjunto de códigos e métodos que podem ser reutilizados por outros programadores. Por exemplo, com ela é possível que programadores realizem a comunicação entre o seu código e o sistema a qual a API foi destinada. Sendo assim, o programador não precisa saber em grandes detalhes como o sistema funciona.

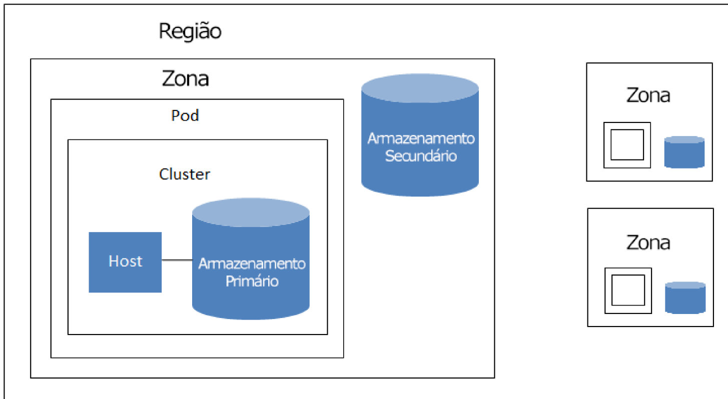

Figura 1 – Infraestrutura do CloudStack, figura traduzida de (FOUNDATION, 2016b).

gerenciáveis. São eles:

- Regiões: são coleções de uma ou mais zonas geograficamente próximas gerenciadas por um ou mais servidores de gerenciamento;

- Zonas: tipicamente, uma zona e equivalente à um único centro de dados. Uma zona consiste de um ou mais *pods* e *secondary storage*, ou, em português, armazenamento secundário;

- *Pods*: um *pod* geralmente é um *rack*, ou conjuntos de *racks* que incluem um *switch* de camada dois e um ou mais *clusters*;

- *Clusters*: um *cluster* consiste de um ou mais *hosts* homogêneo e *primary storage*, ou, em português, armazenamento primário;

- *Host*: um *host* é um nodo computável dentro de um *cluster*;

- Armazenamento Primário: um recurso de armazenamento fornecido à um único *cluster* para as instâncias de imagens de disco que estão rodando;

- Armazenamento Secundário: um recurso que armazena *templates* de disco, imagens ISO e *snapshots*. Este recurso é disponível à uma única zona, isto é, cada zona tem seu conjunto de *templates*, imagens ISO e *snapshots*.

2.4.2 OpenStack

OpenStack é um sistema operacional em nuvem que controla vários recursos como computadores, armazenamento e redes. Isto é feito através de centro de dados gerenciados através de um painel que provê controles de administrador enquanto providencia ao seus usuários recursos computacionais através de uma interface web (OPENSTACK, 2016d) .

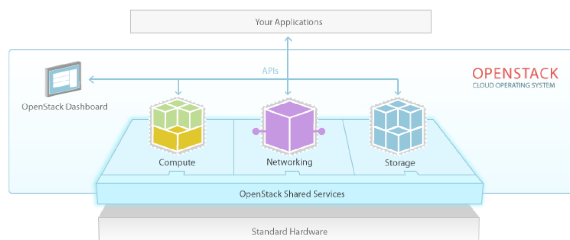

Figura 2 – Arquitetura Básica do OpenStack (OPENSTACK, 2016d).

Como é possível observar na figura 2, o *OpenStack* possui três componentes básicos, são eles:

- *OpenStack Compute*: responsável por escalonar, implantar, destruir e controlar o tempo de vida das máquinas virtuais no ambiente (OPENSTACK, 2016c).

- *OpenStack Networking:* habilita a conexão à outros serviços do *OpenStack*, como o *OpenStack Compute*. Este componente também fornece uma API aos usuários para que eles possam definir redes e configura-las. Este componente possui uma arquitetura plugável habilitando o suporte a outras redes e tecnologias (OPENSTACK, 2016b).

- *OpenStack Storage:* fornece armazenamento persistente às máquinas virtuais. Possui uma arquitetura de disco plugável a qual facilita a criação e gerenciamento dos dispositivos de armazenamento (OPENSTACK, 2016a).

2.4.3 HPE Helion Stackato

O *HPE Helion Eucalyptus* (PACKARD, 2016b) é uma solução de código aberto para criar nuvens privadas e híbridas compatíveis com as APIs dos *Amazon Web Services* (AWS) (SERVICES, 2016). Ela pode ser dinamicamente expandida ou contraída, dependendo das cargas de trabalho de aplicativos, e é adequada para nuvens empresariais. Com o *HPE Helion Eucalyptus*, as organizações podem entregar uma experiência de nuvem pública, usando recursos de TI privada para mais economia, governança de dados mais forte e desempenho do aplicativo mais rápido (PACKARD, 2016b).

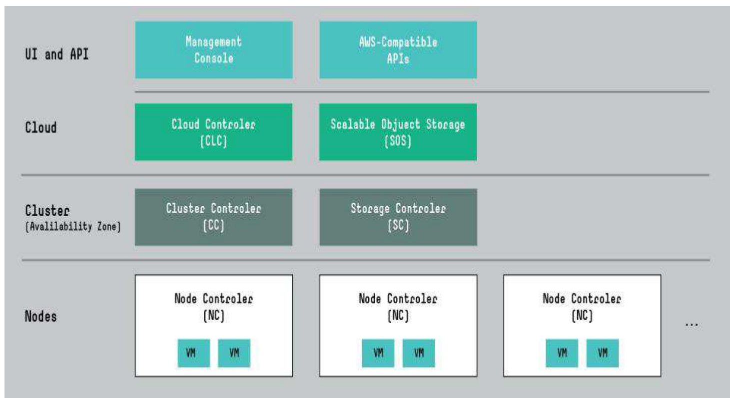

Figura 3 – Arquitetura conceitual do HPE Helion Eucalyptus (PACKARD, 2016a).

De acordo com (PACKARD, 2016a) e a figura 3, o *HPE Helion Eucalyptus* é divido em quatro camadas:

- *User Interface* (UI) e API: esta camada é composta pelos componentes *Management Console* e *AWS-Compatible APIs*. O *Management Console* é uma interface baseada em web que permite os usuários da nuvem prover e gerenciar recursos e também permite criar contas de administradores para gerenciar usuários, grupos e políticas do ambiente. Já o componente *AWS-Compatible APIs* implementa serviços web compatíveis com serviços AWS e APIs. Este componente serve para clientes e usuários interagirem com a plataforma de nuvem;
- *Cloud*: esta camada é composta pelos componentes *Cloud Con-*

troller (CLC) e *Scalable Object Storage* (SOS). O *Cloud Controller* hospeda um banco de dados para o rastreamento de recursos no ambiente de nuvem. Só é possível haver um CLC por ambiente. No*Eucalyptus 4* o CLC também trata o DNS para serviços e recursos dentro do ambiente de nuvem. Já o *Scalable Object Storage* realiza o armazenamento escalável de objetos, permitindo salvar dados dos usuários, além de armazenar as imagens e volumes de todas máquinas virtuais do ambiente de nuvem;

- *Cluster*: esta camada é composta pelos componentes *Cluster Controller* (CC) e *Storage Controller* (SC) *Storage Controller* (SC). O *Cluster Controller* atua como ingresso de rede para um *cluster* dentro do ambiente de nuvem do *Eucalyptus* e se comunica também com o *Storage Controller* (SC) e o *Node Controller* (NC). Ele também é responsável por gerenciar a execução das máquinas virtuais. O *Storage Controller* se comunica com o CC e NC que estão dentro do ambiente de nuvem e gerencia os volumes e *snapshots* das máquinas virtuais.

- *Nodes*: esta camada é composta pelo componente *Node Controller*, podendo haver um ou mais deles no mesmo ambiente de nuvem. Este componente hospeda as máquinas virtuais e gerencia as redes virtuais. Além disto, ele baixa e armazena na *cache* imagens das máquinas virtuais localizadas no armazenamento compartilhado do *cluster* e também cria e armazena máquinas virtuais no seu disco local.

2.4.4 OpenNebula

OpenNebula (OPENNEBULA, 2016a) visa prover uma camada de gerenciamento aberta, flexível, extensível e compreensível para automatizar e orquestrar operações de ambientes computacionais empresariais em nuvem alavancando e integrando soluções implantadas já existentes para redes, armazenamento,virtualização, monitoramento ou gerenciamento de usuários (OPENNEBULA, 2016a).

A plataforma *OpenNebula* provê serviços nas duas principais camadas dos Centros de Dados de Virtualização e Infraestrutura de Ambientes Computacionais em Nuvem, são eles:

- Gerenciamento de Centros de Dados de Virtualização: permite aos usuários consolidar servidores e integrar recursos de TI existentes à computação, armazenamento e rede. A plataforma *OpenNebula*

possui um suporte mais limitado à alguns *hypervisors* e possui total controle sobre os recursos computacionais físicos e virtuais, permitindo um melhor gerenciamento, otimização de recursos e alta disponibilidade (OPENNEBULA, 2016b).

- Gerenciamento de Ambientes Computacionais em Nuvem: permite aos usuários implantar um sistema similar ao de nuvem sobre uma infraestrutura de gerenciamento já existente (por exemplo, o *VMware vCenter* (VMWARE, 2016)). Isto permite aos usuários elasticidade, centro de dados virtuais, ambiente computacional em nuvem híbrido, entre outras funções que ambientes computacionais em nuvem provêm (OPENNEBULA, 2016b).

De acordo com (OPENNEBULA, 2016c), a arquitetura do *OpenNebula* assume que a infraestrutura aonde a plataforma será implantada adote a clássica arquitetura de *cluster* com *front-end*, e um conjunto de *hosts* onde cada máquina virtual irá ser executada. Deve haver pelo menos uma rede física unindo todos os *hosts* com o *front-end*. Como apresentado na figura 4, a arquitetura do *OpenNebula* é definida por três componentes: *storage*, *networking* e *virtualization*, ou em português, armazenamento, rede e virtualização. Adicionalmente, os componentes básicos do *OpenNebula* são:

- *Front-end* que executa serviços do *OpenNebula*;
- *Hosts* com *hypervisors* que providencia recursos necessários pelas máquinas virtuais;
- *Datastores* que armazenam as imagens base das máquinas virtuais;
- Redes físicas usadas para suportar os serviços básicos tais como interconexão dos dispositivos de armazenamento e o controle de operações do *OpenNebula* e as VLANs para as máquinas virtuais.

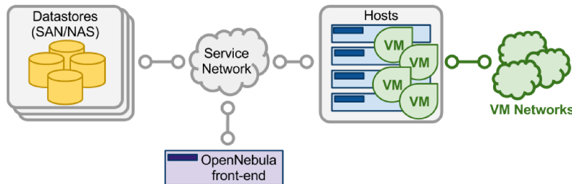

Figura 4 – Infraestrutura do OpenNebula (OPENNEBULA, 2016c).

2.5 QUALIDADE E DEGRADAÇÃO DE SERVIÇO

Qualidade de serviço (QoS) em ambientes computacionais em nuvem é o termo que define a entrega de serviços compatíveis com os requisitos do usuário. Isto é, ele pode exigir que o serviço a ser contratado por ele possua alta disponibilidade, confiabilidade e desempenho. Assim, para atender estes requisitos é necessário que o ambiente possua uma baixa taxa de degradação de serviço. A degradação de serviço pode ocasionar em baixo desempenho nas aplicações do usuário final, tomando longos períodos para responder suas requisições ou até mesmo deixando o serviço indisponível. Isto se deve ao fato de que a degradação de serviço ocorre pela sobrecarga de *hosts* e/ou a disputa por recursos computacionais entre aplicações e máquinas virtuais.

2.6 GERÊNCIA DE AMBIENTES DE COMPUTAÇÃO EM NUVEM

Um dos aspectos fundamentais da virtualização em ambientes computacionais em nuvem é a consolidação e gerência (YOUNGE et al., 2010). Diferente da orquestração de ambientes computacionais em nuvem, onde conectam-se diferentes recursos computacionais e sistemas operacionais para criar a camada *IaaS*, através da gerência e monitoramento destes ambiente, é possível tomar ações sobre eles. Um exemplo disto é a migração de máquinas virtuais entre os *hosts* para balancear o consumo dos recursos computacionais do ambiente. Younge et al. (2010) citam que através do gerenciamento destes ambientes, é possível desligar *hosts* ociosos, visto que em grandes ambientes computacionais em nuvem nunca utilizam a sua capacidade máxima, diminuindo assim o consumo energético e contribuindo com a Computação Verde (*Green Computing*).

2.6.1 Balanceamento

O balanceamento de recursos computacionais em um ambiente computacional em nuvem pode trazer um melhor aproveitamento dos recursos computacionais disponíveis no ambiente, possibilitando a diminuição da sobrecarga dos *hosts*. Isto pode ser realizado de maneiras distintas, podendo ser manual, automática ou autônoma. A diminuição da sobrecarga nos *hosts* influencia diretamente na eficiência energética e desempenho do ambiente.

Srikantaiah, Kansal e Zhao (2008) apresentam que em um ambiente consolidado, quando ocorre o balanceamento, o consumo energético não é linear ao consumo dos recursos computacionais devido à significativa porcentagem de energia ociosa. Porém, ainda pode haver a degradação de serviço pois pode ocorrer concorrência interna pelos mesmos recursos computacionais como CPU, memória RAM, utilização de disco, entre outros. Assim, um balanceamento mais inteligente é necessário.

2.7 SISTEMAS COMPUTACIONAIS AUTÔNOMOS

Kephart e Chess (2003) citam que sistemas computacionais autônomos são capazes de gerenciar a si mesmos dado um objetivo de alto nível. Para alcançar este objetivo, Kephart e Chess (2003) acreditam que o sistema precisar ter algumas características, como: auto-gerenciamento, auto-configuração, auto-otimização, auto-proteção e auto-recuperação. Estas tarefas quando realizadas por seres humanos, demandam muito tempo, degradando a qualidade de serviço e desempenho do sistema.

2.7.1 Sistemas Autônomos x Sistemas Automatizados

Os termos "automáticos" e "autônomo" parecem similares, porém, a essencial diferença é que um sistema automático precisa da intervenção humana para que sua tarefa possa ser concluída, diferente de um sistema autônomo que pode operar sozinho, isto é, sem intervenção humana. Atualmente, as plataformas de orquestração de ambientes computacionais em nuvem são automatizadas. Elas ainda necessitam de administradores humanos para configurá-las, realizar a migração de máquinas virtuais, realizar ativação e desativação de servidores. Por outro lado, em um sistema autônomo como o apresentado por Kephart e Chess (2003), poderia haver um componente de conhecimento que aprende mais a medida que o sistema executa. Assim, o sistema vai aprendendo como combater as ameaças, gerenciar, proteger, recuperar e otimizar o sistema.

2.8 PLATAFORMA AUTONOMICCS

A *Autonomiccs* (AUTONOMICCS, 2016) é uma plataforma escalonadora de máquinas virtuais distribuídas, ou, escalonadora de recursos distribuídos, desenvolvida como um *plugin* para o *Apache CloudStack*. Ela é capaz de gerenciar e otimizar um ambiente computacional em nuvem no modelo IaaS de forma autônoma (AUTONOMICCS, 2016). Esta plataforma faz o uso de heurísticas e agentes que agem no ambiente constantemente buscando a melhor maneira para atingir o objetivo da heurística implantada no ambiente. A vantagem de se utilizar agentes é que cada agente pode agir em diferentes aspectos do ambiente, dividindo o trabalho entre eles e acelerando o desempenho.

3 TRABALHOS RELACIONADOS

Neste capítulo serão apresentados trabalhos desenvolvidos na área de Computação em Nuvem que realizaram algum tipo de balanceamento de recursos computacionais semelhantes ao proposto neste trabalho. O objetivo desta seção é comparar o conteúdo desenvolvido neste trabalho às demais propostas, bem como apresentar o estado da arte nesta área de conhecimento.

Bala e Chana (2016) desenvolveram um algoritmo que tem como base algoritmos de aprendizagem de máquina (como Redes Neurais, *Random Forest*, entre outros) para a previsão do consumo de recursos computacionais de cada máquina virtual em um ambiente de nuvem. Para que verificassem se um *host* estava sobrecarregado ou ocioso, foram inseridos limites máximos e mínimos. Assim, caso um *host* estiver acima do limite de utilização de recursos, ele é classificado como "sobrecarregado", inicializando assim a migração de máquinas virtuais para outros *hosts* até ele ficar dentro dos limites estabelecidos. Caso o *host* estiver abaixo do nível mínimo estabelecido, ele é classificado como "ocioso". Assim, se inicializa a migração das máquinas virtuais para outros *hosts* e quando não houver mais máquinas virtuais nele, ele é desligado.

Em síntese, o trabalho de Bala e Chana (2016) tenta prever o consumo de recursos computacionais das máquinas virtuais para assim migrá-las ou não para outros *hosts* que sejam capazes de suportar a carga que elas irão gerar num dado momento. Apesar de o trabalho de Bala e Chana (2016) levar em consideração diferentes parâmetros como CPU, RAM, rede e Leitura e Escrita em Disco, não procura diminuir a concorrência pelos mesmos recursos entre as máquinas virtuais, como é proposto por este trabalho.

Kulkarni e Annappa (2015) apresentam em seu trabalho um algoritmo de balanceamento de máquinas virtuais alternativo ao algoritmo nativo de balanceamento da ferramenta *CloudAnalyst* (WICKREMASINGHE, 2009). O algoritmo de balanceamento nativo aloca as requisições para as máquinas virtuais menos sobrecarregadas em um dado momento para manter uma alocação uniforme para todas as máquinas virtuais. Kulkarni e Annappa (2015) aponta que o algoritmo nativo pode ocasionar uma super alocação de máquinas virtuais quando há muitas requisições ao controlador do centro de dados.

Para resolver este problema, Kulkarni e Annappa (2015) propuseram um algoritmo alternativo ao algoritmo nativo do *CloudAnalyst*.

Neste algoritmo, eles adicionaram uma tabela de reservas interna ao balanceador de máquinas virtuais para manter as recomendações de reserva de cada máquina virtual sugeridas pelo controlador do centro de dados (por exemplo, quantidade de memória RAM e CPU necessários para executar a máquina virtual). Assim, o balanceador irá levar em consideração a tabela de reservas interna e a tabela de alocação estática de uma máquina virtual em particular. O algoritmo garante a alocação uniforme até nas horas de pico.

Janpan, Visoottiviseth e Takano (2014) apresentam um *framework* de consolidação para o *CloudStack* através de *live migration*. O *framework* proposto consiste de quatro componentes: um operador do *CloudStack*, um operador de monitoramento, um operador de mapeamento e um controlador de força. O operador do *CloudStack* é responsável por interagir com a API do *CloudStack*; o operador de monitoramento é responsável por coletar dados de monitoramento de sistemas de monitoramento; o controlador de força é responsável por ligar e desligar um *host* para economizar energia; e o operador de mapeamento é responsável por mapear máquinas virtuais para outros *hosts*.

O método de consolidação proposto por Janpan, Visoottiviseth e Takano (2014) busca balancear a utilização de CPU no ambiente de computação em nuvem. A essência do algoritmo se dá quando um *host* é liberado para receber migrações de máquinas virtuais. Inicialmente, buscam-se todos os *hosts* e máquinas virtuais do ambiente. Em seguida, é realizada uma busca sobre esses *hosts* para achar aquele com o menor índice de utilização de CPU. Este índice é utilizado para determinar quando o processo de migração para o *host* alvo deve ser finalizado. Então, inicia-se o processo de migração de máquinas virtuais buscando aqueles *hosts* com os maiores índices de utilização de CPU. A escolha da máquina virtual dentro destes *hosts* é feita aleatoriamente. Estes dois últimos passos são repetidos até que o índice de utilização de CPU no *host* alvo seja maior que o limite determinado. Além deste método, Janpan, Visoottiviseth e Takano (2014) apresentaram um método de desconsolidação responsável por migrar as máquinas virtuais de um *host* que será desligado para outros *hosts*.

Beloglazov e Buyya (2012) apresentam uma proposta de heurísticas adaptativas para consolidação dinâmica de máquinas virtuais. As heurísticas se baseiam de acordo com a análise dos históricos de uso dos recursos computacionais de cada máquina virtual. Para realizar a consolidação do ambiente, quatro fatores são levados em consideração: determinar quando um *host* é considerado sobrecarregado, necessitando

um balanceamento de carga; determinar quando um *host* é considerado ocioso, podendo migrar suas máquinas virtuais e ser desligado para economizar energia; selecionar quais máquinas virtuais devem ser migradas de um *host* sobrecarregado; e encontrar um *host* alvo para receber as máquinas virtuais de um *host* sobrecarregado.

O algoritmo desenvolvido em (BELOGLAZOV; BUYYA, 2012), de forma resumida, busca inicialmente a lista de todos os *hosts* do ambiente e verifica quais estão sobrecarregados. Caso um *host* esteja sobrecarregado, o algoritmo executa a seleção das máquinas virtuais a serem migradas de acordo com a política adotada. Após construir a lista de máquinas virtuais que necessitam ser migradas, buscam-se os *hosts* que estão ociosos para receber as máquinas virtuais. O algoritmo então retorna um *map* contendo as informações dos novos locais das máquinas virtuais que foram migradas. Para a verificar se um *host* está sobrecarregado ou não, é considerado o uso de CPU dos *hosts*, porém, o limite utilizado para considerar se um *host* está sobrecarregado ou não é dinâmico. A ideia principal deste limite dinâmico é ajustar o índice limite dependendo do tamanho do desvio absoluto da utilização da CPU. Quanto maior este desvio, menor o índice limite de utilização, quanto maior o desvio, maior a probabilidade da utilização da CPU atingir 100%.

4 PROPOSTA

A computação em nuvem tem ganhado cada vez mais destaque e adesão nos últimos anos devido, principalmente, ao seu baixo custo de alocação e gerenciamento. O custo de um serviço de nuvem comparado à compra, gerenciamento e manutenção de máquinas físicas, seria muito mais alto. Sendo assim, as empresas acabam migrando seus serviços para a nuvem. Apesar dos serviços oferecidos hoje através da computação em nuvem terem um alto desempenho, a concorrência entre máquinas virtuais por recursos computacionais pode ocasionar degradação do serviço oferecido.

Como apresentado por (WEINGÄRTNER; BRÄSCHER; WESTPHALL, 2015) e (GERONIMO et al., 2013), o crescimento dos ambientes computacionais em nuvem e serviços, acaba gerando a necessidade de aumentar a estrutura necessária para hospedar esses serviços, aumentando a complexidade de gerenciamento destes ambientes.

Analisando e estudando os trabalhos relacionados, constatou-se a necessidade de novas formas de consolidação dos ambientes computacionais em nuvem para melhorar o desempenho e eficiência energética dos mesmos. Como apresentado anteriormente nos trabalhos relacionados, existem diferentes propostas para a consolidação dos ambientes computacionais em nuvem, porém, nenhum deles leva em consideração a concorrência por recursos computacionais.

A proposta deste trabalho é minimizar a concorrência por memória RAM entre as máquinas virtuais dentro de um *cluster*. Isto será realizado através de migrações de máquinas virtuais entre os *hosts* buscando deixar o consumo de memória RAM o mais balanceado possível. O escopo deste trabalho se limita apenas à memória RAM devido ao fato de que ao acrescentar outros fatores, como utilização de CPU ou leitura e escrita em disco, por exemplo, aumentaria significativamente a complexidade deste trabalho, indo além de um trabalho de conclusão de curso.

4.1 ILUSTRAÇÃO DO PROBLEMA

Em um ambiente computacional em nuvem que carece de técnicas inteligentes de balanceamento de carga computacional, existe a possibilidade de ocorrer sobrecarga dos *hosts*, influenciando diretamente o desempenho dos serviços entregues através deste ambiente. A figura

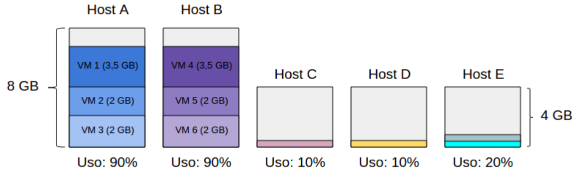

Figura 5 – Ilustração do problema a ser corrigido.

5 apresenta um ambiente heterogêneo onde a quantidade de recursos disponibilizados pelos *hosts* são diferentes. Como é possível analisar, os *hosts* A e B possuem $8GB$ de memória RAM, enquanto os *hosts* C, D e E possuem $4GB$ de memória RAM. Ainda nesta figura, é possível observar um eventual problema que pode acontecer nestes ambientes. Os *hosts* A e B estão com 90% de sua capacidade de memória RAM em uso, enquanto os servidores C, D e E estão com sua taxa de utilização de memória RAM entre 10% e 20% em uso.

A figura 6 representa a proposta deste trabalho. Considerando um ambiente computacional em nuvem heterogêneo como apresentado anteriormente, a heurística de balanceamento irá buscar equilibrar a utilização de memória RAM entre os *hosts* do ambiente. Isto será realizado através de migrações de máquinas virtuais dos *hosts* sobrecarregados para os *hosts* com o menor índice de utilização de memória RAM da sua capacidade total. Como é possível analisar na figura 6, para equilibrar a utilização de memória RAM apresentada na figura 5, as máquinas virtuais $VM3$ e $VM5$ que estavam hospedadas nos *hosts* A e B, respectivamente, foram migradas para os *hosts* C e D. Então, a utilização de memória RAM nos *hosts* A e B passaram de 90% para 62%, enquanto que nos *hosts* C e D, passaram de 10% para 58%. Assim, houve uma descarga de 28% dos *hosts* A e B e um aumento de utilização de memória RAM de 48% nos *hosts* C e D, equilibrando a utilização de memória RAM no ambiente e diminuindo a concorrência por este recurso.

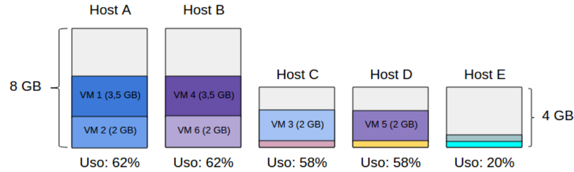

Figura 6 – Ilustracao da proposta deste trabalho.

4.2 ARQUITETURA DO *FRAMEWORK* AUTONOMICCS

O *framework Autonomiccs* foi desenvolvido para atuar no nível de *cluster*, onde múltiplos agentes podem gerenciar simultaneamente ambientes de computação em nuvem grandes e dinâmicos. Cada algoritmo não opera sobre a infraestrutura como um todo, mas sim sobre fragmentos da mesma, permitindo uma aproximação distribuída (WEINGÄRTNER; BRÄSCHER; WESTPHALL, 2016). Este trabalho se encontra na parte "Allocation Manager", como apresenta a figura 7, onde o agente irá executar a heurística desenvolvida neste trabalho periodicamente.

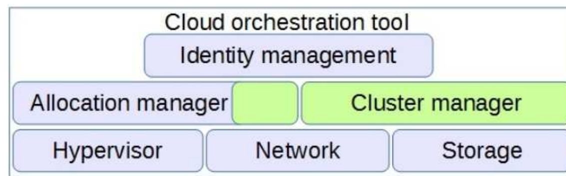

Figura 7 – Arquitetura da orquestração em nuvem com o *framwork Autonomiccs* (WEINGÄRTNER; BRÄSCHER; WESTPHALL, 2016).

Como apresentado na figura 7, o *framework* foi desenvolvido para ser parte de ferramentas de orquestração. Assim, foi adicionado um novo componente chamado *cluster manager* que é responsável por executar as migrações das máquinas virtuais e desligamento de servidores ociosos. Além disso, foi necessário estender o comportamento do *allocation manager* que é responsável pelas alocações das máquinas virtuais quando são criadas, inicializadas ou migradas (WEINGÄRTNER; BRÄSCHER; WESTPHALL, 2016).

A carência de alternativas de ferramentas de gerência autônoma de código aberto junto da facilidade e eficiência de se implementar novas

heurísticas de balanceamento ao *framework* para trabalhar sobre um ambiente de nuvem, motivaram a escolha de utilizá-lo neste trabalho, acelerando a implementação e análise da nova heurística desenvolvida.

4.3 ALGORITMO

Nesta seção será apresentado o algoritmo da heurística e seus principais métodos.

O algoritmo 1 irá executar periodicamente em cada *cluster* isoladamente e serão necessárias algumas informações referente aos níveis de estrutura do ambiente no qual ele irá atuar. Seguindo do nível mais alto para o mais baixo, as informações necessárias serão:

- Cluster: será necessário saber a alocação média de memória RAM[1] no momento em que o algoritmo rodar. Esta média será utilizada de referência para balancear a alocação de memória RAM dos *hosts* ativos que atuam dentro do *cluster*;

- Servidor: será necessário saber a alocação de memória RAM do *host* para verificar se existe a necessite de balanceá-lo ou não; e

- Máquinas Virtuais: será necessário saber a quantidade de memória RAM alocada para cada máquina virtual num dado *host* para verificar a possibilidade de migração de uma dada máquina virtual para outro *host*.

Para uma melhor apresentação do algoritmo 1 (balanceamentoDeMemoriaRAM), ele foi subdivido em partes menores para facilitar o entendimento. Inicialmente, na *linha 1* é verificada a existência de *hosts* ativos no *cluster*. Caso não houverem *hosts* ativos, não há necessidade de balancear o *cluster*. Então, é finalizado o algoritmo *balanceamentoDeMemoriaRAM*, como mostra a *linha 17*. Caso contrário, isto é, caso existam *hosts* ativos, busca-se a média de alocação de memória RAM do *cluster*. Esta média será utilizada como parâmetro para balancear a alocação de memória RAM dos *hosts* contidos neste *cluster*.

[1] É importante ressaltar que neste ponto foi escolhido trabalhar com a alocação de memória RAM ao invés da utilização atual deste recurso. Trabalhar com o uso dinâmico de memória RAM é mais arriscado pois ele pode oscilar muito, o que torna difícil a previsão e controle do mesmo. Caso ele extrapole demais, ele pode causar uma série de erros no *host*, podendo comprometê-lo. Por outro lado, trabalhar com o que foi alocado para a máquina virtual, garantirá que ela nunca irá utilizar mais memória RAM do que lhe foi permitido, evitando assim, possíveis erros e problemas.

A partir da *linha 3* à *linha 9*, é realizada uma busca entre os servidores ativos verificando quais precisam ser balanceados e quais estão abaixo da alocação média de RAM do *cluster*, podendo receber máquinas virtuais de outros *hosts*.

Na *linha 4*, é realizada uma verificação para garantir que dentre os *hosts* ativos existe a necessidade de balanceamento de memória RAM entre eles, isto é, se existe alguma sobrecarga na alocação de memória RAM em algum *host* em relação à média de alocação de memória RAM do *cluster*. Caso negativo, não há necessidade de balancear o *cluster*, finalizando assim o algoritmo como mostra a *linha 14*. Porém, caso for constatada a sobrecarga de ao menos um *host*, então o algoritmo 2 (balanceiaHostsComMigracoes) é executado.

Algorithm 1 balanceamentoDeMemoriaRAM

Entrada: $cluster, hostsAtivos$

```
1:  if hostsAtivos.size()>0 then
2:     media ← obterAlocaoMediaDeRAM(cluster)
3:     for host : hostsAtivos do
4:        if host.mediaDeAlocacaoDeRAM>media then
5:           hostsComMigracoes.add(host)
6:        else
7:           possiveisHostsAlvo.add(host)
8:        end if
9:     end for
10:    if hostsComMigracoes.size()>0 then
11:       balanceiaHostsComMigracoes(hostComMigracoes,
12:       possiveisHostsAlvo, media)
13:    else
14:       terminaAlgoritmo
15:    end if
16: else
17:    terminaAlgoritmo
18: end if
```

O algoritmo 2 recebe como entrada duas listas de *hosts*: *hostsComMigracoes* e *possiveisHostsAlvo*; e também recebe a média de alocação de memória RAM. Inicialmente a lista *hostsComMigracoes* é ordenada por alocação de memória RAM em ordem decrescente, assim, os primeiros *hosts* da lista serão os mais sobrecarregados em questão de alocação de memória RAM. Em seguida, a lista *possiveisHostsAlvo* é organizada em ordem crescente de alocação de memória RAM, assim,

os primeiros *hosts* são os com menores índices de alocação de memória RAM.

Em seguida, para cada *host* que foi adicionado à lista *hostsComMigracoes*, buscam-se as máquinas virtuais ativas do servidor e as ordena por alocação de memória RAM. Assim que todas as máquinas virtuais ativas foram identificadas, o algoritmo 3 é chamado para cada delas. Após percorrer toda a lista de *vmsAtivas*, o algoritmo 2 é finalizado.

Algorithm 2 balanceiaHostsComMigracoes

Entrada: $hostsComMigracoes, possiveisHostsAlvo, media$

1: $hostsComMigracoes \leftarrow ordenaPorAlocacaoDeRAMdecrescente()$
2: $possiveisHostsAlvo \leftarrow ordenaPorAlocacaoDeRAMcrescente()$
3: **for** $host : hostsComMigracoes$ **do**
4: $\quad vmsAtivas \leftarrow buscaVMsAtivas(host)$
5: $\quad$ **if** $vmsAtivas.size() > 0$ **then**
6: $\quad\quad vmsAtivas \leftarrow ordenaPorAlocacaoDeRAM(vmsAtivas)$
7: $\quad\quad$ **for** $vm : vmsAtivas$ **do**
8: $\quad\quad\quad migraVMs(vm, possiveisHostsAlvo, media)$
9: $\quad\quad$ **end for**
10: $\quad$ **else**
11: $\quad\quad$ quebra laço
12: $\quad$ **end if**
13: **end for**

O algoritmo 3 (*migraVMs*) recebe como entrada a lista de *possiveisHostsAlvo*, a máquina virtual *vm* a ser migrada e a média de utilização de RAM do *cluster*. O seu principal papel é buscar um *host* ativo que está contido na lista *possiveisHostsAlvo* que seja capaz de suportar a máquina virtual *vm* e que quando migrá-la para este *host* alvo, o seu índice de utilização de RAM deve ser menor que a média de utilização de RAM do *cluster*. A linha 9 se mostrou importante pois após o *host* alvo receber a máquina virtual, ele pode estar mais carregado que os outros *hosts* alvos da lista. Assim, é feita uma nova ordenação desta lista. O algoritmo 3 é finalizado quando o *host* esta balanceado, ou todas as máquinas virtuais ativas foram balanceadas, como mostra a *linha 16*.

Algorithm 3 migraVMs

Entrada: $vm, possiveisHostsAlvo, media$

1: **if** $possiveisHostsAlvo.size()>0$ **then**
2: $hostAlvo \leftarrow possiveisHostsAlvo[head]$
3: $alocacaoDeRAMHostAlvo \leftarrow hostAlvo.buscaAlocacaoDeRAM()$
4: $maximoRAMHostAlvo \leftarrow hostAlvo.getRAM()$
5: $vmRAM \leftarrow vm.RAMAlocada()$
6: $tR \leftarrow vmRAM + alocacaoDeRAMHostAlvo$
7: **if** $tR<media$ && $tR<maximoRAMHostAlvo$ **then**
8: migra vm para hostAlvo
9: $possiveisHostsAlvo \leftarrow ordenaPorAlocacaoDeRAMcrescente()$
10: **else**
11: pega proximo elemento de possiveis host
12: **end if**
13: **end if**
14: $alocacao \leftarrow host.alocacaoDeRAM()$
15: $numeroVMs \leftarrow vmsAtivas.size()$
16: **if** $alocacao<media \,||\, numeroVMs \leq 0$ **then**
17: quebra laço
18: **end if**

5 DESENVOLVIMENTO E ANÁLISE

Neste capítulo, será discutida a implementação do algoritmo proposto, apresentando as tecnologias envolvidas e o processo para implementar o algoritmo.

Para implementar o algoritmo proposto, foi escolhida a plataforma *Autonomiccs* (AUTONOMICCS, 2016), que, como apresentada anteriormente, é uma plataforma escalonadora de máquinas virtuais distribuídas, ou, escalonadora de recursos distribuídos, desenvolvida como um *plugin* para o *Apache CloudStack* através linguagem de programação *Java* (ORACLE, 2017). Com ela, é possível realizar simulações de ambientes computacionais em nuvem através do *cloud-traces*.

O *cloud-traces* é um projeto que recebe dados de traços da *Google* (GOOGLE, 2017) (*Google data traces*) que são *jobs* (serviços) orientados à cenário e os transforma para um cenário de ambiente computacional em nuvem (AUTONOMICCS, 2017). Assim, o *cloud-traces* consegue realizar ações de consolidação, como utilizar heurísticas aplicadas ao ambiente.

5.1 AJUSTES PRELIMINARES E IMPLEMENTAÇÃO

Como a estrutura da plataforma *Autonomiccs* junto do *cloud-traces* é modular, isto facilita o trabalho de implementação e ajustes na plataforma. Assim, para implementar a proposta deste trabalho, foi necessário criar uma classe chamada *ClusterVMsBalancingByRAMusage.java* no pacote *br.com.autonomiccs.cloudTraces.algorithms.manage ment*. Este pacote contém as heurísticas disponíveis para os gerentes de alocação e consolidação. Foi necessário realizar alguns ajustes na classe principal do *cloud-traces* para que ele utilizasse somente a heurística desenvolvida neste trabalho, assim como foram realizados outros pequenos ajustes para que a plataforma funcionasse.

Assim, na classe *ClusterVMsBalancingByAllocatedRAM.java* que se encontra como apêndice A, foram implementados alguns métodos:

- ***rankHosts:*** função exigida pela classe *ClusterAdministrationAlgorithmEmptyImpl* para organizar os *hosts* de acordo com algum parâmetro;
- ***mapVMsToHost:*** função que agrega a heurística de balanceamento proposta neste trabalho, referente ao algoritmo 1;
- ***updateHostResources:*** atualiza os recursos (memória RAM e

CPU) do *host* sobrecarregado e do *host* alvo de acordo com os recursos alocados pela máquina virtual que será migrada do *host* sobrecarregado para o *host* alvo;

- ***calculateClusterAllocatedRAMaverage:*** calcula a média de alocação de memória RAM realizada pelos *hosts* ativos no *cluster*;

- ***sortHostsByAllocatedRAMdecreasing:*** organiza uma lista de *hosts* passada como parâmetro em ordem decrescente de acordo a quantidade de memória RAM alocada por cada *host.* É possível observar este método na linha 1 do algoritmo 2;

- ***sortHostsByAllocatedRAMascending:*** organiza uma lista de *hosts* passada como parâmetro em ordem crescente de acordo com a quantidade de memória RAM alocada por cada *host.* É possível observar este método na linha 2 do algoritmo 2;

- ***sortVMsByRAMallocatedAscending:*** organiza uma lista de máquinas virtuais passada como parâmetro em ordem crescente de acordo com a alocação[1] de memória especificada para cada uma. É possível observar este método na linha 6 do algoritmo 2;

- ***verifyPossibleOverload:*** função que verifica se um dado *host* irá ficar sobrecarregado com a migração de uma dada máquina virtual para ele de acordo com a média de uso de memória RAM do *cluster*; e

- ***calculateTotalResources:*** função responsável por buscar o número de máquinas virtuais contidas no *cluster*, calcula o total de memória e o uso médio de memória RAM do *cluster.*

Para realizar a ordenação crescente e decrescente de *hosts* e máquinas virtuais de acordo a quantidade de memória RAM alocada, foram criadas duas classes: *VirtualMachineComparator* e *HostComparator.* Os métodos criados para analisar o comportamento da heurística e verificar possíveis erros foram ocultados a fim de focar nos resultados e desempenho da heurística desenvolvida.

[1] Lembrar que foi utilizada a alocação de memória como métrica pois a utilização do uso de memória RAM num dado momento pode ser difícil de tratar como foi apresentado anteriormente.

5.2 RESULTADOS E ANÁLISE

Para realizar estudos de comparação e análises da heurística desenvolvida neste trabalho, foi utilizada uma outra heurística preexistente chamada *ClusterAdministrationAlgorithmEmptyImpl.* Esta heurística não realiza técnicas de gerenciamento de ambientes computacionais em nuvem, ela apenas cria e destrói máquinas virtuais de acordo com as exigências do ambiente.

A heurística desenvolvida neste trabalho, realiza migrações de máquinas virtuais entre os *hosts* ativos de um determinado *cluster*. O parâmetro de referência utilizado para migrar as máquinas virtuais, é a alocação de memória RAM de cada *host* comparado com a média de memória RAM alocada pelos *hosts* dentro de um *cluster*. Caso a média de alocação de memória RAM de um *host* for maior que a média de alocação de memória RAM do *cluster*, então é realizada uma busca por um *host* alvo que esteja apto a receber alguma máquina virtual, diminuindo a sobrecarga do *host* sobrecarregado.

Assim, tem-se alguns fatores, seja $M(c)$ a média de alocação de memória RAM do *cluster* c, e R a quantidade de memória RAM alocada pelos *hosts* ativos no *cluster* c e n o número de *hosts* ativos no *cluster* c, então:

$$M(c) = \frac{R}{n}$$

Seja vr a quantidade de alocação de memória RAM requisitada pela máquina virtual, R a quantidade de memória RAM alocada no *host* h, tR o total de memória RAM alocada, e tH o total de memória RAM[2] do *host*, então:

$$tR = vr + R(h)$$

Logo, para saber se o *host* alvo ficará sobrecarregado em relação a $M(c)$, levando apenas a memória RAM em consideração, são feitas duas verificações: caso $tr > M(c)$ ou $tr > tH$, então o *host* ficará sobrecarregado.

---

[2]O total de memória RAM de um *host* ou *cluster* é a quantidade máxima de memória RAM que um *host* ou *cluster* pode ter, por exemplo, $32GB$. A quantidade de memória RAM alocada é uma reserva de memória RAM feita por um *host* ou máquina virtual do total de memória RAM. Ou seja, caso uma máquina virtual for alocar memória RAM num *host* que tem o total de memória RAM de $32GB$, então, esta máquina virtual pode alocar até $32GB$ de memória RAM.

Então, para realizar os testes foi realizada uma simulação sobre um arquivo de entrada utilizando a heurística *ClusterAdministrationAlgorithmEmptyImpl*, e em seguida, foi realizada uma nova simulação sobre este mesmo arquivo de entrada utilizando a heurística *ClusterVMsBalancingByAllocatedRAM*. No final de cada simulação é gerado um *log* de saída com diversas informações, onde é possível filtrar o tempo de execução e média de memória RAM alocada pelo tempo. A figura 8 apresenta os resultados de ambas as heurísticas.

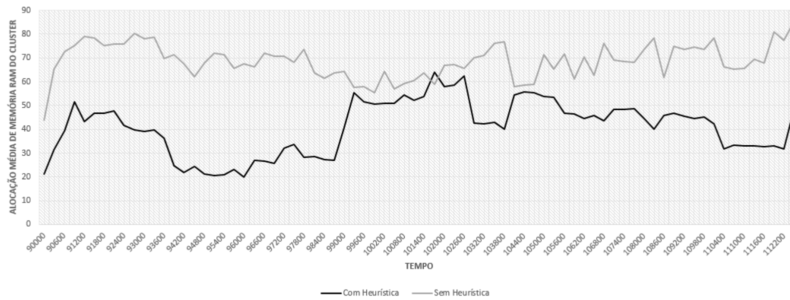

Figura 8 – Resultado de alocação de RAM x Tempo das heurísticas

Na figura 8, no eixo horizontal tem-se o *Tempo* e no eixo vertical tem-se a *Média de Alocação de Memória do Cluster*. Como é possível observar, a simulação mostrou que quando executa sem uma heurística de balanceamento de memória RAM, a alocação média de memória RAM foi de 68.73%, onde teve um pico máximo de aproximadamente 87%. Em um ambiente onde a média de alocação de memória RAM chega à 87% compromete a qualidade de serviço entregue ao usuário. Como é possível observar, a heurística desenvolvida neste trabalho mostrou um ótimo resultado, tendo uma alocação média de memória RAM de 42.74%, contra 68.73%, apresentando uma otimização de 37.81%. Com isso a concorrência por memória RAM entre máquinas virtuais diminui, deixando os *hosts* menos sobrecarregado neste requisito e melhorando a qualidade de serviço entregue ao usuário.

6 CONCLUSÃO E CONSIDERAÇÕES FINAIS

Este trabalho resultou no desenvolvimento de uma heurística de balanceamento de memória RAM em um ambiente computacional em nuvem capaz de ser implementada em qualquer ambiente que contenha a ferramenta de orquestração *CloudStack*, visto que a plataforma *Autonomiccs* é direcionada a ela.

A heurística desenvolvida, apesar de se apresentar simples, se mostrou bastante eficiente, apresentando ótimos resultados como foi mostrado.

Este trabalho contém as seguintes contribuições:

- Diminuiu a necessidade de intervenção manual para o balanceamento de memória RAM em um ambiente de computação em nuvem;

- Agilizou o processo de balanceamento de memória RAM em um ambiente de computação em nuvem; e

- Melhoria e otimização na alocação de memória RAM em ambientes computacionais em nuvem.

As principais contribuições, em relação às ideias propostas pelos trabalhos relacionados apresentados, está na busca de otimização de apenas um recurso computacional, que no caso deste trabalho foi a memória RAM. A heurística busca organizar os *hosts* mais sobrecarregados em primeiro lugar e em outra lista de *hosts* alvos busca-se colocar os menos sobrecarregados primeiro. Assim, empreende-se descarregar os *hosts* mais sobrecarregados primeiro transferindo sua sobrecarga para o menos sobrecarregados. Outra característica importante de ressaltar, é o fato de que uma melhor distribuição de máquinas virtuais que utilizam muita memória RAM entre *hosts* diferentes faz com que a concorrência por este recurso tenda a ser menor.

Como trabalhos futuros têm-se:

- Adicionar um histórico de consumo de recursos computacionais de cada máquina virtual. Assim, seria possível criar heurísticas mais inteligentes que se baseassem em um consumo mais dinâmico e previsível;

- Adicionar uma função que permitisse colocar um valor limite de consumo de um dado recurso computacional, tal que quando uma

máquina virtual atingisse este limite, um aviso seria lançado no sistema para que ele possa realizar ações para amenizar o problema. Por exemplo, colocar um limite de consumo de 60% de memória RAM. Caso uma máquina virtual atinja este limite, o sistema pode ativar uma heurística de balanceamento para resolver o problema;

- Adicionar um aviso de sucesso ou falha de migração de máquinas virtuais ao *CloudStack*;
- Alterar o foco da heurística para balanceamento de CPU e verificar os resultados; e
- Produzir um artigo científico apresentado este trabalho à comunidade.

REFERÊNCIAS

ALLIANCE, O. V. **Kernel Virtual Machine**. 2017. Disponível em: <https://www.linux-kvm.org/page/Main_Page>.

ARCHER, M. et al. **System x Virtualization Strategies**. 2010. Acesso em 29 de abr de 2016. Disponível em: <https://lenovopress.com/redp4480>.

AUTONOMICCS. **Autonomiccs**. 2016. Disponível em: <https://github.com/Autonomiccs/autonomiccs-platform>.

AUTONOMICCS. **cloud-traces**. 2017. Disponível em: <https://github.com/GabrielBrascher/cloud-traces>.

BALA, A.; CHANA, I. Prediction-based proactive load balancing approach through vm migration. **Engineering with Computers**, Springer, p. 1–12, 2016.

BELOGLAZOV, A.; BUYYA, R. Optimal online deterministic algorithms and adaptive heuristics for energy and performance efficient dynamic consolidation of virtual machines in cloud data centers. **Concurrency and Computation: Practice and Experience**, Wiley Online Library, v. 24, n. 13, p. 1397–1420, 2012.

CANONICAL. **LXC**. 2017. Disponível em: <https://linuxcontainers.org/lxc/introduction/>.

CARISSIMI, A. Virtualização: da teoria a soluções. **Minicursos do Simpósio Brasileiro de Redes de Computadores–SBRC**, v. 2008, p. 173–207, 2008.

FORUM, T. C. I. **UK Cloud adoption and trends for 2013**. 2013. Disponível em: <http://cloudindustryforum.org/white-papers/uk-cloud-adoption-and-trends-for-2013>.

FOUNDATION, T. A. S. **Apache CloudStack**. 2016. Disponível em: <https://cloudstack.apache.org/>.

FOUNDATION, T. A. S. **Concepts and Terminology**. 2016. Disponível em: <http://docs.cloudstack.apache.org/en/latest/concepts.html>.

GERONIMO, G. A. et al. Provisioning and resource allocation for green clouds. In: **12th International Conference on Networks (ICN)**. [S.l.: s.n.], 2013.

GOOGLE. **About Google**. 2017. Disponível em: <https://www.google.com.br/intl/pt-BR/about/>.

JANPAN, T.; VISOOTTIVISETH, V.; TAKANO, R. A virtual machine consolidation framework for cloudstack platforms. In: IEEE. **The International Conference on Information Networking 2014 (ICOIN2014)**. [S.l.], 2014. p. 28–33.

KEPHART, J. O.; CHESS, D. M. The vision of autonomic computing. **Computer**, IEEE, v. 36, n. 1, p. 41–50, 2003.

KULKARNI, A. K.; ANNAPPA, B. Load balancing strategy for optimal peak hour performance in cloud datacenters. In: IEEE. **Signal Processing, Informatics, Communication and Energy Systems (SPICES), 2015 IEEE International Conference on**. [S.l.], 2015. p. 1–5.

MELL, P.; GRANCE, T. The nist definition of cloud computing. Computer Security Division, Information Technology Laboratory, National Institute of Standards and Technology Gaithersburg, 2011.

NATHUJI, R.; KANSAL, A.; GHAFFARKHAH, A. Q-clouds: managing performance interference effects for qos-aware clouds. In: ACM. **Proceedings of the 5th European conference on Computer systems**. [S.l.], 2010. p. 237–250.

OPENNEBULA. **About the OpenNebula Project**. 2016. Disponível em: <http://opennebula.org/about/project/>.

OPENNEBULA. **About the OpenNebula Technology**. 2016. Disponível em: <http://opennebula.org/about/technology/>.

OPENNEBULA. **Open Cloud Architecture**. 2016. Disponível em: <http://docs.opennebula.org/5.0/deployment/cloud_design/open_cloud_architecture.html>.

OPENSTACK. **CINDER**. 2016. Disponível em: <http://www.openstack.org/software/releases/mitaka/components/cinder>.

OPENSTACK. **NEUTRON**. 2016. Disponível em: <http://www.openstack.org/software/releases/mitaka/components/neutron>.

OPENSTACK. **NOVA**. 2016. Disponível em: <http://www.openstack.org/software/releases/mitaka/components/nova>.

OPENSTACK. **What is OpenStack?** 2016. Disponível em: <http://www.openstack.org/software/>.

ORACLE. **Obtenha Informacoes sobre a Tecnologia Java**. 2017. Disponível em: <https://www.java.com/pt_BR/about/>.

ORACLE. **Welcome to VirtualBox.org!** 2017. Disponível em: <https://www.virtualbox.org/>.

PACKARD, H. **General Purpose Reference Architecture: HPE Helion Eucalyptus**. 2016. Disponível em: <https://www.hpe.com/h20195/V2/getpdf.aspx/4AA6-2547ENW.pdf?ver=1.0>.

PACKARD, H. **HPE Helion Eucalyptus**. 2016. Disponível em: <http://www8.hp.com/br/pt/cloud/helion-eucalyptus.html>.

SERVICES, A. W. **Amazon Web Services**. 2016. Disponível em: <https://aws.amazon.com/pt/>.

SRIKANTAIAH, S.; KANSAL, A.; ZHAO, F. Energy aware consolidation for cloud computing. In: SAN DIEGO, CALIFORNIA. **Proceedings of the 2008 conference on Power aware computing and systems**. [S.l.], 2008. v. 10, p. 1–5.

SYSTEMS, C. **ABOUT XENSERVER**. 2017. Disponível em: <https://xenserver.org/about-xenserver-open-source.html>.

TANENBAUM, A. S. **Modern Operating Systems**. [S.l.]: Pearson, 2007. ISBN 0136006639.

THORPE, S. Virtual machine history model framework for a data cloud digital investigation. **Journal of Convergence**, v. 3, n. 4, 2012.

VMWARE. **Understanding Full Virtualization, ParaVirtualization, and Hardware Assist**. http://www.vmware.com/resources/techresources/1008: [s.n.], 2007.

VMWARE. **vCenter Server**. 2016. Disponível em: <http://www.vmware.com/br/products/vcenter-server.html>.

WEINGÄRTNER, R.; BRÄSCHER, G. B.; WESTPHALL, C. B. Cloud resource management: A survey on forecasting and profiling models. **Journal of Network and Computer Applications**, Academic Press, v. 47, p. 99–106, 2015.

WEINGÄRTNER, R.; BRÄSCHER, G. B.; WESTPHALL, C. B. A distributed autonomic management framework for cloud computing orchestration. In: **2016 IEEE World Congress on Services (SERVICES)**. [S.l.: s.n.], 2016. p. 9–17.

WICKREMASINGHE, B. **CloudAnalyst: A CloudSim-based Tool for Modelling and Analysis of Large Scale Cloud Computing Environments**. 2009. Disponível em: <http://www.cloudbus.org/students/MEDC_Project_Report_Bhathiya_318282.pdf>.

YOUNGE, A. J. et al. Efficient resource management for cloud computing environments. In: IEEE. **Green Computing Conference, 2010 International**. [S.l.], 2010. p. 357–364.

APÊNDICE A – Código da Heurística Desenvolvida

ClusterVMsBalancingByAllocatedRAM.java

```
 1 package br.com.autonomiccs.cloudTraces.algorithms.management;
 2
 3 import java.util.ArrayList;
16
17 public class ClusterVMsBalancingByAllocatedRAM extends
   ClusterAdministrationAlgorithmEmptyImpl {
18
19     private double totalClusterRAM;                  //In MB
20     protected long clusterRAMallocatedAverage = 0;   //In MB
21     protected int numberOfVms = 0;
22     protected static final long NUMBER_OF_BYTES_IN_ONE_MEGA_BYTE = 1024l;
23
24     @Override
25     public List<Host> rankHosts(List<Host> hosts) {
26         return sortHostsByAllocatedRAMdecreasing(hosts);
27     }
28
29     /**
30      * Balance the cluster based on the allocated RAM of each host in it.
31      * @param hostsAtivos list of active hosts in the cluster.
32      * @param clusterRAMallocatedAverage RAM average used by the hosts.
33      */
34     @Override
35     public Map<VirtualMachine, Host> mapVMsToHost(List<Host> rankedHosts) {
36         Map<VirtualMachine, Host> mapOfMigrations = new HashMap<>();
37         List<Host> possibleTargets = new ArrayList<>();
38         List<Host> hostsToBeBalanced = new ArrayList<>();
39         calculateTotalResources(rankedHosts);
40
41         //Verifica os hosts que precisam ser balanceados e os adiciona numa lista
   hostsToBeBalanced
42         for (Host host : rankedHosts) {
43             if ( host.getMemoryAllocatedInMib() > clusterRAMallocatedAverage ) {
44                 hostsToBeBalanced.add(host);
45             } else {
46                 possibleTargets.add(host);
47             }
48         }
49
50         //Caso tenha hosts para serem balanceados
51         if (hostsToBeBalanced.size() > 0) {
52             hostsToBeBalanced =
   sortHostsByAllocatedRAMdecreasing(hostsToBeBalanced);       //Os primeiros estão mais
   sobrecarregados
53             possibleTargets =
   sortHostsByAllocatedRAMascending(possibleTargets);          //Os primeiros são os menos
   sobrecarregados
54             for (Host host : hostsToBeBalanced)
   {                                            //Hosts sobrecarregados
55                 Set<VirtualMachine> setVms = host.getVirtualMachines();
56                 List<VirtualMachine> vms = new
   ArrayList<VirtualMachine>(setVms);          //vmsAtivas
57                 vms =
   sortVMsByRAMallocatedAscending(vms);                                     //ordena vmsAtivas
   por alocação de RAM
58                 vmsMigration:
59                 for (VirtualMachine vm : vms) {
60
61                     searchTarget:
62                     for (Host targetHost : possibleTargets) {
63                         if (verifyPossibleOverload(targetHost, vm,
   clusterRAMallocatedAverage) != true) {  //Se o host não ficar acima da média, passar a VM
```

```
para ele
64                              updateHostResources(host, targetHost, vm);
65                              mapOfMigrations.put(vm, targetHost);
66                              possibleTargets =
sortHostsByAllocatedRAMascending(possibleTargets);          //Os primeiros são os menos
sobrecarregados
67                              break searchTarget;
68                          }
69                      }
70
71                      if ((long) host.getMemoryAllocatedInMib() < clusterRAMallocatedAverage)
{   //Não precisa mais migrar VMs pois já balanceou
72                          break vmsMigration;
73                      }
74                  }
75
76              }
77
78          }
79
80          return mapOfMigrations;
81      }
82
83      /**
84       * Updates host resources of two given hosts (source and target) moving a VM from the
source to the target.
85       * @param hostToBeOffloaded source host that will migrate its VM.
86       * @param candidateToReceiveVms target host that will receive the VM.
87       * @param vm VM to be migrated.
88       */
89      protected void updateHostResources(Host hostToBeOffloaded, Host candidateToReceiveVms,
VirtualMachine vm) {
90          long vmCpuConfigurationInMhz = vm.getVmServiceOffering().getCoreSpeed() *
vm.getVmServiceOffering().getNumberOfCores();
91          long vmMemoryConfigurationInBytes = vm.getVmServiceOffering().getMemoryInMegaByte()
* NUMBER_OF_BYTES_IN_ONE_MEGA_BYTE;
92
93          hostToBeOffloaded.getVirtualMachines().remove(vm);
94          hostToBeOffloaded.setCpuAllocatedInMhz(hostToBeOffloaded.getCpuAllocatedInMhz() -
vmCpuConfigurationInMhz);
95
hostToBeOffloaded.setMemoryAllocatedInBytes(hostToBeOffloaded.getMemoryAllocatedInBytes() -
vmMemoryConfigurationInBytes);
96
97          candidateToReceiveVms.addVirtualMachine(vm);
98      }
99
100     /**
101      * Calculate the allocated RAM average by hosts in the cluster.
102      * @param activeHosts list of hosts in the cluster.
103      */
104     void calculateClusterAllocatedRAMaverage(List<Host> activeHosts){
105         if (activeHosts.size() > 0) {
106             long allocatedClusterMemory = 0;
107             for (Host host : activeHosts) {
108                 allocatedClusterMemory += host.getMemoryAllocatedInMib();
109             }
110
111             clusterRAMallocatedAverage = allocatedClusterMemory/activeHosts.size();
112         }
113     }
114
```

```
115     /**
116      * Sort a list of hosts in decreasing order of allocated RAM for each host.
117      * @param hostsToBeSorted list of hosts to be sorted
118      * @return returns a list of hosts sorted in decreasing order
119      */
120     protected List<Host> sortHostsByAllocatedRAMdecreasing(List<Host> hostsToBeSorted){
121         List<Host> auxiliarList = hostsToBeSorted;
122         Collections.sort(auxiliarList, new HostComparator());
123         Collections.reverse(auxiliarList);
124         return auxiliarList;
125     }
126 
127     /**
128      * Sort a list of hosts in ascending order of allocated RAM for each host.
129      * @param hostsToBeSorted list of hosts to be sorted
130      * @return returns a list of hosts sorted in ascending order
131      */
132     protected List<Host> sortHostsByAllocatedRAMascending(List<Host> hostsToBeSorted){
133         List<Host> auxiliarList = hostsToBeSorted;
134         Collections.sort(auxiliarList, new HostComparator());
135         return auxiliarList;
136     }
137 
138     /**
139      * Sort a list of virtual machines in ascending order of allocated RAM for each virtual
machine.
140      * @param vmsToBeSorted list of virtual machines to be sorted
141      * @return returns a list of virtual machines sorted in ascending order
142      */
143     protected List<VirtualMachine> sortVMsByRAMallocatedAscending(List<VirtualMachine>
vmsToBeSorted){
144         List<VirtualMachine> auxiliarList = vmsToBeSorted;
145         Collections.sort(auxiliarList, new VirtualMachineComparator());
146         return auxiliarList;
147     }
148 
149     /**
150      * Verify the possibility of a given host to get RAM overloaded by migration of a given
virtual machine.
151      * @param targetHost candidate host to receive the virtual machine
152      * @param vm candidate virtual machine to be migrated to the given host.
153      * @param clusterAllocatedRAMaverage is used as a measure to know if the host allocated
RAM will be over the cluster allocated RAM average
154      * @return returns true if the host will get overloaded with the migration of the
virtual machine, or returns false otherwise
155      */
156     protected boolean verifyPossibleOverload(Host targetHost, VirtualMachine vm, long
clusterAllocatedRAMaverage){
157         long vmRAMallocated = vm.getVmServiceOffering().getMemoryInMegaByte();
158         long hostRAMallocated = targetHost.getMemoryAllocatedInMib();
159         long totalAllocatedRAM = vmRAMallocated + hostRAMallocated;
160 
161         if ((totalAllocatedRAM > clusterAllocatedRAMaverage) ||
targetHost.getTotalMemoryInMib() < vmRAMallocated + targetHost.getMemoryAllocatedInMib()) {
162             return true;
163         } else {
164             return false;
165         }
166     }
167 
168     /**
169      * Calculates the total allocated RAM of the cluster based its hosts list.
```

```
170      * @param hosts list of hosts from the cluster.
171      */
172     private void calculateTotalResources(List<Host> hosts){
173         long totalClusterRAMaux = 0;
174         for (Host host : hosts) {
175             totalClusterRAMaux += host.getTotalMemoryInMib();
176             numberOfVms += host.getVirtualMachines().size();
177         }
178         totalClusterRAM = (double)totalClusterRAMaux;
179         calculateClusterAllocatedRAMaverage(hosts);
180     }
181 }
182
183
184
```

APÊNDICE B – Artigo do SBC deste Trabalho de Conclusão de Curso

Desenvolvimento de uma Heurística de Balanceamento de Memória RAM em Ambientes Computacionais em Nuvem

Rodrigo Pedro Marques, Carlos B. Westphall, Gabriel B. Bräscher, Carla M. Westphall

[1] Universidade Federal de Santa Catarina (UFSC)
Departamento de Informática e Estatística (INE)
Laboratório de Redes e Gerência (LRG) – Florianópolis, SC – Brasil

{rodrigomarques, westphal, brascher, carla}@lrg.ufsc.br

Abstract. *Due to the large growing of cloud computing, the management complexity of these environments has grown, turning the manual management impractical. As a consequence, the automation of these environments takes an important part in making the management faster, improving the environment performance when implemented correctly. This work presents the development of a RAM memory balancing heuristic directed to cloud computing environment. This heuristic aims at making a better usage of this resource among hosts and clusters, improving the RAM memory management in cloud computing environments.*

Keywords: Cloud computing. Balancing heuristic. Resources balancing.

Resumo. *Devido ao grande crescimento do uso de ambientes computacionais em nuvem, a complexidade do gerenciamento destes ambientes tem aumentado significativamente, tornando o gerenciamento manual impraticável. Como consequência disto, a automatização deste processo se mostra importante pois auxilia a agilizar o gerenciamento, melhorando o desempenho quando feito de forma correta. Neste trabalho, é apresentado o desenvolvimento de uma heurística de balanceamento de memória RAM nestes ambientes, buscando aproveitar melhor este recurso entre hosts e clusters. Esta heurística facilitará o gerenciamento deste recurso em ambientes computacionais em nuvem.*

Palavras-chave: Computação em nuvem. Heurística de balanceamento. Balanceamento de recursos.

Introdução

Computação em nuvem é um modelo que possibilita o acesso conveniente e sob-demanda a recursos computacionais (por exemplo, redes, servidores, armazenamento, aplicações e serviços) que podem ser alocados e liberados com um mínimo de esforço de gerenciamento ou interação com o provedor de serviço [Mell and Grance 2011].

Devido ao crescimento da demanda por serviços ofertados por ambientes computacionais, a estrutura necessária para hospedar estes ambientes aumenta em tamanho e complexidade, impactando diretamente no gerenciamento destes ambientes [Weingärtner et al. 2015] e [Geronimo et al. 2013]. Devido à complexidade destas infraestruturas, a gerência de ambientes de computação em nuvem requer o auxílio de sistemas computacionais autônomos, como o modelo autônomo proposto por [Kephart and Chess 2003].

[Weingärtner et al. 2016] desenvolveram um *framework* distribuído autônomo para o *CloudStack* [Foundation 2016] que avalia a infraestrutura e, com base em determinadas heurísticas[1], toma decisões para migrar máquinas virtuais[2] e ligar/desligar servidores. Com a finalidade de demonstrar o potencial do framework desenvolvido, [Weingärtner et al. 2016] implementaram uma heurística que busca a redução do consumo energético em ambientes de computação em nuvem, por meio de técnicas de consolidação[3] de máquinas virtuais. Porém, a solução apresentada por [Weingärtner et al. 2016] possui carência de outras heurísticas, como uma heurística que balanceasse as máquinas virtuais entre os *hosts* com o melhor desempenho e aproveitamento possível.

Técnicas de gerência e otimização de ambientes em nuvem podem trazer benefícios como o controle da qualidade de serviços (QoS) e redução de gastos energéticos. Porém, diferentes técnicas podem trazer consequências, como por exemplo, o desligamento de servidores para economizar recursos energéticos pode vir a sobrecarregar servidores e prejudicar diretamente os clientes.

Apesar de muitos trabalhos desenvolverem técnicas de balanceamento de máquinas virtuais, poucos consideram o dinamismo, escala e heterogeneidade de ambientes em nuvem. Motivado por tal carência, este trabalho tem como objetivo estender o *framework* desenvolvido por [Weingärtner et al. 2016] apresentando uma proposta de heurística que tem como princípio distribuir máquinas virtuais que utilizam mais memória RAM em diferentes *hosts*, diminuindo a concorrência entre elas por este recurso. Isto evita que alguns servidores se encontrem subutilizados ou super utilizados, e também diminui a concorrência pelos mesmos recursos computacionais entre as máquinas virtuais. Assim, a degradação de serviço diminui, melhorando a qualidade de serviço oferecida pelo ambiente.

Motivação e Justificativa

Segundo [Forum 2013], a utilização de computação em nuvem tem crescido bastante. Em virtude deste amplo crescimento dos ambientes de computação em nuvem, a complexidade de gerenciamento aumenta significativamente como apresentado por [Weingärtner et al. 2015].

Atualmente, os ambientes computacionais em nuvem possuem poucas ou nenhuma heurística de balanceamento dos recursos computacionais utilizados pelas máquinas virtuais hospedadas. [Nathuji et al. 2010] mencionam que a virtualização nos servidores não garante o isolamento de desempenho entre as máquinas virtuais. Por exemplo, uma aplicação que esteja utilizando um núcleo de um processador com vários núcleos, pode sofrer redução de desempenho quando outra aplicação estiver rodando simultaneamente em um núcleo adjacente. Tal comportamento ocorre devido ao aumento na taxa de falta (do inglês, *miss rate*) no último nível de *cache*. Além disso, a competição por recursos computacionais também agrava a degradação de serviço, afetando a qualidade de serviço entregue ao cliente.

[1] Uma heurística é um método (ou estratégia) prático suficiente para atingir um objetivo. Elas podem ser utilizadas para acelerar o processo para atingir o objetivo.

[2] Máquinas virtuais são emulações de sistemas operacionais sobre um sistema hospedeiro, por exemplo, um sistema operacional ou *hardware*.

[3] A consolidação é uma técnica aplicada em ambientes virtualizados onde a sua implementação produz resultados significativos na eficiência do uso de recursos computacionais.

Motivado pela importância de evitar a degradação de serviço nestes ambientes e o aproveitamento inteligente dos recursos computacionais em ambientes de computação em nuvem, junto da necessidade de gerenciar estes ambientes de forma eficiente e autônoma, notou-se a possibilidade do desenvolvimento de uma heurística para balancear de forma eficiente a carga computacional utilizada por estes ambientes, com o mínimo possível de intervenção humana. Esta nova heurística pode ser uma alternativa a heurística proposta em [Weingärtner et al. 2016].

O resto deste artigo é organizado nas seguintes seções: trabalhos relacionados, proposta, resultados e análise, conclusões e referências.

Trabalhos Relacionados

[Bala and Chana 2016] desenvolveram um algoritmo que tem como base algoritmos de aprendizagem de máquina (como Redes Neurais, *Random Forest*, entre outros) para a previsão do consumo de recursos computacionais de cada máquina virtual em um ambiente de nuvem. Para que verificassem se um *host* estava sobrecarregado ou ocioso, foram inseridos limites máximos e mínimos. Assim, caso um *host* estiver acima do limite de utilização de recursos, ele é classificado como "sobrecarregado", inicializando assim a migração de máquinas virtuais para outros *hosts* até ele ficar dentro dos limites estabelecidos. Caso o *host* estiver abaixo do nível mínimo estabelecido, ele é classificado como "ocioso". Assim, se inicializa a migração das máquinas virtuais para outros *hosts* e quando não houver mais máquinas virtuais nele, ele é desligado.

Em síntese, o trabalho de [Bala and Chana 2016] tenta prever o consumo de recursos computacionais das máquinas virtuais para assim migrá-las ou não para outros *hosts* que sejam capazes de suportar a carga que elas irão gerar num dado momento. Apesar de o trabalho de [Bala and Chana 2016] levar em consideração diferentes parâmetros como CPU, RAM, rede e Leitura e Escrita em Disco, não procura diminuir a concorrência pelos mesmos recursos entre as máquinas virtuais, como é proposto por este trabalho.

[Kulkarni and Annappa 2015] apresentam em seu trabalho um algoritmo de balanceamento de máquinas virtuais alternativo ao algoritmo nativo de balanceamento da ferramenta *CloudAnalyst* [Wickremasinghe 2009]. O algoritmo de balanceamento nativo aloca as requisições para as máquinas virtuais menos sobrecarregadas em um dado momento para manter uma alocação uniforme para todas as máquinas virtuais. [Kulkarni and Annappa 2015] aponta que o algoritmo nativo pode ocasionar uma super alocação de máquinas virtuais quando há muitas requisições ao controlador do centro de dados.

Para resolver este problema, [Kulkarni and Annappa 2015] propuseram um algoritmo alternativo ao algoritmo nativo do *CloudAnalyst*. Neste algoritmo, eles adicionaram uma tabela de reservas interna ao balanceador de máquinas virtuais para manter as recomendações de reserva de cada máquina virtual sugeridas pelo controlador do centro de dados (por exemplo, quantidade de memória RAM e CPU necessários para executar a máquina virtual). Assim, o balanceador irá levar em consideração a tabela de reservas interna e a tabela de alocação estática de uma máquina virtual em particular. O algoritmo garante a alocação uniforme até nas horas de pico.

[Janpan et al. 2014] apresentam um *framework* de consolidação para o *CloudStack* através de *live migration*. O *framework* proposto consiste de quatro componentes:

um operador do *CloudStack*, um operador de monitoramento, um operador de mapeamento e um controlador de força. O operador do *CloudStack* é responsável por interagir com a API do *CloudStack*; o operador de monitoramento é responsável por coletar dados de monitoramento de sistemas de monitoramento; o controlador de força é responsável por ligar e desligar um *host* para economizar energia; e o operador de mapeamento é responsável por mapear máquinas virtuais para outros *hosts*.

O método de consolidação proposto por [Janpan et al. 2014] busca balancear a utilização de CPU no ambiente de computação em nuvem. A essência do algoritmo se dá quando um *host* é liberado para receber migrações de máquinas virtuais. Inicialmente, buscam-se todos os *hosts* e máquinas virtuais do ambiente. Em seguida, é realizada uma busca sobre esses *hosts* para achar aquele com o menor índice de utilização de CPU. Este índice é utilizado para determinar quando o processo de migração para o *host* alvo deve ser finalizado. Então, inicia-se o processo de migração de máquinas virtuais buscando aqueles *hosts* com os maiores índices de utilização de CPU. A escolha da máquina virtual dentro destes *hosts* é feita aleatoriamente. Estes dois últimos passos são repetidos até que o índice de utilização de CPU no *host* alvo seja maior que o limite determinado. Além deste método, [Janpan et al. 2014] apresentaram um método de desconsolidação responsável por migrar as máquinas virtuais de um *host* que será desligado para outros *hosts*.

[Beloglazov and Buyya 2012] apresentam uma proposta de heurísticas adaptativas para consolidação dinâmica de máquinas virtuais. As heurísticas se baseiam de acordo com a análise dos históricos de uso dos recursos computacionais de cada máquina virtual. Para realizar a consolidação do ambiente, quatro fatores são levados em consideração: determinar quando um *host* é considerado sobrecarregado, necessitando um balanceamento de carga; determinar quando um *host* é considerado ocioso, podendo migrar suas máquinas virtuais e ser desligado para economizar energia; selecionar quais máquinas virtuais devem ser migradas de um *host* sobrecarregado; e encontrar um *host* alvo para receber as máquinas virtuais de um *host* sobrecarregado.

O algoritmo desenvolvido em [Beloglazov and Buyya 2012], de forma resumida, busca inicialmente a lista de todos os *hosts* do ambiente e verifica quais estão sobrecarregados. Caso um *host* esteja sobrecarregado, o algoritmo executa a seleção das máquinas virtuais a serem migradas de acordo com a política adotada. Após construir a lista de máquinas virtuais que necessitam ser migradas, buscam-se os *hosts* que estão ociosos para receber as máquinas virtuais. O algoritmo então retorna um *map* contendo as informações dos novos locais das máquinas virtuais que foram migradas. Para a verificar se um *host* está sobrecarregado ou não, é considerado o uso de CPU dos *hosts*, porém, o limite utilizado para considerar se um *host* está sobrecarregado ou não é dinâmico. A ideia principal deste limite dinâmico é ajustar o índice limite dependendo do tamanho do desvio absoluto da utilização da CPU. Quanto maior este desvio, menor o índice limite de utilização, quanto maior o desvio, maior a probabilidade da utilização da CPU atingir 100%.

Proposta

A computação em nuvem tem ganhado cada vez mais destaque e adesão nos últimos anos devido, principalmente, ao seu baixo custo de alocação e gerenciamento. O custo de um

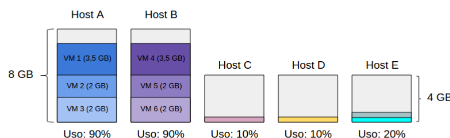

Figure 1. Ilustração do problema a ser corrigido.

serviço de nuvem comparado à compra, gerenciamento e manutenção de máquinas físicas, seria muito mais alto. Sendo assim, as empresas acabam migrando seus serviços para a nuvem. Apesar dos serviços oferecidos hoje através da computação em nuvem terem um alto desempenho, a concorrência entre máquinas virtuais por recursos computacionais pode ocasionar degradação do serviço oferecido.

Como apresentado por [Weingärtner et al. 2015] e [Geronimo et al. 2013], o crescimento dos ambientes computacionais em nuvem e serviços, acaba gerando a necessidade de aumentar a estrutura necessária para hospedar esses serviços, aumentando a complexidade de gerenciamento destes ambientes.

Analisando e estudando os trabalhos relacionados, constatou-se a necessidade de novas formas de consolidação dos ambientes computacionais em nuvem para melhorar o desempenho e eficiência energética dos mesmos. Como apresentado anteriormente nos trabalhos relacionados, existem diferentes propostas para a consolidação dos ambientes computacionais em nuvem, porém, nenhum deles leva em consideração a concorrência por recursos computacionais.

A proposta deste trabalho é minimizar a concorrência por memória RAM entre as máquinas virtuais dentro de um *cluster*. Isto será realizado através de migrações de máquinas virtuais entre os *hosts* buscando deixar o consumo de memória RAM o mais balanceado possível. O escopo deste trabalho se limita apenas à memória RAM devido ao fato de que ao acrescentar outros fatores, como utilização de CPU ou leitura e escrita em disco, por exemplo, aumentaria significativamente a complexidade deste trabalho, indo além de um trabalho de conclusão de curso.

Ilustração do Problema

Em um ambiente computacional em nuvem que carece de técnicas inteligentes de balanceamento de carga computacional, existe a possibilidade de ocorrer sobrecarga dos *hosts*, influenciando diretamente o desempenho dos serviços entregues através deste ambiente. A figura 1 apresenta um ambiente heterogêneo onde a quantidade de recursos disponibilizados pelos *hosts* são diferentes. Como é possível analisar, os *hosts* A e B possuem $8GB$ de memória RAM, enquanto os *hosts* C, D e E possuem $4GB$ de memória RAM. Ainda nesta figura, é possível observar um eventual problema que pode acontecer nestes ambientes. Os *hosts* A e B estão com 90% de sua capacidade de memória RAM em uso, enquanto os servidores C, D e E estão com sua taxa de utilização de memória RAM entre 10% e 20% em uso.

A figura 2 representa a proposta deste trabalho. Considerando um ambiente com-

putacional em nuvem heterogêneo como apresentado anteriormente, a heurística de balanceamento irá buscar equilibrar a utilização de memória RAM entre os *hosts* do ambiente. Isto será realizado através de migrações de máquinas virtuais dos *hosts* sobrecarregados para os *hosts* com o menor índice de utilização de memória RAM da sua capacidade total. Como é possível analisar na figura 2, para equilibrar a utilização de memória RAM apresentada na figura 1, as máquinas virtuais $VM3$ e $VM5$ que estavam hospedadas nos *hosts* A e B, respectivamente, foram migradas para os *hosts* C e D. Então, a utilização de memória RAM nos *hosts* A e B passaram de 90% para 62%, enquanto que nos *hosts* C e D, passaram de 10% para 58%. Assim, houve uma descarga de 28% dos *hosts* A e B e um aumento de utilização de memória RAM de 48% nos *hosts* C e D, equilibrando a utilização de memória RAM no ambiente e diminuindo a concorrência por este recurso.

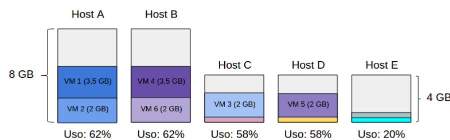

Figure 2. Ilustracao da proposta deste trabalho.

Arquitetura do *Framework* Autonomiccs

O *framework Autonomiccs* foi desenvolvido para atuar no nível de *cluster*, onde múltiplos agentes podem gerenciar simultaneamente ambientes de computação em nuvem grandes e dinâmicos. Cada algoritmo não opera sobre a infraestrutura como um todo, mas sim sobre fragmentos da mesma, permitindo uma aproximação distribuída [Weingärtner et al. 2016]. Este trabalho se encontra na parte "Allocation Manager", como apresenta a figura 3, onde o agente irá executar a heurística desenvolvida neste trabalho periodicamente.

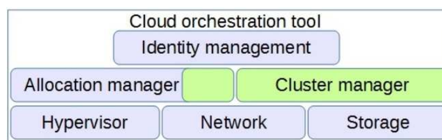

Figure 3. Arquitetura da orquestração em nuvem com o *framwork Autonomiccs* [Weingärtner et al. 2016].

Como apresentado na figura 3, o *framework* foi desenvolvido para ser parte de ferramentas de orquestração. Assim, foi adicionado um novo componente chamado *cluster manager* que é responsável por executar as migrações das máquinas virtuais e desligamento de servidores ociosos. Além disso, foi necessário estender o comportamento do *allocation manager* que é responsável pelas alocações das máquinas virtuais quando são criadas, inicializadas ou migradas [Weingärtner et al. 2016].

A carência de alternativas de ferramentas de gerência autônoma de código aberto junto da facilidade e eficiência de se implementar novas heurísticas de balanceamento ao *framework* para trabalhar sobre um ambiente de nuvem, motivaram a escolha de utilizá-lo neste trabalho, acelerando a implementação e análise da nova heurística desenvolvida.

Algoritmo

Nesta seção será apresentado o algoritmo da heurística e seus principais métodos.

O algoritmo 1 irá executar periodicamente em cada *cluster* isoladamente e serão necessárias algumas informações referente aos níveis de estrutura do ambiente no qual ele irá atuar. Seguindo do nível mais alto para o mais baixo, as informações necessárias serão:

- Cluster: será necessário saber a alocação média de memória RAM[4] no momento em que o algoritmo rodar. Esta média será utilizada de referência para balancear a alocação de memória RAM dos *hosts* ativos que atuam dentro do *cluster*;
- Servidor: será necessário saber a alocação de memória RAM do *host* para verificar se existe a necessite de balanceá-lo ou não; e
- Máquinas Virtuais: será necessário saber a quantidade de memória RAM alocada para cada máquina virtual num dado *host* para verificar a possibilidade de migração de uma dada máquina virtual para outro *host*.

Para uma melhor apresentação do algoritmo 1 (balanceamentoDeMemoriaRAM), ele foi subdividido em partes menores para facilitar o entendimento. Inicialmente, na *linha 1* é verificada a existência de *hosts* ativos no *cluster*. Caso não houverem *hosts* ativos, não há necessidade de balancear o *cluster*. Então, é finalizado o algoritmo *balanceamentoDeMemoriaRAM*, como mostra a *linha 17*. Caso contrário, isto é, caso existam *hosts* ativos, busca-se a média de alocação de memória RAM do *cluster*. Esta média será utilizada como parâmetro para balancear a alocação de memória RAM dos *hosts* contidos neste *cluster*.

A partir da *linha 3* à *linha 9*, é realizada uma busca entre os servidores ativos verificando quais precisam ser balanceados e quais estão abaixo da alocação média de RAM do *cluster*, podendo receber máquinas virtuais de outros *hosts*.

Na *linha 4*, é realizada uma verificação para garantir que dentre os *hosts* ativos existe a necessidade de balanceamento de memória RAM entre eles, isto é, se existe alguma sobrecarga na alocação de memória RAM em algum *host* em relação à média de alocação de memória RAM do *cluster*. Caso negativo, não há necessidade de balancear o *cluster*, finalizando assim o algoritmo como mostra a *linha 14*. Porém, caso for constatada a sobrecarga de ao menos um *host*, então o algoritmo 2 (balanceiaHostsComMigracoes) é executado.

[4]É importante ressaltar que neste ponto foi escolhido trabalhar com a alocação de memória RAM ao invés da utilização atual deste recurso. Trabalhar com o uso dinâmico de memória RAM é mais arriscado pois ele pode oscilar muito, o que torna difícil a previsão e controle do mesmo. Caso ele extrapole demais, ele pode causar uma série de erros no *host*, podendo comprometê-lo. Por outro lado, trabalhar com o que foi alocado para a máquina virtual, garantirá que ela nunca irá utilizar mais memória RAM do que lhe foi permitido, evitando assim, possíveis erros e problemas.

Algorithm 1 balanceamentoDeMemoriaRAM

Entrada: $cluster, hostsAtivos$

```
1: if hostsAtivos.size()>0 then
2:   media ← obterAlocaoMediaDeRAM(cluster)
3:   for host : hostsAtivos do
4:     if host.mediaDeAlocacaoDeRAM>media then
5:       hostsComMigracoes.add(host)
6:     else
7:       possiveisHostsAlvo.add(host)
8:     end if
9:   end for
10:  if hostsComMigracoes.size()>0 then
11:    balanceiaHostsComMigracoes(hostComMigracoes,
12:    possiveisHostsAlvo, media)
13:  else
14:    terminaAlgoritmo
15:  end if
16: else
17:   terminaAlgoritmo
18: end if
```

O algoritmo 2 recebe como entrada duas listas de *hosts*: *hostsComMigracoes* e *possiveisHostsAlvo*; e também recebe a média de alocação de memória RAM. Inicialmente a lista *hostsComMigracoes* é ordenada por alocação de memória RAM em ordem decrescente, assim, os primeiros *hosts* da lista serão os mais sobrecarregados em questão de alocação de memória RAM. Em seguida, a lista *possiveisHostsAlvo* é organizada em ordem crescente de alocação de memória RAM, assim, os primeiros *hosts* são os com menores índices de alocação de memória RAM.

Em seguida, para cada *host* que foi adicionado à lista *hostsComMigracoes*, buscam-se as máquinas virtuais ativas do servidor e as ordena por alocação de memória RAM. Assim que todas as máquinas virtuais ativas foram identificadas, o algoritmo 3 é chamado para cada delas. Após percorrer toda a lista de *vmsAtivas*, o algoritmo 2 é finalizado.

O algoritmo 3 (*migraVMs*) recebe como entrada a lista de *possiveisHostsAlvo*, a máquina virtual *vm* a ser migrada e a média de utilização de RAM do *cluster*. O seu principal papel é buscar um *host* ativo que está contido na lista *possiveisHostsAlvo* que seja capaz de suportar a máquina virtual *vm* e que quando migrá-la para este *host* alvo, o seu índice de utilização de RAM deve ser menor que a média de utilização de RAM do *cluster*. A linha 9 se mostrou importante pois após o *host* alvo receber a máquina virtual, ele pode estar mais carregado que os outros *hosts* alvos da lista. Assim, é feita uma nova ordenação desta lista. O algoritmo 3 é finalizado quando o *host* esta balanceado, ou todas as máquinas virtuais ativas foram balanceadas, como mostra a *linha 16*.

Algorithm 2 balanceiaHostsComMigracoes

Entrada: $hostsComMigracoes, possiveisHostsAlvo, media$

```
1: hostsComMigracoes ← ordenaPorAlocacaoDeRAMdecrescente()
2: possiveisHostsAlvo ← ordenaPorAlocacaoDeRAMcrescente()
3: for host : hostsComMigracoes do
4:    vmsAtivas ← buscaVMsAtivas(host)
5:    if vmsAtivas.size()>0 then
6:       vmsAtivas ← ordenaPorAlocacaoDeRAM(vmsAtivas)
7:       for vm : vmsAtivas do
8:          migraVMs(vm, possiveisHostsAlvo, media)
9:       end for
10:   else
11:      quebra laço
12:   end if
13: end for
```

Algorithm 3 migraVMs

Entrada: $vm, possiveisHostsAlvo, media$

```
1: if possiveisHostsAlvo.size()>0 then
2:    hostAlvo ← possiveisHostsAlvo[head]
3:    alocacaoDeRAMHostAlvo ← hostAlvo.buscaAlocacaoDeRAM()
4:    maximoRAMHostAlvo ← hostAlvo.getRAM()
5:    vmRAM ← vm.RAMAlocada()
6:    tR ← vmRAM + alocacaoDeRAMHostAlvo
7:    if tR<media && tR<maximoRAMHostAlvo then
8:       migra vm para hostAlvo
9:       possiveisHostsAlvo ← ordenaPorAlocacaoDeRAMcrescente()
10:   else
11:      pega proximo elemento de possiveis host
12:   end if
13: end if
14: alocacao ← host.alocacaoDeRAM()
15: numeroVMs ← vmsAtivas.size()
16: if alocacao<media || numeroVMs ≤ 0 then
17:    quebra laço
18: end if
```

Resultados e Análise

Para realizar estudos de comparação e análises da heurística desenvolvida neste trabalho, foi utilizada uma outra heurística preexistente chamada *ClusterAdministrationAlgorithmEmptyImpl*. Esta heurística não realiza técnicas de gerenciamento de ambientes computacionais em nuvem, ela apenas cria e destrói máquinas virtuais de acordo com as exigências do ambiente.

A heurística desenvolvida neste trabalho, realiza migrações de máquinas virtuais entre os *hosts* ativos de um determinado *cluster*. O parâmetro de referência utilizado para

migrar as máquinas virtuais, é a alocação de memória RAM de cada *host* comparado com a média de memória RAM alocada pelos *hosts* dentro de um *cluster*. Caso a média de alocação de memória RAM de um *host* for maior que a média de alocação de memória RAM do *cluster*, então é realizada uma busca por um *host* alvo que esteja apto a receber alguma máquina virtual, diminuindo a sobrecarga do *host* sobrecarregado.

Assim, tem-se alguns fatores, seja $M(c)$ a média de alocação de memória RAM do *cluster* c, e R a quantidade de memória RAM alocada pelos *hosts* ativos no *cluster* c e n o número de *hosts* ativos no *cluster* c, então:

$$M(c) = \frac{R}{n}$$

Seja vr a quantidade de alocação de memória RAM requisitada pela máquina virtual, R a quantidade de memória RAM alocada no *host* h, tR o total de memória RAM alocada, e tH o total de memória RAM[5] do *host*, então:

$$tR = vr + R(h)$$

Logo, para saber se o *host* alvo ficará sobrecarregado em relação a $M(c)$, levando apenas a memória RAM em consideração, são feitas duas verificações: caso $tr > M(c)$ ou $tr > tH$, então o *host* ficará sobrecarregado.

Então, para realizar os testes foi realizada uma simulação sobre um arquivo de entrada utilizando a heurística *ClusterAdministrationAlgorithmEmptyImpl*, e em seguida, foi realizada uma nova simulação sobre este mesmo arquivo de entrada utilizando a heurística *ClusterVMsBalancingByAllocatedRAM*. No final de cada simulação é gerado um *log* de saída com diversas informações, onde é possível filtrar o tempo de execução e média de memória RAM alocada pelo tempo. A figura 4 apresenta os resultados de ambas as heurísticas.

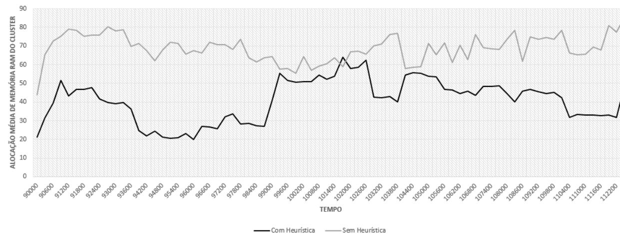

Figure 4. Resultado de alocação de RAM x Tempo das heurísticas

Na figura 4, no eixo horizontal tem-se o *Tempo* e no eixo vertical tem-se a *Média de Alocação de Memória do Cluster*. Como é possível observar, a simulação mostrou

[5]O total de memória RAM de um *host* ou *cluster* é a quantidade máxima de memória RAM que um *host* ou *cluster* pode ter, por exemplo, $32GB$. A quantidade de memória RAM alocada é uma reserva de memória RAM feita por um *host* ou máquina virtual do total de memória RAM. Ou seja, caso uma máquina virtual for alocar memória RAM num *host* que tem o total de memória RAM de $32GB$, então, esta máquina virtual pode alocar até $32GB$ de memória RAM.

que quando executa sem uma heurística de balanceamento de memória RAM, a alocação média de memória RAM foi de 68.73%, onde teve um pico máximo de aproximadamente 87%. Em um ambiente onde a média de alocação de memória RAM chega à 87% compromete a qualidade de serviço entregue ao usuário. Como é possível observar, a heurística desenvolvida neste trabalho mostrou um ótimo resultado, tendo uma alocação média de memória RAM de 42.74%, contra 68.73%, apresentando uma otimização de 37.81%. Com isso a concorrência por memória RAM entre máquinas virtuais diminui, deixando os *hosts* menos sobrecarregado neste requisito e melhorando a qualidade de serviço entregue ao usuário.

Conclusões

Este trabalho resultou no desenvolvimento de uma heurística de balanceamento de memória RAM em um ambiente computacional em nuvem capaz de ser implementada em qualquer ambiente que contenha a ferramenta de orquestração *CloudStack*, visto que a plataforma *Autonomiccs* é direcionada a ela.

A heurística desenvolvida, apesar de se apresentar simples, se mostrou bastante eficiente, apresentando ótimos resultados como foi mostrado.

Este trabalho contém as seguintes contribuições:

- Diminuiu a necessidade de intervenção manual para o balanceamento de memória RAM em um ambiente de computação em nuvem;
- Agilizou o processo de balanceamento de memória RAM em um ambiente de computação em nuvem; e
- Melhoria e otimização na alocação de memória RAM em ambientes computacionais em nuvem.

As principais contribuições, em relação às ideias propostas pelos trabalhos relacionados apresentados, está na busca de otimização de apenas um recurso computacional, que no caso deste trabalho foi a memória RAM. A heurística busca organizar os *hosts* mais sobrecarregados em primeiro lugar e em outra lista de *hosts* alvos busca-se colocar os menos sobrecarregados primeiro. Assim, empreende-se descarregar os *hosts* mais sobrecarregados primeiro transferindo sua sobrecarga para o menos sobrecarregados. Outra característica importante de ressaltar, é o fato de que uma melhor distribuição de máquinas virtuais que utilizam muita memória RAM entre *hosts* diferentes faz com que a concorrência por este recurso tenda a ser menor.

Como trabalhos futuros têm-se:

- Adicionar um histórico de consumo de recursos computacionais de cada máquina virtual. Assim, seria possível criar heurísticas mais inteligentes que se baseassem em um consumo mais dinâmico e previsível;
- Adicionar uma função que permitisse colocar um valor limite de consumo de um dado recurso computacional, tal que quando uma máquina virtual atingisse este limite, um aviso seria lançado no sistema para que ele possa realizar ações para amenizar o problema. Por exemplo, colocar um limite de consumo de 60% de memória RAM. Caso uma máquina virtual atinja este limite, o sistema pode ativar uma heurística de balanceamento para resolver o problema;
- Adicionar um aviso de sucesso ou falha de migração de máquinas virtuais ao *CloudStack*;

- Alterar o foco da heurística para balanceamento de CPU e verificar os resultados; e
- Produzir um artigo científico apresentado este trabalho à comunidade.

Referências

References

Bala, A. and Chana, I. (2016). Prediction-based proactive load balancing approach through vm migration. *Engineering with Computers*, pages 1–12.

Beloglazov, A. and Buyya, R. (2012). Optimal online deterministic algorithms and adaptive heuristics for energy and performance efficient dynamic consolidation of virtual machines in cloud data centers. *Concurrency and Computation: Practice and Experience*, 24(13):1397–1420.

Forum, T. C. I. (2013). Uk cloud adoption and trends for 2013.

Foundation, T. A. S. (2016). Apache cloudstack.

Geronimo, G. A., Werner, J., Westphall, C. B., Westphall, C. M., and Defenti, L. (2013). Provisioning and resource allocation for green clouds. In *12th International Conference on Networks (ICN)*.

Janpan, T., Visoottiviseth, V., and Takano, R. (2014). A virtual machine consolidation framework for cloudstack platforms. In *The International Conference on Information Networking 2014 (ICOIN2014)*, pages 28–33. IEEE.

Kephart, J. O. and Chess, D. M. (2003). The vision of autonomic computing. *Computer*, 36(1):41–50.

Kulkarni, A. K. and Annappa, B. (2015). Load balancing strategy for optimal peak hour performance in cloud datacenters. In *Signal Processing, Informatics, Communication and Energy Systems (SPICES), 2015 IEEE International Conference on*, pages 1–5. IEEE.

Mell, P. and Grance, T. (2011). The nist definition of cloud computing.

Nathuji, R., Kansal, A., and Ghaffarkhah, A. (2010). Q-clouds: managing performance interference effects for qos-aware clouds. In *Proceedings of the 5th European conference on Computer systems*, pages 237–250. ACM.

Weingärtner, R., Bräscher, G. B., and Westphall, C. B. (2015). Cloud resource management: A survey on forecasting and profiling models. *Journal of Network and Computer Applications*, 47:99–106.

Weingärtner, R., Bräscher, G. B., and Westphall, C. B. (2016). A distributed autonomic management framework for cloud computing orchestration. In *2016 IEEE World Congress on Services (SERVICES)*, pages 9–17.

Wickremasinghe, B. (2009). Cloudanalyst: A cloudsim-based tool for modelling and analysis of large scale cloud computing environments.