UNIVERSIDADE ESTADUAL PAULISTA
"JÚLIO DE MESQUITA FILHO"
Câmpus de São José do Rio Preto

João Antonio Magri Rodrigues

Otimização de alocação de máquinas virtuais em *datacenter* heterogêneo de sistema de computação em nuvem

São José do Rio Preto

2019

João Antonio Magri Rodrigues

Otimização de alocação de máquinas virtuais em *datacenter* heterogêneo de sistema de computação em nuvem

Dissertação apresentada como parte dos requisitos para obtenção do título de Mestre em Ciência da Computação, junto ao Programa de Pós-Graduação em Ciência da Computação, Área de Concentração - Arquitetura de Computadores e Sistemas Distribuídos, do Instituto de Biociências, Letras e Ciências Exatas da Universidade Estadual Paulista "Júlio de Mesquita Filho", Campus de São José do Rio Preto.
Financiadora: CAPES

Orientador: Prof. Dr. Aleardo Manacero Junior

São José do Rio Preto

2019

Rodrigues, João Antonio Magri
R696o Otimização de alocação de máquinas virtuais em datacenter heterogêneo de sistema de computação em nuvem / João Antonio Magri Rodrigues. -- São José do Rio Preto, 2019
89 p. : il., tabs.

Dissertação (mestrado) - Universidade Estadual Paulista (Unesp), Instituto de Biociências Letras e Ciências Exatas, São José do Rio Preto
Orientador: Aleardo Manacero

1. Ciência da Computação. 2. Sistemas distribuídos.
3. Alocação de recursos. I. Título.

Sistema de geração automática de fichas catalográficas da Unesp. Biblioteca do Instituto de Biociências Letras e Ciências Exatas, São José do Rio Preto. Dados fornecidos pelo autor(a).

João Antonio Magri Rodrigues

Otimização de alocação de máquinas virtuais em *datacenter* heterogêneo de sistema de computação em nuvem

Dissertação apresentada como parte dos requisitos para obtenção do título de Mestre em Ciência da Computação, junto ao Programa de Pós-Graduação em Ciência da Computação, Área de Concentração - Arquitetura de Computadores e Sistemas Distribuídos, do Instituto de Biociências, Letras e Ciências Exatas da Universidade Estadual Paulista "Júlio de Mesquita Filho", Campus de São José do Rio Preto.
Financiadora: CAPES

Banca examinadora:

Prof. Dr. Aleardo Manacero Junior
UNESP – São José do Rio Preto
Orientador

Prof. Dr. Rafael Pasquini
UFU – Uberlândia

Prof. Dr. Rodolfo Ipolito Meneguetti
IFSP – Catanduva

São José do Rio Preto
4 de Janeiro de 2019

Às minhas avós Angelica Bellini e Aparecida Magri.

Agradecimentos

É uma grande satisfação chegar até a etapa final deste trabalho e poder concluir o curso de mestrado acadêmico por uma das mais conceituadas universidade do país, a UNESP, a qual sempre terei orgulho em dizer que fiz parte, desde a graduação até este momento. Durante este período, tive momentos de alegria, aprendizado e superação, os quais compartilhei com pessoas importantes para mim que gostaria que fossem lembradas neste trabalho. Assim, agradeço:

Aos meus pais, Edgard e Zezé, à minha irmã Ana Carolina e à minha namorada Alana pelo apoio e união familiar.

Aos meus tios e tias, Antonio Bellini, Iracema, Antonio Magri, Claudete, José Carlos, às meus avôs e avós Antonio, João, Aparecida e Angelica, que já não estão mais entre nós, aos meus primos e primas Lucianna, Daniela, Camila, Wesley, Rogério, Victor Hugo, Heitor, Alice, Priscila, Ana Luiza e Pedro Henrique pelos bons momentos que compartilhamos.

Aos amigos, Guilherme Rodrigues, Guilherme Pasqualato, Guilherme Avino, Artur, Victor Hugo, João Marcos, Nilson Henrique, Carlos Nimer e Alexandre Bugati pela amizade e companheirismo.

Aos companheiros de treino e amigos do Jiu-Jitsu, Kleber, Wilson, Cezar, Alceu, Douglas, Francisco, Johnny, José Roberto, Diane e especialmente aos professores Bruno Henrique e Thyago Fernandes, pela amizade e ensinamentos.

Aos amigos do Grupo de Sistemas Paralelos e Distribuídos da UNESP, Fernanda, Diogo, Lucas, Luis Vinicius, Renan Miguel, Gabriel, Victor Hugo, Hethini e Vinicius e especialmente ao meu orientador Prof. Dr. Aleardo Manacero e a Profª. Drª. Renata Spolon Lobato, pelas orientações e contribuições para este trabalho.

A Me. Fabíola M. C. Oliveira e ao Prof. Dr. Edson Borín, da UNICAMP, pela importante contribuição que suas orientações trouxeram para este trabalho.

O presente trabalho foi realizado com apoio da Coordenação de Aperfeiçoamento de Pessoal de Nível Superior - Brasil (CAPES) - Código de Financiamento 001.

"Ser tão forte que nada possa perturbar a paz da tua mente."
- Helio Gracie

Resumo

Computação em nuvem pode ser definida como uma tecnologia de oferta de serviços de computação por meio da *Internet*, utilizando virtualização de máquinas. A virtualização é um procedimento em que se estabelece um ambiente virtual para execução de tarefas consumindo parte dos recursos de uma máquina real. Desse modo, o desempenho de um sistema de computação em nuvem depende da eficiência da alocação de máquinas virtuais em máquinas reais, atendendo restrições e metas diversas. Neste trabalho se propõe uma nova abordagem para alocação de máquinas virtuais que tem como objetivo otimizar o número de máquinas físicas ativas e o tráfego na rede do sistema, tratando situações de conflito e balanço entre estes dois objetivos em sistemas heterogêneos. A solução proposta é baseada em uma modificação do algoritmo para particionamento de grafos de *Kernighan-Lin* para tratar os custos de comunicação, além de heurísticas para a minimização do número de máquinas físicas. O texto apresenta um levantamento bibliográfico a respeito de computação em nuvem, o estado da arte relacionado ao problema de alocação de máquinas virtuais, a implementação do algoritmo e sua avaliação. O algoritmo proposto é avaliado contra uma heurística convencional e um algoritmo do estado da arte em diversos cenários. Os resultados obtidos mostram que, apesar da dificuldade de conciliação entre estes dois objetivos em se tratando de sistemas heterogêneos, as soluções obtidos pela abordagem desenvolvida são de boa qualidade.

Palavras-chaves: Computação em nuvem, máquinas virtuais, alocação.

Abstract

Cloud computing is a term referring to a computing service technology offered through the Internet using machine virtualization, which is a process where a virtual environment is deployed to run an application, consuming part of the real machine resources. Therefore, the performance of a cloud computing system depends on the efficiency of the virtual machines placement in real machines, given certain goals and constraints. This work aims to present a new approach for virtual machine placement that optimizes the number of active physical machines and network traffic in its datacenter, as well as evaluate the conflict between these goals in heterogeneous systems. The proposed approach is based in a modification of the Kernighan-Lin algorithm for graph partitioning to deal with communication costs, and heuristics to minimize the number of physical machines. The text presents a conceptual review about cloud computing, the state of art of the virtual machine placement problem, the algorithm implementation and its evaluation. The proposed algorithm is evaluated against a conventional heuristic and a state of art algorithm in various scenarios. The results reveal the hardness to balance the two defined goals in heterogeneous systems as well as the quality of the solution achieved by the proposed approach.

Key-words: Cloud computing, virtual machines, placement.

Lista de ilustrações

Lista de tabelas

Lista de abreviaturas e siglas

VM	Virtual Machine
IaaS	Infrastructure as a Service
PaaS	Platform as a Service
SaaS	Software as a Service
IoT	Internet of things
RAM	Random Access Memory
GB	Giga Byte
Mbps	Mega bit per second
MIPS	Million Instructions Per Second
FF	First-Fit
FFD	First-Fit Decreasing
BF	Best-Fit
BFD	Best-Fit Decreasing
WF	Worst-Fit
WFD	Worst-Fit Decreasing
MinCut	Minimum Cut

Sumário

1 Introdução

Computação em nuvem (do inglês *Cloud Computing*) é o nome que se dá a um tipo de sistema distribuído composto por uma coleção de máquinas virtuais (VMs) interconectadas e provisionadas dinamicamente, as quais tem sua disponibilidade baseada em um contrato de nível de serviço, o SLA (BUYYA; BROBERG; GOSCINSKI, 2011).

Esses sistemas surgiram pois uma grande parcela dos recursos computacionais de empresas, principalmente as do ramo de Tecnologia da Informação (TI), fica ociosa boa parte do tempo. Assim, com todo um poderio computacional aliado ao *know-how* que essas empresas possuíam em gerenciamento de TI, foi possível estabelecer um mercado de computação que inclui serviços como aluguel de infraestruturas computacionais, armazenamento e *backup* de arquivos pessoais, plataformas para desenvolvimento e implantação de aplicações. Todos esses serviços contratados e gerenciados por meio da *Internet* (BUYYA; BROBERG; GOSCINSKI, 2011).

Os serviços oferecidos por um sistema de computação em nuvem atendem um grande nicho de mercado, desde usuários comuns até grandes empresas, como por exemplo a *Netflix*, que tem seus servidores hospedados no complexo computacional da *Amazon* (AMAZON, 2018).

Para atender a diferentes demandas de clientes em diferentes momentos, o sistema deve prover elasticidade de recursos, aumentando-os ou diminuindo-os quando necessário. Para tanto é preciso que haja uma política eficiente de alocação de máquinas virtuais. É nesse ponto que surge a necessidade de bons escalonadores para melhorar a distribuição de máquinas virtuais entre os recursos físicos, com diversas finalidades, como por exemplo, balanceamento de carga, redução do número de máquinas ativas, otimização de rede, etc.

1.1 Motivação

Para a computação em nuvem, vista como um negócio, o provedor do serviço deve, no mínimo, cumprir o Acordo de Nível de Serviço (SLA) estabe-

lecido com seus clientes. Assim, um bom algoritmo de alocação de máquinas virtuais, além de garantir o cumprimento dos requisitos acordados entre cliente e provedor, deve atenuar gastos. Uma das estratégias empregadas para esta finalidade é alocar o menor número possível de dispositivos (nós de processamento, comunicação, etc) para os serviços contratados e manter os demais desligados enquanto não forem requisitados. Isso melhora a escalabilidade do sistema e evitando o desperdício de recursos.

A partir da premissa de que máquinas virtuais consomem uma quantidade de recursos (memória, armazenamento, etc) de máquinas físicas e que essas, por sua vez, possuem uma quantidade disponível limitada dos mesmos recursos, tem-se um problema de otimização em que cada máquina virtual deve ser alocada em uma máquina física, obedecendo a quantidade de recursos disponíveis por essa, buscando minimizar o número total de dispositivos ativos. Quando o objetivo do problema compreende, além de otimizar o número de máquinas físicas utilizadas, otimizar também o uso da rede do sistema, é necessário considerar, além das dimensões de recursos, a malha de comunicação entre as máquinas virtuais e a topologia de rede do sistema. Este é um típico problema de otimização combinatória, em que se tem restrições (disponibilidade de recursos de máquinas físicas e demandas de máquinas virtuais) e funções objetivos (minimizar número de máquinas ativas, minimizar tráfego em rede, etc). Aplicar um método exato para a busca de uma solução ótima para este problema demandaria um tempo inviável, assim, os trabalhos do estado da arte utilizam-se de métodos heurísticos para busca de soluções aproximadas, que nem sempre retornam um bom resultado para todos os casos.

O caso em especial tratado neste problema é o de sistemas heterogêneos em relação a capacidade das máquinas físicas, em que se busca minimizar o número de máquinas físicas ativas (que hospede pelo menos uma máquina virtual) e o custo de rede total do sistema (soma total de *bytes* transmitidos por cada *switch*). Apesar de já haver algoritmos voltados para este problema, não há uma preocupação de como as capacidades disponíveis e o posicionamento de máquinas físicas na rede pode afetar o processo de busca por soluções eficientes para ambos os objetivos.

1.2 Objetivo

O processo de alocação de máquinas virtuais é parte vital de um sistema de computação em nuvem. Neste trabalho o objetivo foi propor uma solução para a alocação de máquinas virtuais que considere o volume de dados a serem transferidos e o número de máquinas físicas ativas como parâmetros principais no processo decisório. Isso é importante pois muitos provedores de computação em nuvem oferecem soluções para aplicações que demandam processamento de grandes massas de dados, tal como mineração de dados e computação científica. Pela característica distribuída de ambientes em nuvem esse tipo de aplicação gera um grande tráfego de dados entre máquinas virtuais. A solução proposta busca arranjar estas máquinas virtuais dinamicamente de forma a reduzir o número de saltos entre as mesmas e o número total de máquinas físicas utilizadas em seu provisionamento.

1.3 Organização do texto

O restante do texto começa com uma revisão bibliográfica sobre sistemas de computação em nuvem no Capítulo 2. A seguir, no Capítulo 3, é apresentado o problema de alocação de máquinas virtuais juntamente com os principais algoritmos para a sua resolução encontrados na literatura.

O Capítulo 4 trás a descrição da especificação do algoritmo proposto, incluindo sua fundamentação, definição de procedimentos e uma visão geral de sua implementação.

Após a descrição do algoritmo o Capítulo 5 contém um conjunto de testes para sua validação. Por fim, no Capítulo 6 são apresentadas as conclusões e considerações finais a respeito do trabalho desenvolvido.

2 Sistemas de computação em nuvem

Neste capítulo serão apresentados os conceitos básicos sobre computação em nuvem. Isso inclui a definição dos tipos de serviços oferecidos, virtualização e ferramentas de gerenciamento e implantação.

2.1 Conceitos sobre computação em nuvem

O termo *nuvem* tem sido historicamente utilizado como uma metáfora para *Internet*. É comum encontrar em diagramas a *Internet* representada pela ilustração de uma nuvem (RITTINGHOUSE; RANSOME, 2009). Dessa forma, o termo computação em nuvem refere-se a serviços de computação (armazenamento, infraestrutura, processamento) que são contratados e gerenciados por meio da *Internet*.

Para alguns estudiosos, o advento da computação em nuvem faz parte de um processo natural de evolução da computação. Assim como, no século passado, fábricas deixaram de ter estações de geração de energia elétrica própria para utilizar uma rede pública, empresas atualmente deixam de implantar recursos computacionais próprios para a fazer a locação de recursos remotamente (RITTINGHOUSE; RANSOME, 2009). Esse fenômeno decorre do fato de que o preço da aquisição, configuração e manutenção de infraestruturas de TI muitas vezes é muito maior que o preço da locação de recursos de um provedor de computação em nuvem.

Um sistema de computação em nuvem é capaz de oferecer uma grande variedade de serviços, atendendo a diversos nichos de mercado. Para o usuário comum existem serviços de soluções de *softwares*, armazenamento e *backup* de arquivos pessoais, etc. Muitas empresas de desenvolvimento de IoT também possuem seu próprio conjunto de serviços em nuvem para seus usuários.

Sistemas de computação em nuvem também contemplam soluções em nível empresarial, como plataformas para desenvolvimento e implantação de aplicações e servidores virtuais para configuração arbitrária. A ideia é oferecer uma infraestrutura computacional para que um cliente empreendedor possa

desenvolver uma aplicação, que será utilizada por muitos usuários. Assim, tem-se um sistema de relações em que o cliente contrata a infraestrutura computacional para implantar uma aplicação e essa aplicação é então distribuída para outros usuários. Neste caso é também comum que desenvolvedores e usuários finais estejam contidos no mesmo grupo de pessoas, como por exemplo, quando fazem parte do mesmo corpo de funcionários de uma empresa. Essa é uma situação comum que ocorre quando empresas desenvolvem sistemas para gestão de processos internos e hospedam os mesmos em alguma nuvem.

Essas diferentes soluções são classificadas em três diferentes classes de serviço: *Software as a Service (SaaS)*, *Platform as a Service (PaaS)* e *Infrastructure as a Service (IaaS)* (BUYYA; BROBERG; GOSCINSKI, 2011). Nas seções a seguir serão apresentados aspectos mais detalhados sobre computação em nuvem, bem como as três diferentes classes de serviços mencionadas.

2.1.1 Características

Um sistema de computação em nuvem é um sistema de computação autonômica de ofertas de serviços de computação. Para tanto, segundo Mell e Grance (2011), o mesmo apresenta algumas características e funcionalidades. São essas:

- **Capacidade de autoatendimento:** o sistema oferece uma interface para contratação, gerenciamento e pagamento de serviços sem haver necessidade de interação humana;
- **Acesso remoto:** o cliente é capaz de acessar e gerenciar os recursos contratados remotamente por meio de uma rede de computadores, geralmente a *Internet*. O acesso pode ser realizado utilizando computadores pessoais ou dispositivos móveis;
- **Elasticidade:** a quantidade de recursos pode ser alterada para ajustar a demanda, possibilitando o aumento ou redução dos mesmos de uma maneira ágil;
- **Medição de uso:** computação em nuvem trabalha com a ideia de serviço sob demanda, ou seja, o sistema é capaz de redimensionar os recursos para

ajustar a necessidade de seu cliente. Assim o provedor utiliza métricas de uso de seus serviços a fim de cobrar um valor proporcional;

- **Transparência:** o máximo que um cliente de um sistema de computação em nuvem conhece a respeito da localização de seus recursos contratos é o país ou continente em que este se encontra. Demais detalhes, como quantidade de unidades de processamento, topologia da rede, etc., são transparentes a seus clientes.

Quanto a disponibilidade de serviços, um sistema de computação em nuvem pode ser classificado nas seguintes categorias:

- **Nuvem pública:** São sistemas que oferecem serviços a qualquer pessoa que queira utilizá-los (não necessariamente gratuitos). São exemplos de nuvens privadas *Amazon Web Services*, *Google Cloud* e *Microsoft Azure*;
- **Nuvem privada:** São sistemas que têm seus serviços disponíveis a um grupo restrito de pessoas. Esses sistemas são utilizados em ambientes corporativos e acadêmicos para oferecer serviços de computação apenas às pessoas ligadas a essa empresa ou instituição (funcionário, professores, alunos, etc.);
- **Nuvem híbrida:** São sistemas que integram uma nuvem pública a uma nuvem privada e por isto têm sua disponibilidade restrita. Neles os recursos de uma nuvem pública são contratados para complementar a demanda de seus utilizadores, uma vez que muitos sistemas de nuvem privada possuem capacidade limitada. Assim, para lidar com picos de demanda utiliza-se do provisionamento de recursos de uma nuvem pública. O emprego desta técnica é conhecido como *cloud bursting* (BUYYA; BROBERG; GOSCINSKI, 2011);
- **Nuvem federada:** Similar a ideia de nuvem híbrida, uma nuvem federada é um sistema público integrado de duas ou mais nuvens públicas. Dessa forma, um provedor pode solicitar a alocação de VMs de seus clientes no *datacenter* de outros provedores que integram um dado sistema de computação em nuvem federado. Esse processo ocorre de maneira transparente aos seus clientes.

2.1.2 Classes de serviço

Como antecipado no início desta seção, os serviços oferecidos por um sistema de computação em nuvem podem ser classificados em três categorias diferentes. São essas:

- ***Infrastructure as a Service (IaaS):*** Nesta classe de serviço o provedor oferece ao cliente infraestrutura de *hardware* para qualquer finalidade arbitrária, sendo a este delegada a responsabilidade de gerenciamento do recurso. Assim, o cliente tem maiores privilégios sobre o servidor, tais como a instalação arbitrária de pacotes de softwares, anexação de discos virtuais ou implementação de políticas de *firewall* (BUYYA; BROBERG; GOSCINSKI, 2011);

- ***Platform as a Service (PaaS):*** Neste modelo de serviço o provedor disponibiliza ao cliente um ambiente para desenvolvimento e implantação de aplicações de forma que não seja necessário conhecer detalhes inerentes à plataforma que está utilizando (BUYYA; BROBERG; GOSCINSKI, 2011). Desse modo, fica a cargo do sistema o gerenciamento e configuração de recursos;

- ***Software as a Service (SaaS):*** Esta é a classe de serviço voltada para usuários comuns de computação. Este modelo de serviço consiste na oferta de *softwares* como editores de documentos, *mail user agents*, etc., implementados em plataforma *Web*, que podem ser acessados utilizando apenas um navegador. A ideia deste tipo de serviço é oferecer uma alternativa ao modelo tradicional de aquisição de *softwares*, em que o usuário adquire o *software* e o instala em seu computador pessoal (RITTINGHOUSE; RANSOME, 2009).

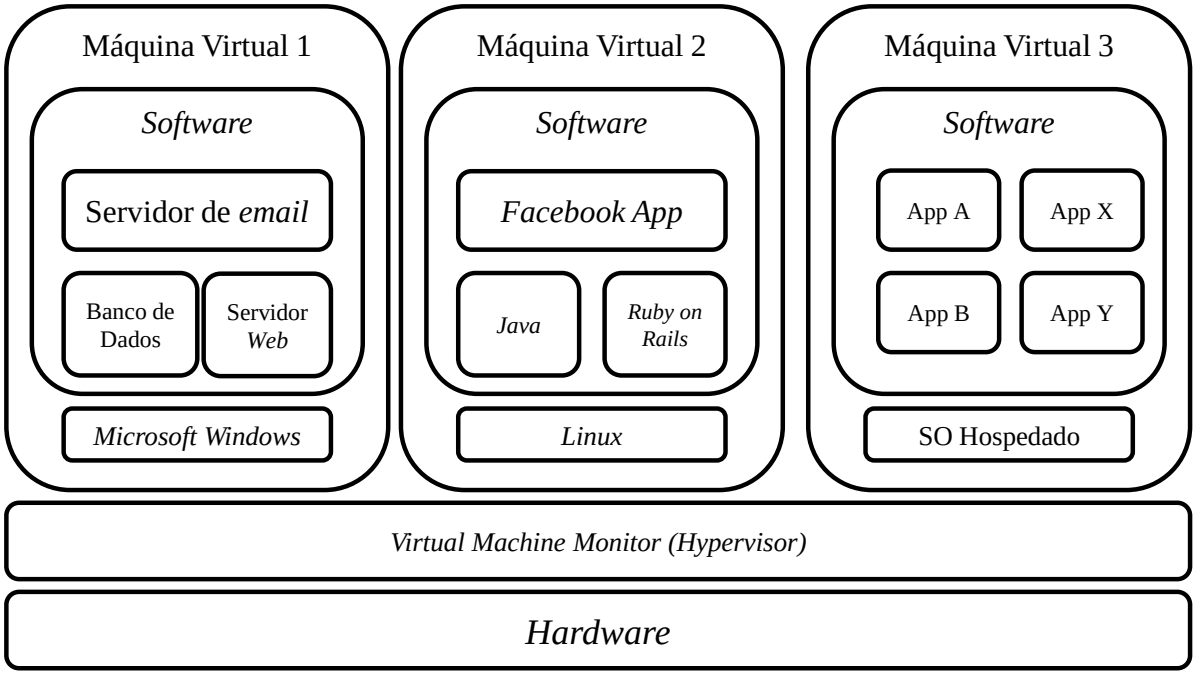
2.2 Virtualização de hardware

A infraestrutura física de um sistema de computação em nuvem consiste em um grande número de máquinas interconectadas compondo uma grade computacional. Como mencionado, o advento da computação em nuvem veio do fato

de muitas dessas máquinas estarem, boa parte do tempo, ociosas, e assim, estabelecer um novo modelo de negócio com a locação de parte dessa infraestrutura computacional. No entanto, a alocação de recursos não pode ser estática, pois clientes possuem diferentes necessidades, tanto em relação ao poder computacional como quanto à plataforma do sistema ou ao momento em que demandará recursos.

A solução para esse problema é a virtualização de *hardware*. A virtualização permite agrupar na mesma máquina física (*host*) diferentes máquinas virtuais, possivelmente com diferentes sistemas operacionais. A aplicação responsável pela instanciação e mediação de acesso às máquinas virtuais é denominada *Virtual Machine Monitor* (VMM) ou *hypervisor*. Na Figura 1 é ilustrado um caso de virtualização, em que três máquinas virtuais com diferentes sistemas operacionais são hospedadas no mesmo *host*.

Figura 1 – Hospedagem de máquinas virtuais com diferentes sistemas operacionais (BUYYA; BROBERG; GOSCINSKI, 2011)



(Traduzido pelo autor)

O uso de virtualização trás inúmeros benefícios para a operabilidade do sistema. Tem-se, por exemplo, a facilidade de replicação de servidores, conferindo maior escalabilidade de recursos, e o isolamento entre máquinas virtuais, conferindo maior segurança ao sistema (BUYYA; BROBERG; GOSCINSKI,

2011).

2.3 Ferramentas de implantação

A parte lógica de um sistema de computação em nuvem é composta por um conjunto de *softwares* específicos para automação do processo de virtualização e de seu monitoramento. Para o monitoramento de virtualização, a aplicação responsável por essa funcionalidade é denominada *hypervisor*. Já a automação do processo é feita pelo que se chama *sistema gestor*. Um sistema gestor funciona de forma distribuída, realizando o escalonamento de instâncias de máquinas virtuais nos *hosts* (máquinas físicas) que compõem o sistema. Além disso, é o sistema gestor que provê a comunicação com o *hypervisor* de cada máquina física. Neste seção serão apresentados alguns *hypervisors* e sistemas gestores.

2.3.1 *Hypervisors*

Hypervisor é a aplicação responsável por mediar o acesso às máquinas virtuais do sistema hospedeiro, sendo parte vital do funcionamento da camada de software de um sistema de computação em nuvem. A seguir são apresentados os *hypervisors* mais conhecidos comercial e cientificamente.

- ***VMWare:*** desenvolvido e mantido pela empresa homônima *VMWare Inc.*, a ferramenta possui diversas distribuições, desde ambientes para virtualização de *desktop*, como o *VMWare Workstation*, até para a implantação de servidores de grande porte, como o *VMWare ESX Server*, que consiste em um sistema operacional especializado em virtualização (VMWARE, 2018).

- ***Xen:*** a priori, *Xen* foi desenvolvido como um projeto de código aberto pela Universidade de Cambridge. Mais tarde foi comprado pela *Citrix System Inc*, que, juntamente com outras grandes empresas, suporta o desenvolvimento em código aberto do projeto e distribui em paralelo versões de código fechado do produto. Suas distribuições de código aberto são largamente utilizadas em plataformas de implantação de nuvens de código aberto (XEN, 2018).

- ***KVM:*** o *Kernel-based Virtual Machine (KVM)* é um *hypervisor* integrado a sistemas operacionais *Linux*. Por se tratar de uma ferramenta diretamente acoplada ao núcleo do sistema operacional, permite a virtualização assistida por *hardware*, que implica em ganhos de performance em relação à paravirtualização (KVM, 2019).

- ***QEMU:*** ferramenta de código aberto mantida por uma comunidade de contribuidores, possui distribuições para *Windows*, *Mac Os* e vários sistemas operacionais *Linux*. (QEMU, 2018)

- ***Microsoft Hyper-V:*** desenvolvido pela *Microsoft*, encontra-se integrado em distribuições do sistema operacional *Windows Server*. Também permite virtualização assistida por *hardware*. (HYPER-V, 2018)

2.3.2 Sistemas gestores

Existem diversas soluções prontas para gestão de sistemas de computação em nuvem. As ferramentas mais populares para esse propósito são:

- ***OpenStack:*** mantido por uma comunidade de profissionais que incluem grandes empresas do ramo de TI, tais como IBM, Intel, Red Hat Inc., o *OpenStack* é uma plataforma de código aberto composta por diferentes *softwares* para diferentes finalidades, tais como provisionamento de recursos, implantação de redes virtuais, diretório distribuído, etc. Também possui suporte aos *hypervisors VMWare*, *KVM*, *QEMU*, *Xen*, *Microsoft Hyper-V* e *Docker*. Suporta a integração com serviços de computação em nuvem da *Amazon* (OPENSTACK, 2018);

- ***OpenNebula:*** apresentada como uma ferramenta coesa e de fácil instalação e manejo, o *OpenNebula* é uma plataforma de código aberto voltada para a implantação de sistemas de computação em nuvem empresariais. Suporta os *hypervisors KVM*, *QEMU* e *VMWare* (OPENNEBULA, 2018);

- ***Eucalyptus:*** acrônimo de *Elastic Utility Computing Architecture for Linking Your Programs To Useful Systems* (NURMI et al., 2009), é uma plataforma de código aberto de implantação de sistema de computação

em nuvem híbrido, com suporte para integração com serviços da *Amazon*. Oferece suporte aos *hypervisors KVM*, *VMWare* e *Xen*;

- ***Apache CloudStack:*** mantido pela *Apache*, é um software de código aberto que, além da implantação de um sistema de nuvem privado, suporta a integração com serviços de computação em nuvem da *Amazon*. A plataforma também suporta diversos *hypervisors*, tais como *KVM*, *VMWare*, *OracleVM*, *Xen* e *Microsoft Hyper-V* (CLOUDSTACK, 2018);

- ***System Center Virtual Machine Manager:*** distribuído pela *Microsoft*, é um software de código fechado utilizado em sistemas *Windows*. Também conhecido pela sigla *SCVMM*, a ferramenta opera juntamente com o *hypervisor Microsoft Hyper-V* (MICROSOFT, 2018).

2.4 Infraestrutura e organização

Sistemas de computação em nuvem, principalmente os sistemas de nuvem pública, são geralmente implantados sobre grandes *datacenters* com dezenas de milhares de nós de processamento. No entanto, alguns sistemas de nuvem privada também podem ser implantados em pequenos *datacenters* com algumas dezenas de nós de processamento.

No caso de grandes corporações, como *Google*, *Amazon*, *Microsoft*, etc., a rede de serviços é composta por grandes *datacenters* geograficamente distribuídos. A seguir são discutidos aspectos sobre organização e rede de comunicação de *datacenters* que compõem grandes sistemas de computação em nuvem.

2.4.1 Arquitetura de rede de *datacenters*

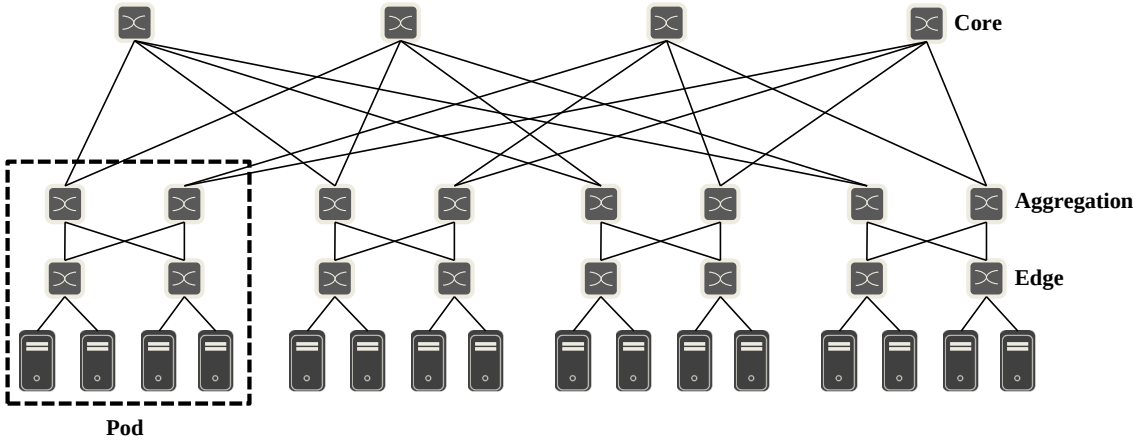
A infraestrutura de rede comunicação em *datacenters* de sistemas de computação em nuvem segue os padrões de sistemas de computação de alto desempenho, ou seja, é composta por nós de processamento (máquinas físicas) interconectados por elementos de comunicação.

Redes *fat-tree* (LEISERSON, 1985) são uma das topologias comumente empregadas em redes de *datacenters* de sistemas de computação em nuvem (ZHANG; CHENG; BOUTABA, 2010). Como seu nome sugere (árvore gorda

em português), trata-se de uma topologia baseada em árvore, em que os nós de processamento são dispostos nas folhas e seus respectivos nós pais, avós, etc. são compostos por elementos de comunicação. O diferencial nesta topologia para uma árvore comum é que, neste estrutura, quanto mais próximo da raiz da árvore, maior é a largura de banda da rede. Uma rede em *fat-tree* é composta por três tipos de *switches*, organizados em camada na forma: *edge*, *aggregation* e *core*. *Switches* do tipo *edge* conectam os nós de processamento aos *switches* tipo *aggregation*, enquanto estes conectam *switches edge* aos *switches* do tipo *core*.

Um exemplo de implementação da topologia *fat-tree* é a arquitetura *PortLand* (AL-FARES; LOUKISSAS; VAHDAT, 2008), composta por *switches* de mercado (*commodity*). Esta estrutura emprega o conceito de redundância, provendo mais de um caminho para o mesmo destino, tanto para lidar com falhas, quanto para melhor balancear o tráfego, minimizando o congestionamento na rede. Um conjunto de nós de processamento que apresentam três ou menos saltos entre cada par, junto de *switches edges* que os conectam e os *aggregation* que conectam estes mesmo *edges* é denominado *pod*. Na Figura 2 é ilustrado um *datacenter* com essa arquitetura formado por *switches* de 4 portas.

Figura 2 – Ilustração do *datacenter PortLand* formado por *switches* de 4 portas



A dimensão de um *datacenter PortLand* depende do número de portas de seus *switches*. Assim, sendo k o número de portas de cada *switch*, um *datacenter* completo possuirá k *pods*, $k^2/2$ *edges*, $k^2/2$ *aggregations*, $k^2/4$ *cores* e $k^3/4$ nós de processamento. Assim sendo, um *datacenter PortLand* formado por *switches* de 48 portas pode ter até 27648 nós de processamento (AL-FARES; LOUKISSAS;

VAHDAT, 2008). Há também outras alternativas de arquiteturas, como BCube (GUO et al., 2009) e VL2 (GREENBERG et al., 2009).

Quanto ao endereçamento de VMs na rede, há dois modos possíveis de configuração da placa de rede de seus *hosts*: *NAT* e *bridge*. No modo *NAT* as VMs são endereçadas utilizando redirecionamento de portas por meio de uma tabela *NAT* de forma que uma VM fique visível apenas para o *host* que a hospeda e para as demais VMs hospedadas também por esse *host*. No modo *bridge* o *host* passa a funcionar como um comutador para as VMs fazendo com que as mesmas fiquem visíveis dentro da rede recebendo um endereço *IP* interno e eventualmente um externo.

Para sistemas de computação em nuvem, o modo de endereçamento *bridge* é a solução mais adequada em se tratando do provisionamento de VMs para seus clientes enquanto o endereçamento por *NAT* destina-se a aplicações locais.

2.4.2 Estrutura lógica

O processo de alocação de VMs no sistema ocorre de forma automática, realizado pelo sistema gestor implantado em seu *datacenter*. O modelo mais comum de organização em um sistema genérico é o de mestre-escravo. É dito nó mestre aquele responsável por coordenar as ações estratégicas. Os nós escravos, também dito *hosts*, são responsáveis pela hospedagem das VMs.

Toda a comunicação do sistema é realizada por meio de troca de mensagens. Assim, os nós escravos são responsáveis pela geração de relatório a respeito da execução e comunicação de suas VMs e envio dos mesmos para o nó mestre, para que este possa tomar decisões de escalonamento.

Um sistema gestor de computação em nuvem também provê outras funcionalidades, tais como (BUYYA; BROBERG; GOSCINSKI, 2011):

- **Distribuição de imagens:** VMs são instanciadas utilizando *templates*, que nada mais são que discos virtuais formatados com o sistema operacional desejado. O *host* escolhido para hospedar uma dada VM deve, de alguma forma, conter a imagem relativa a mesma. O serviço de distribuição de imagens é responsável por armazenar e dispor estes arquivos na rede;

- **Redes de VMs:** para oferecer uma melhor experiência para usuários, sistemas de computação em nuvem também podem fornecer automação do processo de criação de redes virtuais entre VMs, de forma a criar domínios de *broadcast* entre as mesmas;

- **Sistema de informação:** sistemas de computação em nuvem também demandam uma base de dados com informações gerais sobre seus clientes e serviços, bem como a quantidade de uso de recursos a fim de efetuar as devidas cobranças;

- **Painel de autoatendimento:** essa funcionalidade é destinada a clientes, para que estes criem contas, contratem serviços, etc. São implementados por meio de páginas *web* que podem ser acessadas remotamente pela *Internet*;

- **Painel administrativo:** Trata-se de uma interface gráfica, implementada por meio de páginas *web*, restrita a administradores para a realização de funções administrativas relativas à configuração do sistema e de sua infraestrutura.

Existem outras funcionalidades além dessas listadas, principalmente no núcleo do sistema, como auditoria administrativa, gerenciamento de falhas e alocação de VMs. Esta última, como é o objeto de estudo deste trabalho, será tratada no próximo capítulo.

2.5 Considerações finais

O projeto de um sistema de computação em nuvem não é uma tarefa simples, pois esses sistemas são complexos e demandam diversas camadas de *softwares* especializados para sua autogestão. Além disso, o sistema deve ser dinâmico, no sentido de alocar diversas máquinas virtuais com diferentes sistemas operacionais, de acordo com a demanda de clientes. Essa complexidade leva aos sistemas descritos neste capítulo e também aos problemas que serão tratados na sequência sobre a otimização da alocação de VMs.

3 Alocação de máquinas virtuais

A alocação de máquinas virtuais é um processo crucial para o funcionamento de sistemas de computação em nuvem. Esse processo pode ser entendido como um problema de escalonamento em uma grade computacional, uma vez que a alocação de máquinas virtuais consiste em escolher uma máquina física (*host*) da grade, com base em critérios definidos no projeto do sistema, para hospedar a VM. No restante deste capítulo serão apresentados os conceitos necessários para entendimento do problema de alocação, bem como soluções para a sua resolução encontradas na literatura.

3.1 Conceitos básicos

Como indicado, a alocação de uma VM é fundamentalmente um problema de otimização combinatória. Isso significa que para a sua solução é preciso definir objetivos e restrições do sistema. Nesta seção serão discutidos que tipos de objetivos e restrições são encontrados e algumas das técnicas básicas para a solução de problemas de otimização combinatória.

3.1.1 Parâmetros para alocação de máquinas virtuais

A especificação de uma VM a ser alocada depende do serviço contratado para a qual esta será provisionada. As propriedades dessa máquina são definidas por variáveis dimensionais, tais como quantidade de memória principal (RAM), poder de processamento e por vezes largura de banda e latência. Portanto, um *host* deve ter disponível a quantidade necessária dos componentes específicos para estar apto a hospedar uma dada VM. Vale a pena salientar que toda vez que um *host* hospeda uma VM, a quantidade disponível de cada um desses componentes é reduzida de acordo com a demanda da VM hospedada.

O processo de alocação de VMs pode ser feito de duas diferentes formas. São essas:

- **Criação de uma nova instância:** Esse procedimento nada mais é que

alocar uma nova VM no sistema. A criação de uma nova VM ocorre em duas situações: Quando o cliente solicita um novo recurso ou quando um recurso já provisionado é clonado. A clonagem de uma instância pode ocorrer quando o cliente solicita explicitamente ou quando o sistema realiza *autoscaling*;

- **Migração de uma VM:** O procedimento de migração consiste em transportar uma VM de um *host* para outro. O sistema pode realizar esse procedimento por vários motivos como, por exemplo, para atualização do sistema da máquina hospedeira ou na aplicação de estratégias de otimização de distribuição de recursos.

Ambas as formas constituem processos críticos para o funcionamento do sistema e possuem um grande impacto no crescimento e expansão do negócio para o qual os recursos são provisionados. Existem na atualidade técnicas eficazes empregadas na alocação de VMs para realizar essa tarefa em questão de segundos, buscando garantir qualidade de serviço aos seus usuários (BUYYA; BROBERG; GOSCINSKI, 2011).

A seguir são apresentadas maneiras comuns de tratar o problema de alocação de VMs bem como algoritmos do estado da arte que se baseiam nestas abordagens.

3.1.2 Classificação do problema

A alocação de máquinas virtuais é tratada como um problema de otimização combinatória de um ou mais objetivos, os quais podem ser a redução do gasto de energia elétrica, redução do tráfego na rede, maximização do tempo de processamento de aplicações, etc. A seguir são apresentadas classes de problemas de otimização nas quais são comumente enquadrados problemas de alocação de máquinas virtuais.

3.1.2.1 Problema de empacotamento

A alocação de máquinas virtuais pode ser entendida como um problema de empacotamento (*bin-packing* ou apenas BP) (COFFMAN; GAREY; JOHNSON,

1984) aplicado ao contexto de computação em nuvem, em que itens com um dado volume devem ser colocados dentro de finitos pacotes que por sua vez possuem uma capacidade finita. No contexto de computação em nuvem, os pacotes referem-se às máquinas físicas, os itens às máquinas virtuais e medidas de volume ao poder computacional (memória, velocidade de processamento, etc).

O problema de empacotamento é um problema de otimização combinatória NP-difícil. Assim, é muito comum a utilização de heurísticas para a obtenção de soluções aproximadas (COFFMAN; GAREY; JOHNSON, 1984).

3.1.2.2 Problema quadrático de alocação

Em um problema quadrático de alocação (*quadratic assignment problem* ou apenas QAP) tem-se n instalações que devem ser alocadas em n locações. Entre cada instalação existe um fluxo e para cada locação um custo (LAWLER, 1963). No contexto de alocação de VMs, as instalações referem-se às VMs e as locações a porções de recursos disponíveis para alocação em máquinas físicas. Dessa forma, o fluxo entre instalações representa a troca de dados entre VMs e o custo entre locações representa o custo de comunicação entre máquinas físicas.

Dessa maneira, métodos de otimização baseados no problema quadrático de alocação são empregados para otimização de rede de computadores, buscando a redução de latência, redução de saltos entre máquinas virtuais, etc.

O problema quadrático de alocação se enquadra na classe de problemas de otimização combinatória NP-Difíceis. Para pequenas instâncias, o problema pode ser resolvido por meio de métodos exatos, como programação dinâmica (BERTSEKAS et al., 1995) e *branch-and-bound* (LAWLER; WOOD, 1966). Para grandes instâncias do problema, o emprego de métodos exatos demanda um tempo computacional alto, tornando inviável sua aplicação. Para estes casos, são empregados métodos heurísticos, que buscam soluções aproximadas em um tempo aceitável.

A maneira mais comum de resolver este tipo de problema é implementar uma solução inicial e permutar sua configuração buscando melhorar a solução final. Tem-se como exemplo, o algoritmo de Armour e Buffa (1963), que busca

por soluções localmente ótimas.

3.2 Algoritmos de alocação de máquinas virtuais

Toda vez que a alocação de uma máquina virtual é solicitada, o sistema recorre ao seu escalonador para escolher qual das máquinas físicas hospedará essa VM. Obviamente, o funcionamento de escalonadores é regido por algum algoritmo de escalonamento. Muitos desses algoritmos têm como objetivo melhorar o desempenho do sistema em algum aspecto específico, tal como economia de energia, economia de memória, melhoria da qualidade de serviço, etc. No caso de sistemas de computação em nuvem esse problema é ainda mais difícil pois esses sistemas são majoritariamente compostos por uma grade de máquinas, que podem estar geograficamente distribuídas.

A seguir, são apresentados algoritmos genéricos para alocação de VMs juntamente com algoritmos encontrados na literatura.

3.2.1 Soluções heurísticas de empacotamento

Muitos algoritmos do estado da arte são desenvolvidos com base em soluções genéricas para o problema de empacotamento, como as apresentadas a seguir:

- ***First-Fit (FF):*** O algoritmo *First-Fit* é uma abordagem gulosa para o problema de empacotamento. Neste algoritmo um dado item a ser empacotado será atribuído para o primeiro pacote encontrado que esteja apto a recebê-lo. O objetivo desta abordagem é utilizar o menor número de pacotes possíveis, para tanto, o processo de busca de pacotes deve seguir sempre o mesmo sentido na listagem de todos os pacotes.

- ***First-Fit Decreasing (FFD):*** Trata-se de uma variação do *First-Fit*, no qual a lista de itens a serem empacotados é ordenada de forma decrescente em relação a alguma dimensão arbitrariamente escolhida e assim atendida nesta ordem utilizando o princípio do algoritmo *First-Fit*.

- ***Best-Fit (BF):*** Neste algoritmo o pacote escolhido para receber um dado item é aquele que, ao receber a item, terá capacidade restante menor do que os demais. Para tanto, o algoritmo deve calcular o restante da capacidade que cada pacote terá ao receber o item a ser empacotado.

- ***Best-Fit Decreasing (BFD):*** O mesmo método de escolha de pacotes do algoritmo *Best-Fit* é utilizado, porém com a lista de itens organizada de forma decrescente de acordo com alguma dimensão arbitrariamente escolhida.

- ***Worst-Fit (WF):*** Neste algoritmo o pacote escolhido para receber um dado item é aquele que, ao receber o item, terá capacidade restante maior do que os demais pacotes caso recebessem esse item. Assim como ocorre no algoritmo *Best-Fit*, no algoritmo *Worst-fit* é preciso calcular o restante da capacidade que cada pacote terá ao receber o item a ser empacotado.

- ***Worst-Fit Decreasing (WFD):*** Nesta variação do algoritmo *Worst-Fit* a lista de itens é organizada em ordem decrescente e então são empacotados segundo o critério do algoritmo *Worst-fit*.

- ***Next-Fit (NF):*** O algoritmo *Next-Fit* realiza a distribuição de itens de forma circular aos seus respectivos destinos, similar o conceito de *Round-Robin*. Mais especificamente, o algoritmo organiza os pacotes em uma lista circular e distribui os itens circularmente de forma que para cada dois itens consecutivos, o algoritmo deve tentar empacotá-los em diferentes pacotes em sentido circular.

3.2.2 Soluções meta-heurísticas

Problemas de otimização combinatória geram, para diferentes configurações de sistema, espaços de busca por soluções que precisam ser percorridos na busca pela solução ótima. Meta-heurísticas são métodos genéricos de busca por soluções para estes problemas, podendo ser empregadas tanto nos problemas citados neste texto (BP e QAP) quanto a outros problemas de otimização combinatória.

Diferente de métodos exatos, meta-heurísticas possuem complexidade computacional relativamente baixa e buscam por soluções localmente ótimas ou próximas do ótimo global. São alguns dos exemplos mais populares de meta-heurísticas:

- **Algoritmo genético:** Inspirado por processos evolutivos da biologia, algoritmos genéticos implementam técnicas que simulam processos de mutação, seleção natural, combinação de genes, etc., para gerar soluções cada vez melhores para o problema no qual é aplicado. O algoritmo inicia o processo de otimização com diferentes configurações para o sistema, arbitrariamente implementadas, e aplica os processos genéticos citados para obter melhores soluções (WHITLEY, 1994).

- **Colônia de formigas:** Este método inspira-se no processo de busca de alimentos por colônias de formigas. Inicialmente, formigas percorrem caminhos aleatórios em busca de alimentos. Ao encontrar uma fonte de alimentos, estas retornam a sua colônia deixando rastros de feromônio, que indicam um caminho a seguir para as demais formigas da colônia. Analogamente, o algoritmo inicia o processo de otimização percorrendo caminhos aleatórios, deixando "rastros" para caminhos que convergem para melhores soluções. Uma vantagem deste algoritmo é que, na vida real, rastros de feromônio podem evaporar fazendo com que o caminho se torne cada vez menos atrativo, este conceito, aplicado ao processo de otimização diminui a probabilidade do algoritmo caminhar rapidamente para ótimos locais (DORIGO; BIRATTARI, 2011).

- ***Simulated annealing:*** Este método faz uma analogia ao processo de recozimento de sólidos utilizado na metalurgia. Em uma primeira fase, o material é aquecido a uma alta temperatura. Após atingir a temperatura adequada, o material é então resfriado lentamente a fim de se obter uma configuração homogênea das moléculas do produto. Aplicado ao problema de otimização, o algoritmo utiliza uma solução inicial e uma variável que representa a temperatura, inicialmente alta, que se decrementa iterativamente. A temperatura neste caso funciona como um fator de aleatoriedade em que quanto maior seu valor, maior a probabilidade do algoritmo selecio-

nar uma solução que esteja na contra-mão da função objetivo no espaço de busca, de forma a evitar uma convergência muito rápida para um mínimo local (LAARHOVEN; AARTS, 1987).

- **Busca tabu:** Neste método inicia-se a busca pela vizinhança da solução atual, atualizando a solução atual para um determinado vizinho, caso este seja uma solução melhor que a atual ou não haja mais vizinhos melhores que a atual que ainda não foram visitados. Para realizar esta operação, o algoritmo mantém uma lista denominada tabu, que contém todas as alternativas de soluções já verificadas (GLOVER, 1989). Sua grande vantagem é a eficiência em se aproximar do mínimo global, escapando de mínimos locais, embora demande um tempo de computação maior que as demais meta-heurísticas.

3.2.3 Estado da arte

O estado da arte é composto majoritariamente por abordagens heurísticas baseadas em empacotamento e meta-heurísticas, que buscam atender a diversos objetivos, como minimização do número de *hosts* ativos, minimização do tráfego em rede, etc. A seguir são apresentados alguns desses trabalhos:

3.2.3.1 Remanejamento de VMs por predição

Denominado *Measure-Forecast-Remap (MFR)* (BOBROFF; KOCHUT; BEATY., 2007), o objetivo do algoritmo é minimizar o tempo médio de utilização de *hosts* utilizados para o provisionamento de VMs tendo como restrição a probabilidade de ocorrerem violações no SLA. Seu funcionamento pode ser particionado três em diferentes etapas:

- **Medição:** Coleta de dados acerca das atividades no sistema e estabelecimento de um histórico.
- **Predição:** Uso de métodos para análise de séries temporais para determinar demandas futuras de recursos para VMs.
- **Rearranjo:** Migração de VMs baseadas nos resultados de predição a fim de satisfazer o objetivo respeitando a restrição.

O algoritmo opera dinamicamente, realizando as três etapas periodicamente. Na etapa de rearranjo é implementado um algoritmo baseado em *First-Fit Decreasing*, levando em consideração as quantidades de recursos de cada VM previstas na etapa de predição.

3.2.3.2 Programação dinâmica para eficiência energética

Denominado *Energy Efficient Virtual Machine Placement* (GOUDARZI; PEDRAM, 2012), o algoritmo tem como objeto principal reduzir gastos em energia elétrica. Este algoritmo é aplicado em casos em que é necessário replicar VMs e distribuir suas cópias pelo *datacenter*, tendo como objetivo tanto o balanceamento de carga, quanto tolerância a falhas. Assim, a ideia do algoritmo é otimizar a distribuição dessas réplicas de forma a minimizar o número de *hosts* utilizados na hospedagem das mesmas.

O algoritmo utiliza um método de resolução baseado em programação dinâmica para determinar quantas cópias serão feitas de VMs e selecionar os *hosts* mais adequados a recebê-las. Ainda assim, alguns *hosts* selecionados podem apresentar subutilização de recursos. Dessa forma, as VMs hospedadas nestes *hosts* são relocadas para outros *hosts* utilizando um método de busca local.

3.2.3.3 Aplicação do Método Húngaro com algoritmo genético

Proposto por TAO et al. (2015), o algoritmo, denominado *Binary Graph Matching-Based bucket-code Learning Algorithm (BGM-BLA)*, busca minimizar o número de *hosts* ativos, o custo de migração de VMs e a quantidade de tráfego gerada pela alocação de VMs em diferentes pontos da rede. Sua implementação utiliza modelagem de soluções em alto nível, denominadas pelo autor do trabalho como *bucket codes*.

Cada *bucket code* tem sua eficiência avaliada por meio do Método Húngaro, também denominado *Kuhn-Munkres*. O Método Húngaro é um algoritmo utilizado para a resolução do Problema de Alocação (KUHN, 1955), no qual um conjunto de tarefas, as VMs no caso de nuvem, devem ser atribuídas a um conjunto de trabalhadores, uma a uma, tal que para cada par tarefa-trabalhador existe um custo associado.

Caso a solução avaliada seja rejeitada, é aplicado algoritmo genético no *bucket code* associado para sua mutação, para que seja novamente submetido ao processo de avaliação, e assim por diante, até que se encontre uma solução ideal para o problema.

3.2.3.4 Inferência *fuzzy* e algoritmo genético

Denominada *MGGA*, acrônimo de *Multi-objective Grouping Genetic Algorithm*, a abordagem proposta por XU e FORTES (2010) utiliza um algoritmo genético para a busca de soluções e um sistema de classificação baseado em lógica *fuzzy* para avaliação dessas soluções. O *MGGA* enxerga a alocação de máquinas virtuais como um problema de otimização multiobjetivo. Os objetivos consistem em:

- **Redução de resíduo:** É dito resíduo a diferença entre a capacidade total de um *host* e a soma de recursos consumidos pelas VMs nele alocadas. No caso de *hosts* sem nenhuma VM alocada, o resíduo é 0.
- **Redução de picos de temperatura:** Quanto maior a ocupação de *hosts*, mais calor este irá gerar.
- **Redução do consumo total de energia:** Assim como a geração de calor, *hosts* com maior ocupação geram maior gasto em energia elétrica.

Assim, a decisão de alocação de VMs consiste em aplicar o algoritmo genético para encontrar um grupo de soluções relativas aos objetivos descritos e escolher a melhor solução por meio de inferência *fuzzy*.

3.2.3.5 Cortes mínimos para solução QAP

Denominado *Cluster-And-Cut* (MENG; PAPPAS; ZHANG, 2010), é um algoritmo que tem como objetivo diminuir o tráfego de dados entre máquinas físicas do sistema, de forma a alocar o mais perto possível as VMs que possuírem maior tráfego entre si. O algoritmo trata o problema de alocação de VMs como um Problema Quadrático de Alocação (QAP). O algoritmo divide os recursos (memória, processamento, etc.) em porções, ditas *slots*, de forma que em cada *slot* caiba pelo menos qualquer uma VM do sistema.

O algoritmo tem como entrada uma matriz que representa a taxa de comunicação entre as VMs e outra que representa o custo entre *hosts* (que pode ser o número de saltos entre os mesmos). Como cada *slot* pertence a um único *host*, o custo entre dois *slots* é o mesmo custo entre os *hosts* que os hospedam. Assim, o algoritmo trata VMs e *slots* como grafos, buscando um mapeamento um a um por meio das seguintes etapas:

1. Agrupar *slots* em *k clusters*;
2. Cortar o grafo de VMs em *k clusters* utilizando árvore de cortes *Gomory-Hu* (GOMORY; HU, 1961);
3. Mapear cada *cluster* de VMs para cada *cluster* de *slots*;
4. Repetir o processo para cada par de *clusters slots*-VMs, atribuindo a *k* o número de VMs em cada par.

Estas etapas são repetidas recursivamente até que mapeie-se uma VM para um *slot*. Em uma primeira execução, *k* deve ser escolhido arbitrariamente.

Para o agrupamento de *slots*, o autor recomenda que seja desenvolvido um algoritmo que melhor se enquadre na estrutura do *datacenter* no qual este será implantado, mas podendo também utilizar algumas abordagens genéricas, como a de Gonzalez (1985), que é um algoritmo heurístico para agrupamento de dados (clusterização).

3.2.3.6 Cortes mínimos para solução QAP e BP

Proposto por Huang et al. (2012), esta abordagem, denominada *Energy-Aware Joint VM Placement*, tem como objetivos redução do número de *hosts* ativos e redução do tráfego na rede. Apesar de denominar-se *energy-aware*, este algoritmo não quantifica gastos com energia elétrica mas parte do principio de que a redução do número de dispositivos de computação e comunicação ativos pode levar à redução de gastos em energia elétrica.

Seu funcionamento é similar ao funcionamento do algoritmo *Cluster-And-Cut*, utilizando teoria dos grafos e cortes mínimos para o mapeamento entre VMs e *slots*, por meio das seguintes etapas:

1. Cortar o grafo que representa a comunicação entre VMs em *k clusters* utilizando árvore de cortes *Gomory-Hu*;

2. Estabelecer e agrupar *slots* em *k clusters* de acordo com a dimensão de VMs e capacidades de *hosts*;

3. Mapear cada *cluster* de VMs para cada *cluster* de *slots* e avaliar a qualidade da solução de cada par através de inferência *fuzzy*. Para cada par de solução reprovado na avaliação, repetir este procedimento.

Diferente do algoritmo *Cluster-and-Cut*, em que os *slots* são pré-definidos como entrada, aqui os *slots* são definidos dinamicamente de acordo com as dimensões das VMs utilizando uma abordagem similar ao algoritmo *First-Fit*.

3.2.3.7 *Best-fit decreasing* baseado em consumo de energia

Power-aware Best-Fit Decreasing (BELOGLAZOV; BUYYA, 2012) é uma abordagem heurística baseada em *Best-Fit Decreasing* que tem como objetivo redução de gastos em energia-elétrica. Um *datacenter* pode conter diferentes máquinas de diferentes modelos. Assim, para cada modelo há um padrão de consumo de energia elétrica e, além disso, cada VM também consome uma quantidade de energia elétrica em função da sua carga computacional em relação a cada máquina. Partindo deste princípio o algoritmo realiza as seguintes etapas:

1. Busca por *hosts* sobrecarregados ou subutilizados e seleciona suas VMs para migração;

2. Ordena as VMs selecionadas em ordem decrescente por utilização de CPU;

3. Para cada VM da lista seleciona um *host* que a comporte e que incremente o menor valor ao consumo total de energia elétrica.

3.2.3.8 Clusterização hierárquica com cortes mínimos

Proposto por Dong, Wang e Cheng (2015), este algoritmo, denominado *MinCut Hierarchical Clustering*, tem como objetivos minimizar o número de

hosts ativos e o número de dispositivos de comunicação utilizados. O algoritmo trata o problema sob diferentes perspectivas em diferentes estágios. Para a otimização de tráfego, o problema é enxergado sob a perspectiva de um Problema Quadrático de Alocação e para a otimização do número de *hosts* ativos, o problema de alocação de VMs é tratado como um Problema de Empacotamento.

Em um primeiro estágio, para a otimização de tráfego é utilizada uma abordagem baseada em agrupamento hierárquico por cortes mínimos, separando VMs em diferentes *clusters* de acordo com sua taxa de comunicação. Após esse processo é aplicado um algoritmo baseado em *Best-fit* para alocar estas VMs de forma a otimizar o número de *hosts* utilizados.

Assim como o *Energy-Aware Joint VM Placement*, este algoritmo também tem como meta a redução do consumo total de energia elétrica do sistema reduzindo a utilização de elementos de comunicação e número de *hosts* ativos, sem quantificar numericamente gastos de energia elétrica no processo decisório.

3.2.3.9 Combinação de *First-fit* e *Kernighan-Lin*

NICE (*Network-aware virtual machine consolIdation scheme for energy conservation in data Center*) (CAO et al., 2014), é um algoritmo que busca reduzir o número de *hosts* ativos, utilização de *switches* e o custo de migração da operação de rearranjo de VMs com a aplicação da heurística *Firs-Fit Decreasing* com o algoritmo *Kernighan-Lin*, que é uma abordagem heurística para particionamento de grafos (KERNIGHAN; LIN, 1970), em diferentes etapas, buscando separar VMs em partições e mapeá-las para o sistema. São essas etapas:

1. Associação de VMs a *hosts*, mapeando cada VM para uma partição que representa um dado *host*, utilizando a heurística *First-Fit Decreasing*;

2. Aplicação do algoritmo *Kernighan-Lin* nas partições obtidas buscando realizar trocas de VMs entre cada partição de forma a aprimorar o particionamento.

3. Mapeamento das partições para os *hosts* através de uma abordagem gulosa, buscando associar VMs de uma partição p_i a um host h_j de forma a minimizar a quantidade de tráfego na rede e migração.

3.2.3.10 Redução do tempo total de utilização de *hosts*

KNAUTH e FETZER (2012) propuseram um algoritmo denominado *OptSched*, em que a decisão de alocação baseia-se em consumo de energia. O funcionamento do algoritmo busca minimizar a função *cumulative machine uptime (CMU)* (KNAUTH; FETZER, 2012). Essa função nada mais é que a somatória do tempo total de utilização de CPU de cada *host* em função de suas respectivas VMs. Minimizar essa função tem como consequência agrupar VMs em um menor número de *hosts* possível, podendo reduzir a atividade de alguns deles a zero.

Para realizar esse tipo de abordagem o algoritmo precisa conhecer períodos de atividades de grupos de VMs, para então poder encaixá-las da melhor maneira possível nos *hosts* candidatos a recebe-las. Assim, o algoritmo buscar alocar uma dada VM em um *host* que a comporte e que apresente menor diferença entre o tempo de expiração da VM a ser alocada e o tempo máximo de expiração da VM mais durativa deste *host*.

O algoritmo apresenta uma abordagem similar ao *Best-fit*, aplicando a diferença mínima da função CMU como critério de busca/alocação.

3.2.3.11 Eficiência energética e QOS para SDN

Denominado *Energy-efficient and QoS-aware Virtual Machine Placement for Software Defined Datacenter Networks (EQVMP)* (WANG et al., 2014), este algoritmo tem como objetivo principal reduzir o consumo de energia sem degradar a qualidade de serviço para *datacenters* que possuem redes definidas por *software* (SDN). O algoritmo foi proposto para redes definidas por *software* e possui três objetivos: redução de saltos entre VMs, redução do número de *hosts* utilizados, balanceamento de carga. Para cada um desses objetivos o algoritmo realiza as seguintes etapas:

1. **Redução de saltos:** Neste processo as VMs são separadas em diferentes grupos utilizando cortes mínimos em grafos.

2. **Redução de gastos de energia:** Após o processo de agrupamento o algoritmo decide em quais máquinas físicas alocar as VMs utilizando a

heurística *Best-Fit*.

3. **Balanceamento de cargas:** Como o algoritmo foi proposto para redes definidas por *software*, nesta etapa são realizadas novas configurações na rede para melhor adaptar o fluxo gerado pelo novo mapamento de VMs.

3.2.3.12 Alocação com base em meta-escalonador

Proposto por ALAHMADI et al. (2014), este algoritmo, denominado *Enhanced First-Fit Decreasing*, trata como entrada tarefas que devem ser atribuídas a VMs já alocadas ou aloca novas VMs para estas. Isso o diferencia dos demais algoritmos apresentados, que tratam como entrada apenas as VMs.

O algoritmo possui duas funções objetivo: maximizar a taxa de utilização de VMs às quais as tarefas serão submetidas e minimizar gastos em energia elétrica de cada conjunto de tarefas. O algoritmo possui dois diferentes componentes:

- **Escalonador de tarefas:** Componente que define para qual VM deve ir cada tarefa utilizando uma abordagem similar à heurística *First-fit Decreasing*, ordenando tarefas por *deadline* e VMs por sua ocupação;
- **Alocador de VMs:** Busca rearranjar VMs com base na sua demanda por recursos para evitar violações de SLA.

Este algoritmo apresenta uma abordagem diferente dos demais vistos até agora, mas que restringe sua utilização a contextos em que o provedor de serviço tem controle sobre o escalonamento de tarefas para VMs. Essa situação pode ocorrer em serviços de PaaS e SaaS, mas não em IaaS.

3.2.3.13 Alocação baseada em receita financeira

Proposto por SHI e HONG (2011), o algoritmo *Towards Profitable Virtual Machine Placement* é voltado para melhorar a receita financeira do provedor do serviço de computação em nuvem, especificamente para o serviço de IaaS.

Uma das maneiras de maximizar lucros em sistemas de computação em nuvem é reduzir gastos com energia elétrica. A ideia deste algoritmo é estabelecer

um orçamento de gastos com energia elétrica para cada máquina física, evitando problemas de extrapolação de gastos com energia elétrica e violações de SLA. Para tanto utiliza a heurística *First-Fit* para escolher onde alocar VMs, levando em conta o orçamento com gastos de energia de cada máquina física da nuvem.

3.2.3.14 Algoritmo genético e predição

Denominado *Genetic Algorithm Based Approach (GABA)* (MI et al., 2010), é uma abordagem que utiliza algoritmo genético para melhorar a utilização de máquinas físicas a fim de economizar energia elétrica, mantendo desligadas máquinas físicas que não estão sendo utilizadas. Como o processo de inicialização ou parada de VMs pode ser demorado, o algoritmo utiliza modelos estocásticos para prever a demanda por recursos, podendo assim antecipar esse processo.

3.2.3.15 Escalonamento com monitoramento de carga

Proposto por BACHIEGA et al. (2014), este algoritmo apresenta uma abordagem baseada em *Worst-fit* para alocação tanto estática quanto dinâmica de VMs. Na alocação estática o *host* escolhido é aquele que possui maior quantidade de recursos disponíveis no momento da alocação. Na alocação dinâmica se avalia periodicamente a taxa de ocupação dos *hosts* para realizar migrações quando detectar uma sobrecarga de recursos e assim migrar VMs de um *host* sobrecarregado para outro *host* com maior disponibilidade de recursos.

3.3 Visão geral dos algoritmos

Apesar dos algoritmos apresentarem objetivos diferentes, é possível identificar classes comuns de objetivos, que são:

- ***Load-aware:*** Algoritmos desta classe buscam promover o balanceamento de carga em *hosts*. Nestes algoritmos, o *software* gestor realiza periodicamente monitoramento de ocupação de *hosts*, podendo realizar migrações para balancear a carga dos mesmos.
- ***Energy-aware:*** Algoritmos pertencentes a essa classe são voltados para a redução do consumo de energia, o que pode ser feito por diferentes

abordagens, desde a redução do número de *hosts* ativos até mesmo a alocação baseada em estimativa de gastos de VMs e *hosts*.

- ***Network-aware:*** Algoritmos desta classe buscam melhorar algum aspecto da rede de computadores relativa à infraestrutura do sistema, como redução de latência e melhoria de escalabilidade.

- ***SLA-aware:*** Algoritmos desta classe orientam sua decisão de acordo com informações do Acordo de Nível de Serviço entre cliente e provedor. Nesta abordagem o algoritmo busca arranjar o sistema de forma a reduzir a probabilidade de haver algum tipo de violação dos termos acordados, como por exemplo falta de memória, baixa capacidade de banda, etc.

3.4 Considerações finais

Quanto ao estado da arte, é possível observar que existe uma variedade de soluções diferentes para o problema de alocação de VMs, como algoritmos baseados em soluções aproximadas para o problema da mochila, algoritmos bioinspirados (algoritmo genéticos e afins), aplicação de modelos estocásticos, etc. Também é possível observar uma forte tendência em considerar a economia de energia como um objetivo importante no campo de pesquisa de escalonamento de VMs.

A Tabela 1 contém um resumo das principais propriedades encontradas nos algoritmos estudados, identificando a abordagem e método usado para a solução do problema, bem como a classe de objetivo em que se enquadram.

Tabela 1 – Propriedades dos algoritmos estudados

Algoritmo	Abordagem	Classe	Método
MFR	Heurística	*SLA-Aware* e *Energy-Aware*	Baseado em FFD
Energy Efficient VM Placement	Heurística	*Energy-Aware*	Programação dinâmica e busca local
BGM-BLA	Metaheurística	*Energy-Aware*	Algoritmo Genético
MGGA	Metaheurística	*Energy-Aware*	Algoritmo Genético
GABA	Metaheurística	*Energy-Aware*	Algoritmo Genético
Cluster-And-Cut	Heurística	*Network-Aware*	Baseada em *MinCut*
Energy-Aware Joint VM Placement	Heurística	*Energy-Aware*	Baseado em FF e *MinCut*
PABFD	Heurística	*Energy-Aware*	Baseado em BFD
MinCut Hierarchical Clustering	Heurística	*Energy-Aware*	Baseado em BF e *MinCut*
NICE	Heurística	*Energy-Aware*	Baseado em FFD e *Kernighan-Lin*
OptSched	Heurística	*Energy-Aware*	Baseado em BF
EQVMP	Heurística	*Energy-Aware*	Baseado em BF e *MinCut*
Enhanced First-Fit Decreasing	Heurística	*Energy-Aware*	Baseado em FFD
Towards Profitable VM Placement	Heurística	*SLA-Aware*	Baseado em FF
Escalonamento com monitoramento de carga	Heurística	*Load-Aware*	Baseado em WF

4 Algoritmo KLVMP

O objetivo deste trabalho é a proposição de um algoritmo para alocação de máquinas virtuais que tem dois objetivos: minimizar o número de máquinas físicas ativas e minimizar o custo de comunicação, atendendo restrições de SLA. Neste capítulo será inicialmente formalizado o problema de alocação, para na sequência serem apresentadas a proposta de um algoritmo que atenda aos objetivos indicados e a sua implementação de seu protótipo.

4.1 Definição do problema

A justificativa para os objetivos definidos para o problema de alocação pode ser dada da seguinte forma:

- **Minimizar o número de máquinas ativas:** em um sistema de computação em nuvem, um *host* deve estar ativo caso hospede alguma VM, mesmo que esta esteja ociosa (salvo excessões, como em casos de VMs temporárias, nas quais o cliente está ciente que esta pode ser retirada a qualquer momento). Assim, é mais econômico para o provedor dos recursos agrupar o maior número possível de VMs em um menor número possível de *hosts*, podendo então desativar *hosts* que estiverem ociosos.
- **Minimizar o custo de rede:** muitas vezes processos executando em VMs distintas necessitam trocar informações, gerando um custo de comunicação que pode elevar o custo de processamento se a entrega dos processos for mais demorada. Assim, é interessante que pares de VMs com alto fluxo de comunicação sejam alocados no mesmo *hosts*, ou o mais próximo possível em relação a topologia da rede, de forma a minimizar o uso de enlaces e o *overhead* de comunicação.

Formalmente, o problema de alocação pode ser descrito da seguinte maneira: Um sistema possui um conjunto de *hosts* $[pm_1, ..., pm_p]$ e um conjunto de máquinas virtuais $[vm_1, ..., vm_q]$, das quais, uma parcela trocam dados entre si.

Tabela 2 – Descrição de símbolos utilizados na formalização do problema

Símbolo	Descrição
vm_i	Máquina Virtual i
pm_i	*Host* i (*physical machine*)
w_{ik}	Quantidade do recurso k disponível no *host* i
$S(v,w)$	Número de saltos entre v e w
$F(v,w)$	Fluxo entre v e w
$R(w)$	Função que retorna 1 se w possuir VMs e 0 caso contrário
p	Número total de *hosts*
q	Número total de VMs
d	Número de dimensões de cada recurso
T	Função objetivo Custo de rede
V	Função objetivo Número de máquinas ativas

Devemos rearranjar estas VMs atribuindo cada vm_i a um pm_j de forma a reduzir o número de *hosts* ativos e o custo de comunicação na rede.

Com relação ao tráfego na rede, podemos calcular seu custo realizando a somatória do fluxo de dados entre cada par de VMs multiplicado pelo número de saltos entre as mesmas. Assim, reduzir o custo de rede equivale a reduzir o número de saltos entre VMs, priorizando pares que possuam maior fluxo de dados entre si.

Para melhor auxiliar no entendimento do problema, na Tabela 2 são descritos os símbolos utilizados nas expressões apresentadas a seguir no texto.

Neste problema de otimização combinatória tem-se duas funções objetivo, T e V, que devem ser minimizadas segundo a formulação dada pela Equação 4.1.

$$\begin{aligned} &\min T = \sum_{i=1}^{q-1} \Big(\sum_{j=i+1}^{q} F(vm_i, vmj) * S(vm_i, vmj) \Big) \\ &\min V = \sum_{i=1}^{p} R(pm_i) \\ &\text{Sujeito a: } \forall i \in [1..p] ek \in [1..d] : w_{ik} \geq 0 \end{aligned} \tag{4.1}$$

Como pode ser notado na Equação 4.1, a solução para o problema buscar minimizar funções objetivos T, relativa ao custo de rede, e V, relativa ao número

de *hosts* ativos, respeitando as restrições relativas à capacidade de *hosts* em hospedarem uma certa quantidade de VMs, de forma que um *host* e uma VM têm d dimensões cada. Cada dimensão refere-se a uma quantidade de recursos considerada, por exemplo, se considerarmos memória e poder computacional, teremos $d = 2$. Dessa forma, para qualquer *host* do sistema, o custo de alocação de um conjunto de VMs hospedados por este não deve violar a capacidade de nenhuma de suas d dimensões.

4.1.1 Justificativa da formulação multiobjetivo

A formulação multiobjetivo apresentada pela Equação 4.1 pode ser justificada por trabalhos anteriores, que mostram a necessidade em se otimizar também o custo de comunicação. Dentre esses trabalhos pode-se citar o de MENG, PAPPAS e ZHANG (2010), envolvendo cerca de dezessete mil máquinas virtuais hospedadas em um *datacenter* da IBM Global Services. Esse conjunto de máquinas processavam cargas reais de diversos clientes da empresa, e a partir deles foi possível constatar as seguintes características a respeito do padrão de comunicação entre as máquinas virtuais:

- **Distribuição desigual:** Os resultados mostraram que o tráfego se distribui de maneira desigual entre os pares de VMs. Enquanto 80% das VMs apresentavam tráfego em média menor que 800 KBytes/min, 4% delas apresentavam uma taxa dez vezes maior que estas.
- **Estabilidade a longo prazo:** Computando o tráfego em grandes intervalos de tempo, foi possível constatar que a maioria das VMs (82%) apresentavam proporcionalidade entre a taxa média de comunicação e seu desvio padrão, o qual era cerca de duas vezes a média.

Os resultados deste estudo indicaram que é possível identificar um padrão de comunicação entre VMs para aplicar um método de otimização do custo de rede e obter um resultado positivo. Ainda segundo MENG, PAPPAS e ZHANG (2010), é possível apontar dois benefícios em otimizar o custo de rede por meio da alocação de VMs:

- **Redução de latência:** métodos de alocação de VMs voltados para otimização do custo de rede buscam alocar VMs que apresentam alta taxa de comunicação o mais próximo possível com relação a topologia da rede, de forma a minimizar o número de saltos entre estas, contribuindo assim para a redução de latência entre as mesmas.
- **Escalabilidade da rede:** a otimização do custo de rede pode contribuir para a escalabilidade da rede com a redução da carga nos dispositivos de comunicação, aumentando assim a disponibilidade dos mesmos.

Com relação à otimização do número de *hosts* ativos, sistemas de computação em nuvem possuem uma característica crucial, principalmente com relação a oferta de serviços de infraestrutura (IaaS), no qual, o servidor provisionado deve estar sempre ativo para ser acessado a qualquer momento pelo seu contratante, mesmo que a máquina virtual provisionada esteja ociosa. Isso implica que *hosts* que estejam hospedando VMs relativas a estes serviços devem estar sempre ativos. Outra característica importante é que o cliente de um serviço de computação em nuvem deve receber no mínimo a quantidade de recursos (memória, processamento, etc) que foi acordado na contratação do serviço, mas não necessariamente mais do que isso.

Assim, é vantajoso para o provedor de um sistema de computação alocar o maior número de VMs no menor número de *hosts* possível, respeitando certas condições para que haja o cumprimento mínimo do SLA. Com isso seria possível manter desligados os *hosts* que estejam ociosos até o momento em que estes sejam de fato necessários.

De um modo geral, a intenção do algoritmo é diminuir o consumo total de recursos computacionais do sistema. Esta abordagem tem como vantagem uma melhoria na escalabilidade do sistema e eventualmente uma redução do gasto de energia elétrica, mesmo não trabalhando diretamente com a quantificação de gastos de energia elétrica, mas sim apenas com o número de recursos utilizados.

4.2 Especificações do algoritmo KLVMP

O algoritmo aqui proposto foi denominado *Kernighan-Lin based heuristic for Virtual Machine Placement* (KLVMP). Essa denominação surge do fato de ser usado o algoritmo *Kernighan-Lin* (KERNIGHAN; LIN, 1970). para tratar os custos de comunicação, além de heurísticas para a minimização do número de *hosts* ativos.

Considerando que a alocação de máquinas virtuais envolve um problema de otimização combinatória de múltiplos objetivos, adotou-se a seguinte metodologia para o desenvolvimento do algoritmo:

1. Gerenciamento da lista de *hosts*, ordenando-as de forma a construir uma solução inicial boa com relação a ambos os objetivos;
2. Alocar cada *cluster* de VMs via *first-fit* e aprimorar a solução por meio da heurística desenvolvida baseada no algoritmo de *Kernighan-Lin*.

Com esta abordagem o problema de alocação de VMs fica, por um lado, reduzido a um problema de empacotamento, para seleção de *hosts* e validação em termos de capacidade, enquanto por outro lado se torna um problema um problema quadrático de alocação, quando se trata a otimização do custo de rede. Os dois problemas são reconhecidamente problemas da classe NP-difícil, sendo assim, inviável a utilização de métodos exatos para grandes instâncias. Por esse motivo optou-se pela implementação de métodos heurísticos para a obter uma solução aproximada.

4.2.1 Descrição dos dados fundamentais

Antes de introduzir o pseudocódigo do *KLVMP* é importante definir algumas das informações que serão utilizadas como dados entrada para alguns componentes. São essas:

- **Matriz de comunicação D:** é uma matriz de duas dimensões que apresenta a taxa de comunicação D_{ij} entre cada par vm_i e vm_j para VMs presentes em $[vm_0, ..., vm_q]$. Os valores de $D_{i,j}$ são determinados pelos

processos executados em cada VM e podem ser obtidos com *softwares* de análise de rede.

- **Matriz de topologia C:** é uma matriz de duas dimensões contendo o número de saltos C_{ij} entre cada par de *hosts* pm_i e pm_j presentes em $[pm_0, ..., pm_p]$. Os valores de C_{ij} podem ser determinados por meio de funções que caracterizam cada topologia, como por exemplo para a topologia *fat-tree*, em que o número de saltos é dado pela Equação 4.2.

$$S_{ij} = \begin{cases} 0, & \text{se } i = j \\ 1, & \text{se } \lfloor \frac{2i}{k} \rfloor = \lfloor \frac{2j}{k} \rfloor \\ 3, & \text{se } \lfloor \frac{2i}{k} \rfloor \neq \lfloor \frac{2j}{k} \rfloor \wedge \lfloor \frac{4i}{k^2} \rfloor = \lfloor \frac{4j}{k^2} \rfloor \\ 5, & \text{se } \lfloor \frac{4i}{k^2} \rfloor \neq \lfloor \frac{4j}{k^2} \rfloor \end{cases} \tag{4.2}$$

As matrizes de comunicação e de topologia serão utilizadas na determinação da alocação das VMs no algoritmo KLVMP. Essa determinação usa componentes para definição do grupo inicial de *hosts* (seleção de domínio inicial), definição de *clusters* de máquinas virtuais (direcionamento de solução), emparelhamento entre *hosts* e *clusters* (construção inicial) e de aprimoramento da solução. A seguir serão descritos esses componentes, incluindo seu pseudocódigo.

4.2.2 Método construtivo

Nesta etapa é realizada a construção de uma solução inicial baseada no algoritmo *First-fit*. No algoritmo 1 é apresentado este método.

Como apresentado no Algoritmo 1, a lista de *hosts* deve ser ordenada no passo da linha 5. Essa ordenação pode ocorrer de formas diferentes, de acordo com o objetivo buscado com maior prioridade. Há duas maneiras diferentes de se realizar esta ordenação, que são:

- **Ordenação por saltos:** neste tipo de ordenação, escolhe-se um *host* como primeiro da lista e, a partir dele, dois *hosts* consecutivos devem apresentar o menor número de saltos possível entre si. Para topologias regulares,

Algoritmo 1: MÉTODO CONSTRUTIVO

Entrada: V (List de VMs), D (Malha de comunicação entre VMs), P (Lista de *hosts*), C (Matriz de custo entre *hosts*)

1 $A \leftarrow \emptyset$ {*hosts* inativos}
2 $S \leftarrow \emptyset$ {*hosts* ativos}
3 $R \leftarrow \emptyset$ {*hosts* reservas}
4 $W \leftarrow \emptyset$ {lista de *clusters* de VMs}
5 Ordenar P
6 $n \leftarrow$ número de *hosts* ativos de P aplicando *first-fit* de V em P
7 Adicionar $[P_1 ... P_n]$ em T
8 Adicionar $[P_{n+1} ... P_{|P|}]$ em R
9 Separa V em *clusters* e armazena cada um em G
10 Ordena W em ordem decrescente em relação a soma de arestas de cada *cluster*
11 **para** $i = 1 \rightarrow |G|$ **faça**
12 Alocar G_i em P via *first-fit*
13 $L \leftarrow$ *hosts* que contém VMs de W_i
14 **KLQAP(W_i, D, L, C)**
15 Adicionar todos *hosts* de A e R que contém pelo menos uma VM em S e removê-los da sua lista fonte (A ou R)
16 Esvaziar P
17 Apendar A, S e R em P nesta ordem
18 **fim**

como a *fat-tree*, uma configuração ideal pode ser encontrada por meio de uma busca gulosa.

- **Ordenação por capacidade:** neste procedimento a lista de *hosts* é ordenada de forma decrescente em relação a capacidade de um cada um deles.

A ordenação por saltos pode levar a um resultado melhor para a rede utilizando um maior número de *hosts*, enquanto a ordenação por capacidade pode reduzir o número de *hosts* ativos aumentando a utilização de rede.

4.2.3 Aprimoramento de solução

A otimização final do custo de rede em um *cluster* de VMs é feito por uma variação do algoritmo *Kernighan-Lin*, que será denominada *Kernighan-Lin for Quadratic Assignment Problem* (KLQAP).

O algoritmo *Kernighan-Lin* é um algoritmo heurístico que trata o problema de particionamento de grafos (KERNIGHAN; LIN, 1970). O algoritmo busca particionar um grafo em dois grupos diferentes de forma a maximizar a soma de arestas internas (arestas que ligam vértices do mesmo grupo) de cada grupo, o que equivale a minimizar a soma de arestas externas (arestas que ligam vértices do grupo a vértices de grupos diferentes). Na Figura 3 é apresentado um fluxograma a respeito de seu funcionamento.

Seja um grafo W com n vértices e duas partições A e B, as quais receberão $n/2$ vértices cada. Inicialmente define-se uma solução qualquer, distribuindo arbitrariamente $n/2$ vértices de W para cada partição. O algoritmo então troca pares de A e B de forma a minimizar a soma de arestas externas.

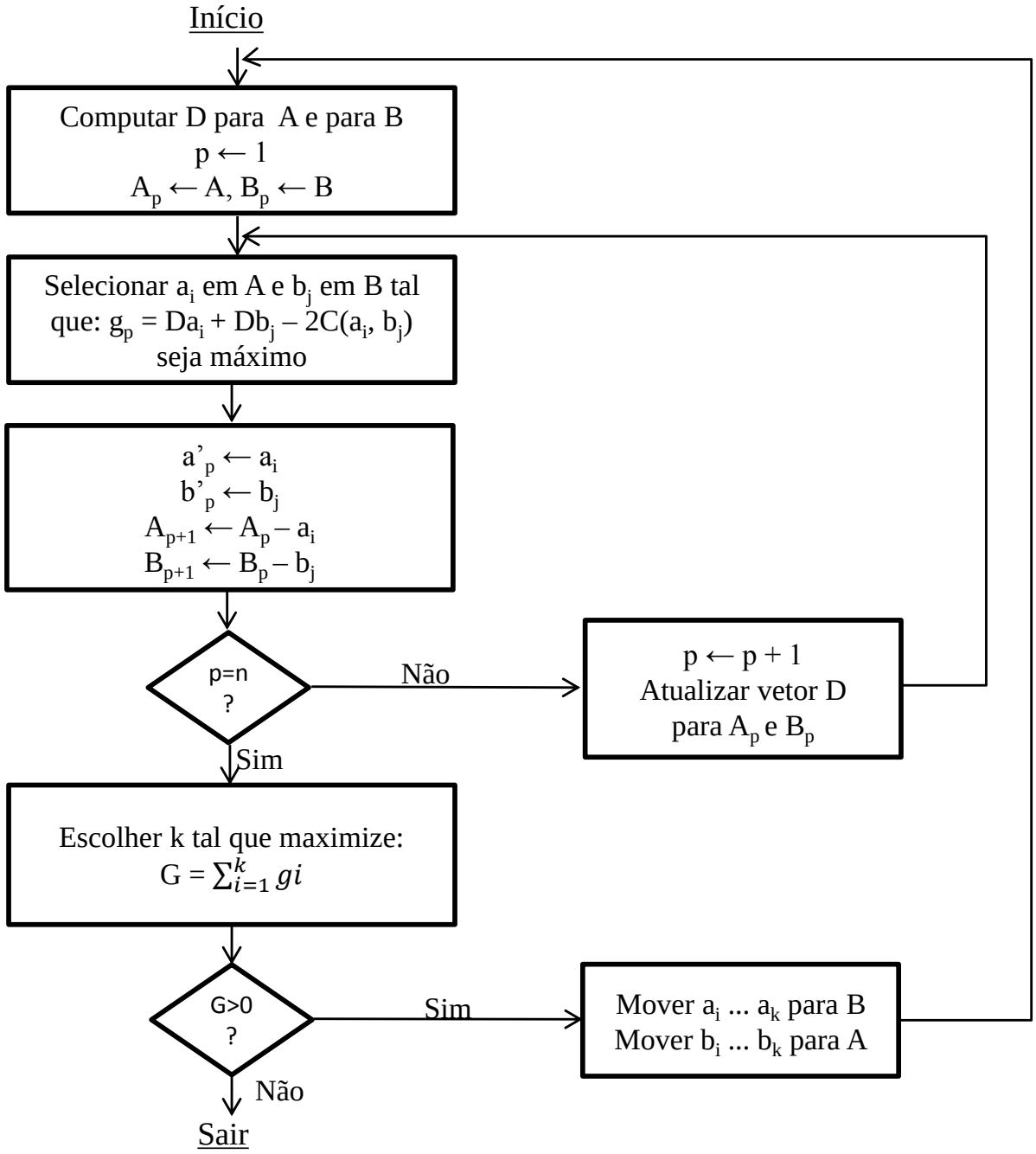
No fluxograma apresentado na Figura 3, tem-se que a variável D refere-se a um vetor com n posições, tal que o valor de cada posição é dado por $D_v = E_v - I_v$ para um vértice v pertencente ao grafo W, em que E_v representa a soma das arestas externas de v e I_v a soma das arestas internas de v. A função $C(v_a, v_b)$ retorna o peso da aresta entre v_a e v_b.

O algoritmo ainda pode ser otimizado dividindo o vetor D em duas instâncias: D_a e D_b, respectivamente para os vértices em A e B. Assim, ordenando D_a e D_b, é possível tornar o processo de busca do maior ganho mais rápido, de forma que o laço responsável pela busca do maior g_p possa ser interrompido no momento em que o valor encontrado for menor que seu antecessor.

Apesar de sua eficiência, a solução original do algoritmo não resolve o problema de alocação apresentado de maneira eficiente. Para que possa fazer isso é necessário implementar algumas modificações no algoritmo original, atendendo algumas demandas do problema. As modificações realizadas foram as seguintes:

- **Multiparticionamento:** o algoritmo original considera grafos de entrada em apenas duas partições, assim, para particionamentos em mais de 2 vias,

Figura 3 – Fluxograma de funcionamento do algoritmo *Kernighan-Lin* (KERNIGHAN; LIN, 1970)



(Traduzido pelo autor)

reduzia o problema de particionamento em k-vias para um problema de 2-vias. Isso pode prender facilmente a solução a ótimos locais. Portanto, a abordagem aqui utilizada modifica esse processo de forma a particionar o grafo de entrada em k-vias, em que cada partição representa um *host* do sistema e cada vértice uma VM.

- **Peso entre partições:** nesta etapa o problema de alocação de VMs deve ser reduzido ao problema quadrático de alocação. Assim, é necessário haver pesos entre partições. Como cada partição será mapeada para um determinado *host*, o peso entre um par de partições será o número de saltos entre os *hosts* que estas representam.

- **Validação de troca:** Um troca deve respeitar as restrições de capacidade das partições envolvidas. Por exemplo, supondo uma possível troca entre vértices a e b, pertencendo respectivamente às partições p e q, a troca só pode ser realizada se b couber em $p-a$ e a couber em $q-b$.

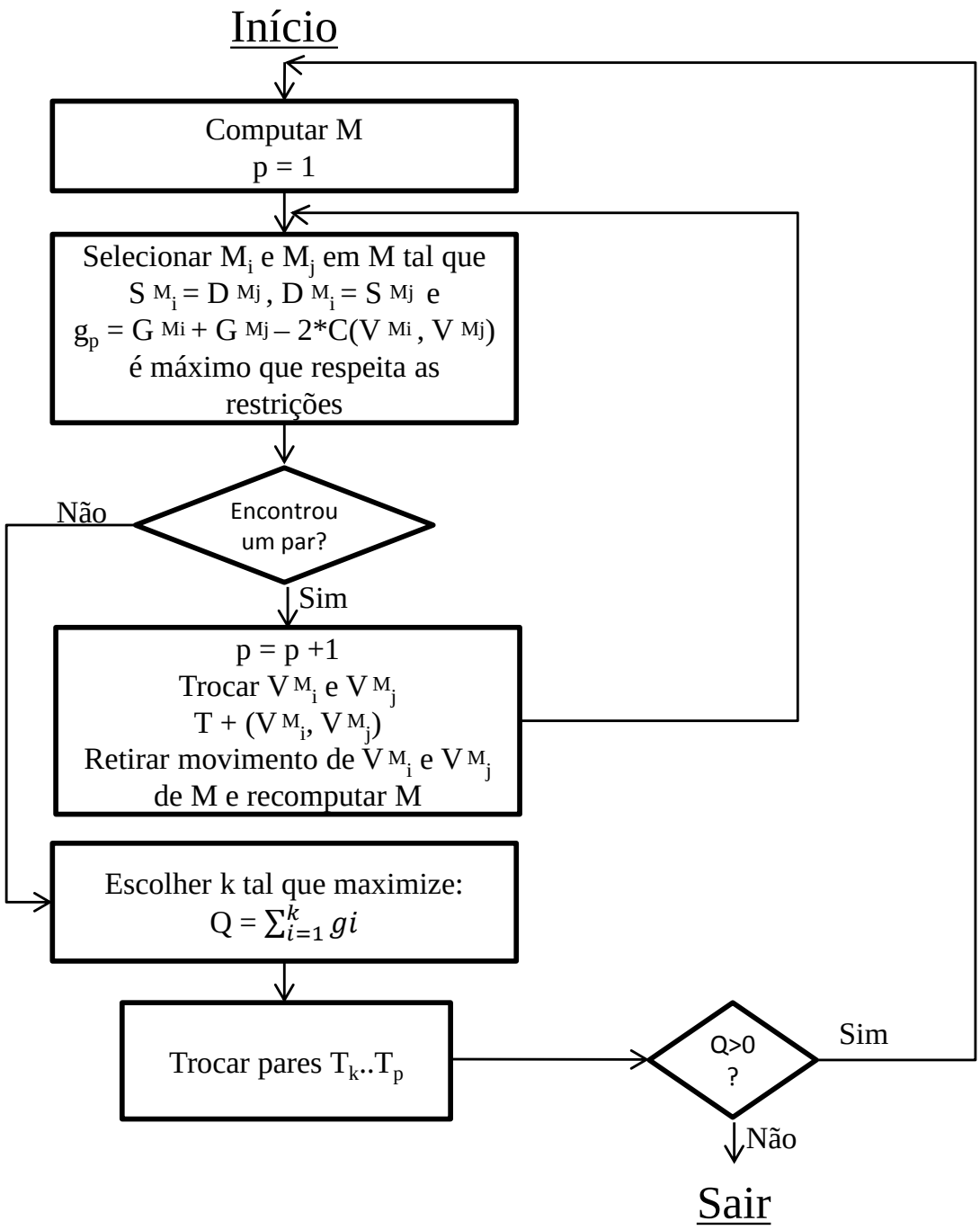
Na Figura 4 é ilustrado um fluxograma a respeito do método desenvolvido para atender essas demandas.

A variável T apresentada no fluxograma é uma lista em que cada item possui um par de vértices que foram trocados de partições. No algoritmo original a troca de pares é apenas especulada durante o processo de busca por pares, de forma que no final, são selecionados apenas os pares convenientes e assim realizada a troca. Nesta modificação as trocas já são efetuadas, de forma que ao terminar o processo de busca, os pares trocados que não propiciam o melhor ganho são então destrocados. Isso acontece pois, a cada troca, o estado do sistema é alterado, podendo modificar também o valor objetivo (soma de arestas externas). Assim, no caso do particionamento em mais de duas vias não é possível apenas especular uma troca, é necessário realizar a troca de fato para mudar o estado do sistema.

A variável M apresentada no fluxograma representa uma lista de movimentações. Uma movimentação é um conjunto de quatro diferentes elementos:

- **Vértice (V):** vértice que será submetido a uma movimentação.

Figura 4 – Fluxograma de funcionamento da heurística de trocas do algoritmo *KLQAP*



- **Origem (S):** partição de origem do vértice V que será movido.
- **Destino (D):** partição para a qual o vértice V será movido.
- **Ganho (G):** diferença entre o custo de rede total no estado atual do sistema e o custo de rede total após a movimentação do vértice V de S para D.

Para uma melhor performance do algoritmo, o ganho G de cada movimentação deve ser calculado individualmente para cada vértice. A equação 4.3 apresenta o cálculo de G, considerando que o conjunto $[v_1,...,v_k]$ representa os vértices adjacentes a V e $P[v_i]$ é a partição que o vértice v_i está alocado, e $d(r,q)$ o peso da aresta que liga os elementos r e q.

$$G = \sum_{i=1}^{k} (d(V,v_i) * d(S,P[v_i])) - \sum_{i=1}^{n} (d(V,v_i) * d(D,P[v_i])) \tag{4.3}$$

Inicialmente, para cada vértice em $[v_1,...,v_n]$ e partição em $[p_1,...,p_m]$ deve-se calcular a movimentação M_{ij} para todo $1 \leq i \leq n$ e $1 \leq j \leq m$, tal que p_j é diferente da partição de origem de v_i e adicionar M_{ij} na lista M.

Ao encontrar um par de ganho máximo e efetuar a troca, todo item em M, cujo vértice a ser movido seja um dos vértices do par escolhido, deve ser removido da lista e todo item cujo vértice tenha adjacência com um dos pares deve ser recalculado segundo a Equação 4.3.

É possível ainda melhorar o desempenho do algoritmo por meio de duas etapas consecutivas:

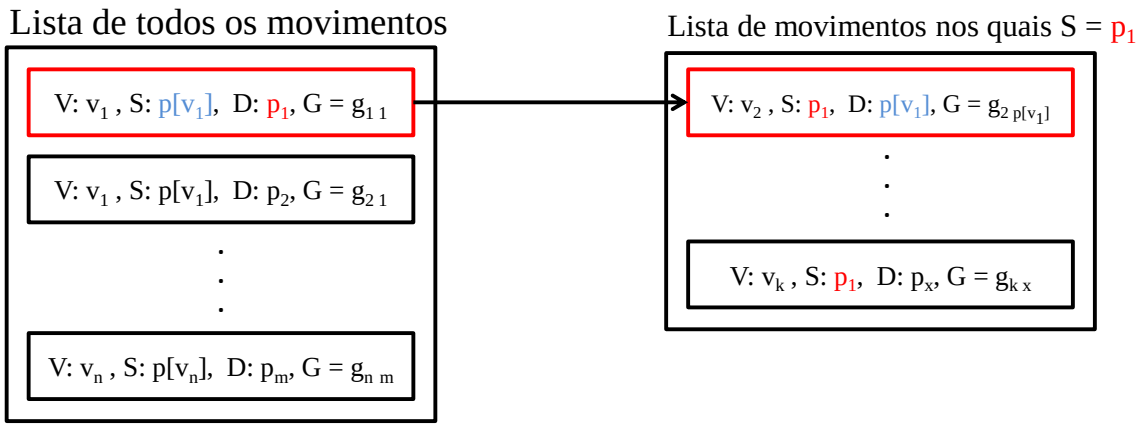
1. Alocando cada item de M em outras sub-listas de forma que apenas os itens de uma mesma partição de origem estejam em uma mesma lista. Dessa forma evita-se comparações desnecessárias, pois um par de movimentação só pode ser combinado caso a partição de origem de um vértice seja a mesma de destino do outro e vice-versa. Na Figura 5 é ilustrado este esquema, como exemplo em destaque uma possível combinação de movimentos envolvendo os vértices v_1 e v_2.

2. Ordenando todas as listas, tanto a lista geral, quanto as listas de movimentos por partição, em ordem decrescente em relação ao ganho antes de se iniciar o processo de busca pelo maior ganho. Assim, ao selecionar um item da lista geral e buscar uma combinação na lista de movimentos específicos de sua partição de destino, é possível aplicar as seguintes condições:

   a) Se uma combinação de movimentos for válida e o ganho g_p for maior que a combinação testada anteriormente, selecionar o par de movimentos e continuar a busca.

   b) Se uma combinação de movimentos for válida e o ganho g_p não for maior que o antecessor, encerrar buscar na lista movimentos específicos por partição e selecionar o próximo item da lista geral.

Figura 5 – Esquema de otimização de busca por movimentos.



4.3 Análise de complexidade

De uma forma resumida, o algoritmo KLVMP utiliza as seguintes etapas para a resolução do problema:

- **Busca em largura:** Utilizado para separar VMs em diferentes *clusters*, este procedimento possui complexidade de tempo $O(n^2)$ utilizando matriz de adjacência.

- ***First-fit:*** A construção de solução inicial é realiza por meio do algoritmo *first-fit*, que possui complexidade de tempo $O(n^2)$.

- ***KLQAP:*** Inicialmente, para cada VM é necessário computar o custo de rede gerado por esta caso seja alocada em outra partição. A computação de custo de rede individual apresenta complexidade de tempo $O(n)$, enquanto esta operação deve ser realizada para todas as VMs em todas as partições, exceto a sua própria, gerando assim uma complexidade total de $O(n^3)$. O processo de recomputação da lista de movimentos consiste em remover movimentos relativos a VMs já selecionadas para troca e recomputar do ganho de movimento relativos as VMs adjacentes ao par trocado, tendo assim também complexidade $O(n^3)$, repetida para todo par selecionado, totalizando complexidade $O(n^4)$. Os demais procedimentos, como cálculo de ganho máximo e troca de pares, possuem complexidade de tempo $O(n)$. Finalmente, considerando todos estes procedimentos tem-se complexidade de tempo $O(n^4)$.

Todos os procedimentos descritos são realizados sequencialmente, de forma que a complexidade total do algoritmo seja a soma de cada etapa. Portanto, a complexidade de tempo total equivale a $O(n^2+n^2+n^4) = O(n^4)$.

O procedimento mais custoso em termos de complexidade de tempo está na etapa de aprimoramento do custo de rede. Algoritmos do estado que tem como função objetivo minimização desse custo, como *Cluster-And-Cut* e *EAJOINT*, apresentados na Tabela 1 também apresentam complexidade de tempo $O(n^4)$.

4.4 Uso de memória

O algoritmo proposto tem como entrada uma lista de VMs, uma lista de *hosts*, uma matriz que representa a comunicação entre VMs e outra que representa o custo de comunicação entre *hosts*.

Com relação a programação, o uso de matrizes estáticas tem como vantagem acesso imediato ao dado requirido. No entanto, matrizes muito grandes podem acarretar em uso excessivo de memória. Esse problema pode ser parcialmente resolvido substituindo a matriz de custos entre *hosts* por uma função que

calcule o valor desejado no momento que este for necessário, dispensando assim a necessidade de se armazenar uma grande quantidade de dados na memória.

Um exemplo dessa redução, se a topologia do *datacenter* em questão for uma *fat-tree*, é possível implementar a Equação (4.2) por meio de uma função para retornar o custo entre *hosts*.

4.5 Considerações finais

Neste capítulo foi apresentado o problema de alocação de máquinas virtuais juntamente com a solução desenvolvida, que foi denominada KLMVP. O algoritmo proposto implementa um método construtivo baseado em heurísticas para tratamento do problema de empacotamento, que inicialmente determina o número de *hosts* utilizados e um método heurístico para tratar o problema quadrático de alocação baseada no algoritmo de *Kernighan-Lin*, que aprimora a solução buscando reduzir o custo de rede.

5 Avalição do algoritmo KLVMP

Neste capítulo serão apresentados testes a respeito do algoritmo KLVMP. Estes testes tem como objetivo, além de avaliar sua eficiência, realizar um estudo de como a capacidade disponível de cada *host* e sua respectiva posição na rede pode afetar o balanço entre otimização de rede e *hosts* ativos.

Para tanto, os testes são conduzidos por dois diferentes cenários com relação ao posicionamento dos *hosts* na rede. No pior cenário, em que o posicionamento se dá de maneira aleatória, existe grande probabilidade de haver alto custo entre *hosts* com grande capacidade disponível. No melhor cenário, em uma rede mais estruturada, a probabilidade de que os *hosts* de maior capacidade estejam mais próximos entre si é maior, levando a um menor custo da rede.

Para um efeito comparativo, os mesmos testes aplicados ao *KLVMP* são aplicados a dois outros algoritmos, um do estado da arte, *EAJOINT* (HUANG et al., 2012), apresentado na revisão bibliográfica deste texto e outra heurística, denominada neste texto como *Network-Aware First-Fit (NAFF)*, que nada mais é que o algoritmo *First-Fit* tendo sua entrada de VMs ordenada em *clusters*, separando as VMs em *clusters* de acordo com sua malha de comunicação.

A escolha desses algoritmos como critério de comparação se deve ao fato de que o *EAJOINT* possui as mesmas funções objetivo (minimização de *hosts* ativos e custo de rede) e o *NAFF*, por utilizar a heurística *first-fit* e ordenação por clusters é capaz de prover um resultado melhor que uma solução de alocação aleatória ou baseada em *Round-Robin*. Além disso, os testes serão realizados tendo como entrada modelos de até milhares de VMs e *hosts*, sendo assim, impraticável a busca por soluções ótimas.

A seguir são apresentados maiores detalhes sobre os algoritmos escolhidos, conflitos de objetivos, modelos de testes e implementação.

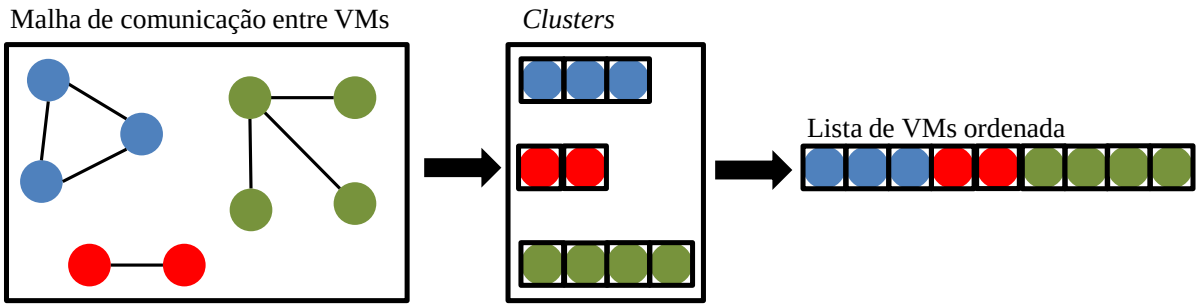
5.1 Algoritmos de comparação

Nesta seção são apresentados maiores detalhes sobre os algoritmos escolhidos para comparação com o proposto neste trabalho. Ambos algoritmos implementam métodos heurísticos para busca de uma solução aproximada, uma vez que a busca de soluções ótimas para este problema é impraticável para instâncias maiores. Deve ser observado que a maioria dos trabalhos publicados na área comparam seus resultados contra soluções de alocação aleatória ou baseadas em *First-Fit Decreasing* e *Best-Fit*. Entretanto, estas heurísticas apresentam bons resultados quanto à redução do número de máquinas ativas, mas resultados ruins com relação ao custo de rede.

5.1.1 *Network-Aware First-Fit*

O algoritmo proposto para teste é uma variação do *First-Fit* em que a entrada de VMs é separada em *clusters*. Esses *clusters* são então justapostos em uma lista, de forma a criar um tipo de ordenação baseada na malha de comunicação de VMs. Isso gera um menor custo de rede em relação a heurísticas que se baseiem apenas em capacidade. Na Figura 6 é ilustrado esse esquema, sendo que seu pseudocódigo é apresentado no Algoritmo 2.

Figura 6 – Ordenação por justaposição de *clusters*.



Algoritmo 2: NAFF

Entrada: V (Lista de VMs não atendidas), D (Matriz de comunicação), P (Lista de *hosts*)

```
1 Ordenar V em clusters em relação a D
2 para i = 0 → |V| faça
3     para j = 0 → |P| faça
4         se vm_j em Q_i couber em P_j então
5             Alocar vm_j em P_i
6             Sair do loop
7         fim
8     fim
9 fim
```

5.1.2 *Energy-Aware Joint VM Placement*

Energy-Aware Joint VM Placement (EAJOINT) (HUANG et al., 2012), como apresentado em 3.2.3.6, usa uma abordagem heurística baseada em cortes mínimos de grafo e *bin-packing*, que tem como objetivos redução do custo de rede e redução do número de máquinas ativas. Um corte em um grafo G é definido como a escolha de um conjuntos de arestas de G, que removidas o dividem em dois subgrafos desconexos. Cada corte possível possui uma capacidade, que é a soma dos pesos de todas as arestas do conjunto. Assim, a árvore *Gomory-Hu*, utilizada no Algoritmo 4 é uma árvore geradora, tal que esta contém todos os vértices de G e cada aresta da árvore representa a capacidade de corte dos vértices ligados pela mesma. Na sequência o algoritmo é detalhado, separando-se sua análise em três componentes:

1. ***VMGrouping:*** é o componente principal do algoritmo, executado recursivamente e é responsável por chamar os componentes para corte e empacotamento de VMs. No Algoritmo 3 é apresentado esse processo. Na implementação original, a avaliação da solução (linha 5) é realizada por meio de uma inferência *fuzzy*, em que o algoritmo utiliza como critério de parada os valores das funções objetivos, caso estes estejam abaixo de um valor parametrizado. No entanto, no caso de teste aqui avaliado o critério

Algoritmo 3: VMGROUPING

Entrada: V (Lista de VMs não atendidas), D (Matriz de comunicação), P (Lista de *hosts*), C (Matriz de topologia)

```
1 G ← VMKCut(D, k) (Particiona o grafo D em k grupos [G_1, ..., G_k], em
    que k representa o número de clusters de hosts)
2 S ← SlotGrouping(G, V, P, k) (Gera o particionamento de slots em k
    grupos [S_1, ..., S_k])
3 Alocar G em S, para todo i = 1, ..., k (Associa cada grupo de VMs a cada
    grupo de slots)
4 para i = 1 → k faça
5 |   se G_i e S_i não forem satisfatórios então
6 |   |   VMGrouping(G_i, S_i, D_{G_i}, C_{G_i})
7 |   fim
8 fim
```

a ser seguido será o custo de rede. Caso o programa esteja na primeira chamada da função, não haverá parada, caso contrário, a parada acontece quando esse valor não diminuir em relação a solução antecessora. Se for decidido pela parada, então a solução atual é descartada e a antecessora é aceita.

2. ***VMKCut:*** Computa uma árvore de cortes Gomory-Hu (GOMORY; HU, 1961) e particiona o grafo que representa a comunicação entre as VMs em k grupos utilizando esta árvore. No Algoritmo 4 é apresentado o pseudocódigo para esta etapa.

3. **SlotGrouping:** Após o particionamento de VMs este procedimento mapeia cada grupo para um *slot*, que nada mais é que uma porção de recursos em um *host* capaz de acomodar uma VM. Este procedimento utiliza uma abordagem baseada em *first-fit* para determinar para qual *host* um determinado *slot* deve ser alocado, de modo que um *slot* tem exatamente as dimensões da VM para qual este servirá. Assim, em termos de programação, uma variável que represente um *slot* pode equivaler a um *host*. No Algoritmo 5 é mostrado o pseudocódigo deste componente.

Algoritmo 4: VMKCUT

Entrada: D (Matriz de comunicação), k (número de *clusters*)
Saída: G (Grupo de VMs)

```
1  n ← número de vértices em D
2  B ← n − 1 cortes de D obtidos pela árvore Gomory-Hu
3  Ordenar B em ordem crescente por capacidade de corte
4  para i = 1 → k faça
5  |  Buscar j mínimo tal que removendo B_1,...,B_j particionará D em dois
   |   subgrafos desconexos: g_1 e g_2
6  |  G_i ← VMs de g_1
7  |  D ← g_2
8  fim
9  retornar G
```

Algoritmo 5: SLOTGROUPING

Entrada: G (Lista de grupos de VMs), D (Matriz de comunicação), P (Lista de *hosts*), k (número de grupos de *slots*)
Saída: S (Lista de *slots*)

```
1  para i = 1 → k faça
2  |  S_i ← ∅
3  |  para j = 1 → |P| faça
4  |  |  para l = 1 → |G_i| faça
5  |  |  |  se G_il couber em P_j então
6  |  |  |  |  Retirar as dimensões de G_i da capacidade de P_j
7  |  |  |  |  S_i ← S_i + P_j
8  |  |  |  fim
9  |  |  fim
10 |  fim
11 fim
12 retornar S
```

5.2 Conflito entre objetivos

Em se tratando de sistemas heterogêneos, otimização de rede e número de *hosts* ativos podem ser objetivos conflitantes, de forma que a melhora de

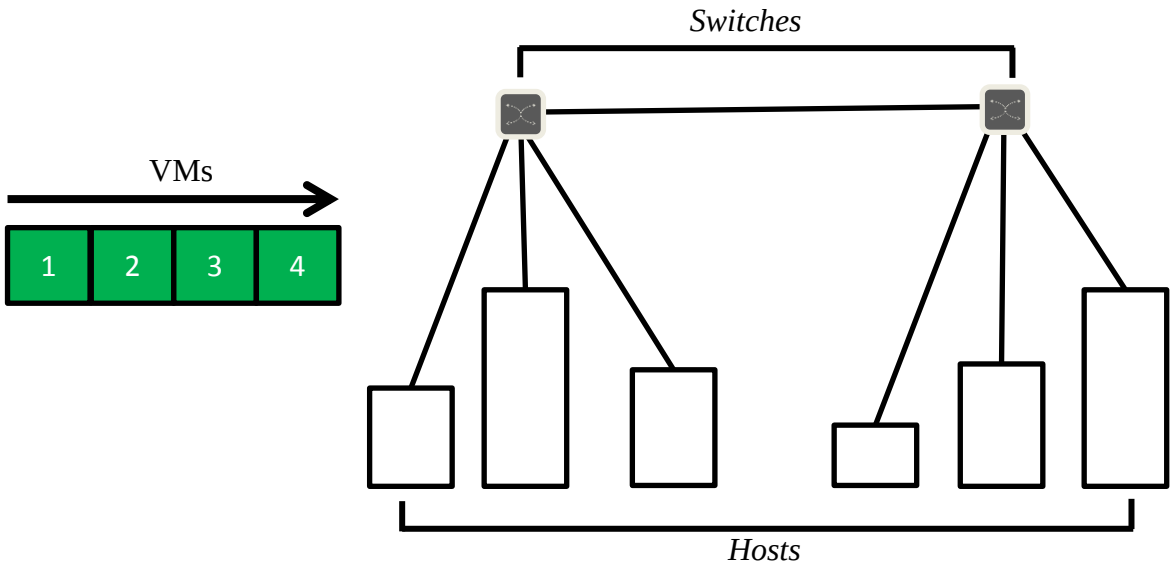
um pode causar a piora do outro. Essa situação pode ocorrer caso os *hosts* de maior capacidade no sistema estejam muito distantes um do outro com relação a topologia da rede. Neste cenário, na otimização do número de *hosts* ativos busca-se, preferencialmente, alocar as VMs naqueles de maior capacidade. Como estes estão em pontos distantes da rede, há uma tendência em se gerar um alto custo de rede, considerando que não seja possível alocar um *cluster* inteiro de VMs em apenas um *host*. Já se primeiro for buscada a otimização do custo de rede, aloca-se primeiro *hosts* que estejam o mais próximo possível, o que leva à probabilidade de selecionar máquinas com menor capacidade, aumentando o número de *hosts* ativos.

Assim, serão abordados dois casos de testes:

- **Caso de Pior Posicionamento:** o *datacenter* simulado será gerado com *hosts* posicionados aleatoriamente pela rede, aumentando a probabilidade de *hosts* de maior capacidade estarem distantes de entre si.
- **Caso de Melhor Posicionamento:** o *datacenter* será gerado tendo seus *hosts* ordenados na rede de forma que *hosts* de maior capacidade estejam o mais próximo possível.

Na Figura 7 é ilustrado um sistema organizado de acordo com o caso de pior posicionamento, no qual quatro VMs serão alocadas e entre cada VM par de VMs há um fluxo de peso 1.

Figura 7 – Sistema com pior caso de posicionamento.



Neste caso, há duas abordagens possíveis, uma para um melhor custo de rede, buscando alocar as VMs o mais próximo possível e outra pra reduzir o número de máquinas ativas, buscando alocar as VMs nos *hosts* de maior capacidade, de acordo com as funções objetivos apresentadas na Equação 4.1. Na Figura 8a é ilustrada a alocação com menor custo de rede, tendo um custo de rede 6 e 3 *hosts* ativos. Na Figura 8b é ilustrada a alocação com menor número de máquinas ativas, tendo um custo de rede 8 e 2 *hosts* ativos.

Do mesmo modo, na Figura 9 é ilustrado um sistema organizado de acordo com o caso de melhor posicionamento, com as mesmas configurações apresentadas na Figura 7, mudando apenas o posicionamento dos *hosts*.

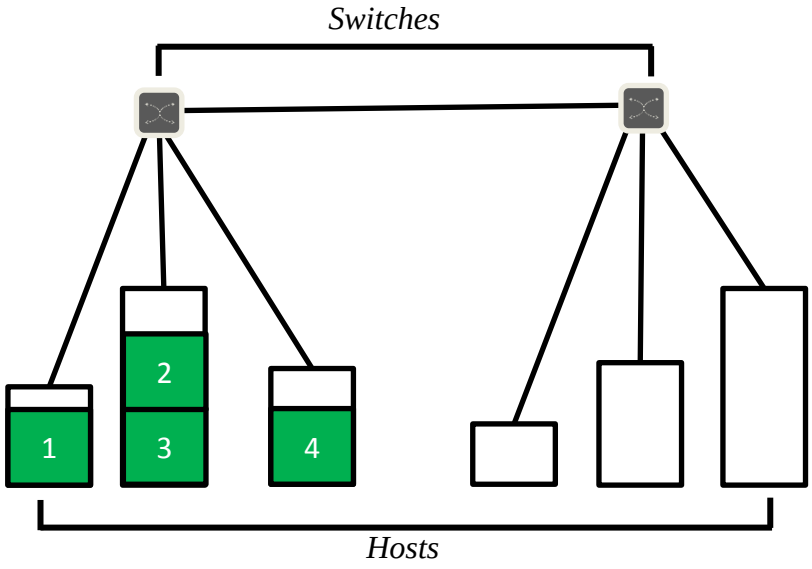
Da mesma forma, aplicando duas abordagens, uma para um menor custo de rede e outro para um menor número de *hosts* ativos, na Figura 10a é ilustrada a alocação com menor custo de rede, tendo um custo de rede 4 e 2 *hosts* ativos. Na Figura 8b é ilustrada a alocação com menor número de máquinas ativas, tendo um custo de rede 4 e 2 *hosts* ativos.

Apesar de utilizar diferentes abordagens nas Figuras 10a e 10b, a solução foi a mesma, resultado em valores objetivos iguais ou menores que o valores alcançados por cada tipo de solução do pior caso de posicionamento.

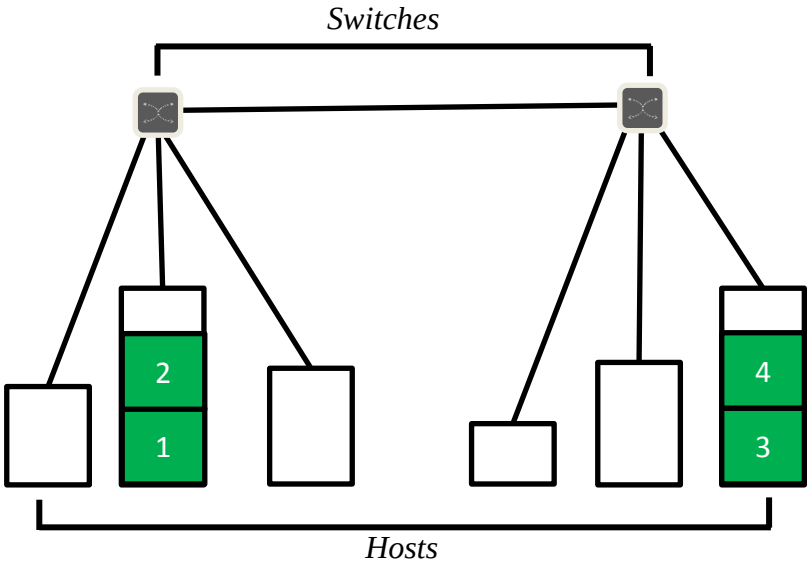
Assim, para explorar esses diferentes cenários nos casos de testes, para o caso de pior posicionamento, cada algoritmo testado possui duas variantes,

Figura 8 – Alocação no pior caso de posicionamento

(a) Menor curto de rede

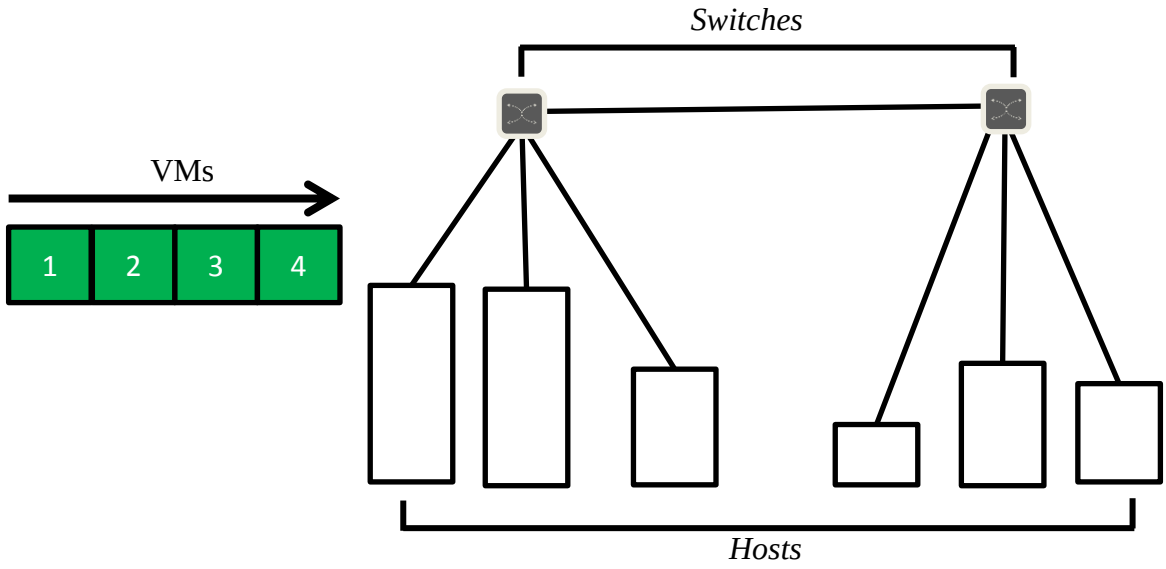


(b) Menor número de *hosts* ativos



$KLVMP^1$ e $KLVMP^2$, $EAJOINT^1$ e $EAJOINT^2$, $NAFF^1$ e $NAFF^2$. As variantes de índice 1 tem a lista de *hosts* ordenada por volume descrescente, enquanto as variantes de índice 2 tem a lista de *hosts* ordenada por custo de rede. A ordenação por volume decrescente considera volume como sendo o produtório de todas suas dimensões, que no caso são memória (GB) e poder de processamento (GHz). A ordenação por custo de rede consiste em ordenar a lista de forma que para cada par consecutivo de *hosts*, o custo de rede entre estes é mínimo.

Figura 9 – Sistema com melhor caso de posicionamento.



Como no caso de melhor posicionamento os *hosts* de maior capacidade já estão ordenados na rede, logo com custo também ordenado, qualquer uma das duas ordenações retorna o mesmo resultado, sendo assim dispensável usar mais de uma variação para um mesmo algoritmo.

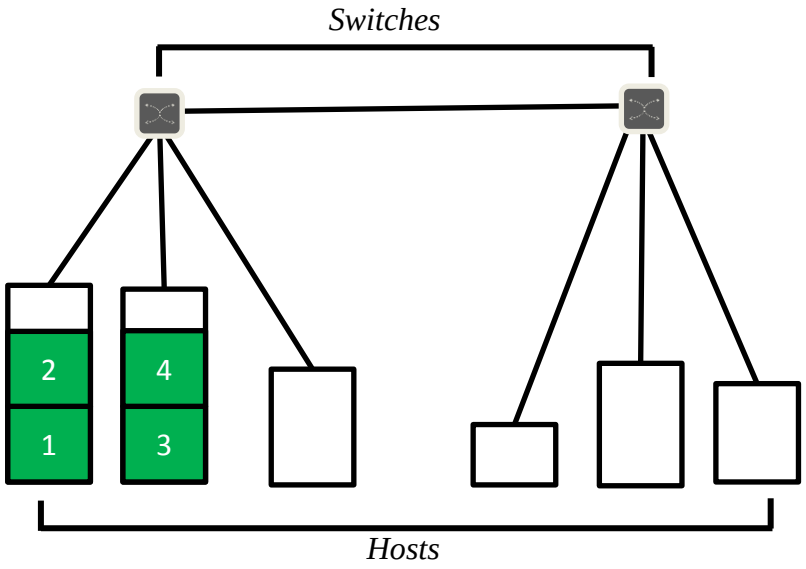
5.3 Modelo de ambiente

É muito difícil estabelecer um modelo representativo para a infraestrutura de um sistema de computação em nuvem, uma vez que, como majoritariamente se trata de um sistema comercial, informações a respeito de dimensões e quantidades de *hosts* são escassas. Os trabalhos publicados na área procuram estabelecer um modelo com base em informações técnicas, como topologias de rede comumente usadas, gerando dimensões e pesos a partir de variáveis aleatórias. Isso dificulta a comparação de resultados entre diferentes propostas.

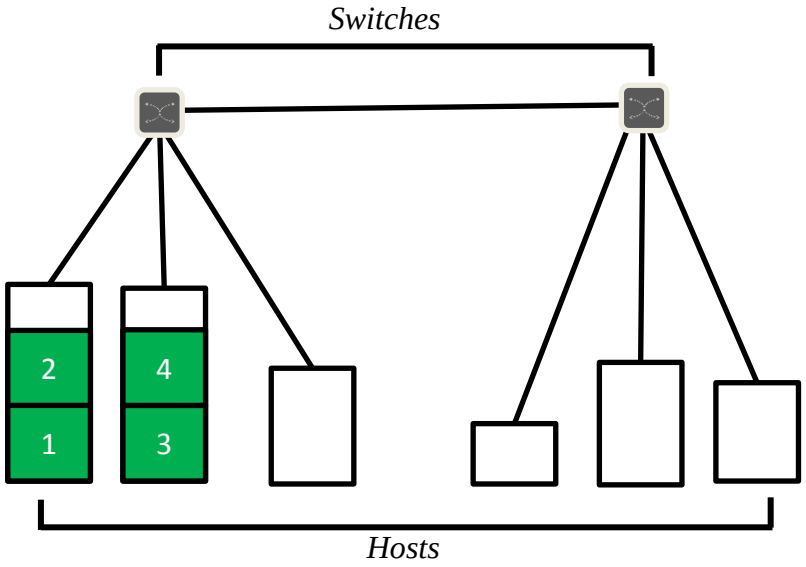
O mesmo pode ser dito a respeito da carga de trabalho. No caso do problema apresentado, seriam necessárias informações sobre quantidade de recursos utilizadas pelas VMs e a malha de comunicação entre as mesmas. No entanto, essas informações são ainda mais difíceis de se obter. Assim, a carga gerada será baseada em modelos propostos por trabalhos previamente desenvolvidos.

Figura 10 – Alocação no melhor caso de posicionamento

(a) Menor curto de rede



(b) Menor número de *hosts* ativos



O modelo estabelecido busca oferecer um espaço de busca por soluções em que se justifique a aplicação de uma abordagem mais trabalhada para a otimização do problema. Do ponto de vista de custo de rede, otimizar o problema consiste basicamente em alocar o mais próximo possível pares de VMs que apresentem alta taxa de comunicação. Assim, se os *hosts* de um sistema possuírem uma capacidade tal que consigam alocar quase um *cluster* inteiro de VMs, o problema pode ser resolvido de forma trivial. Por outro lado, se os *hosts* possuírem uma capacidade pequena, de forma que consigam alocar uma parcela

muito pequena dos *clusters*, o espaço de busca passa a ser muito restrito e sua exploração fica inviável.

Assim, no modelo proposto para testes duas dimensões de recursos são consideradas: Memória (GB), e Poder de processamento (GHz). A taxa de comunicação de VMs é denotada em Mbps. O modelo de teste é composto por um *datacenter* e a carga de trabalho que será submetida, descritos com maiores detalhes a seguir.

5.3.1 *Datacenter*

O modelo de *datacenter* representa a infraestrutura de um *datacenter* de um sistema de computação em nuvem. Seus *hosts* possuem as seguintes dimensões, todas geradas por uma distribuição uniforme:

- **Memória:** Um valor aleatório de $256 * n$ GB, tal que n é um número inteiro entre 1 e 8, totalizando um valor de até 2048 GB.
- **Poder de processamento:** O poder de processamento total é a soma do poder de cada núcleo de processamento. A característica de cada procesador foi baseada no processador Intel®Xeon®Platinum 8180, que possui 28 núcleos de processamento e 3,8 GHz de frequencia pro núcleo. Cada *host* pode ter de 1 a 4 desses processadores, tendo assim $k * 28 * 3,8$ GHz, resultando em um valor entre 106,4 GHz e 425,6 GHz.

Quanto a topologia de rede, o *datacenter* estará organizado em uma *fat-tree*. Serão consideradas duas dimensões de rede diferentes porém com mesmo diâmetro, uma com *switches* de 28 portas e outra com *switches* de 48 portas (*28-fat-tree* e *48-fat-tree*), o que significa conectar, respectivamente, uma quantidade de 5488 e 27648 *hosts*.

5.3.2 Carga de trabalho

Cargas de trabalhos neste caso são VMs, sendo que cada VM demanda uma quantidade de recursos que deve ser fornecida por algum *host*. As dimensões

dos recursos demandados são geradas por uma distribuição uniforme e estão nas seguintes faixas:

- **Memória:** Um valor aleatório entre 2 GB e 64 GB.
- **Poder de processamento:** O poder de processamento total é a soma do poder demandado para cada núcleo de processamento. Assim, cada máquina tem uma quantidade de núcleos entre 1 e 28 e um poder de processamento entre 1 GHz e 3,8 GHz por núcleo, totalizando um valor aleatório entre 1 GHz e 106,4 GHz.

Quanto à malha de comunicação, a carga de trabalho pode ser composta por um ou mais *clusters* de VMs, cada um com 2^n VMs, sendo n um número aleatório entre 0 e 7. O peso de comunicação entre pares de um mesmo *cluster* pode assumir o módulo de uma entre quatro distribuições normais: $N[5,5]$, $N[10,10]$, $N[15,15]$ e $N[20,20]$, sendo 25% de chance para cada. Os *clusters* são gerados iterativamente, até que se atinja o valor máximo de VMs. Caso este teto seja extrapolado, o último *clusters* gerado terá a quantidade de VMs truncada para este teto.

5.4 Implementação

A metodologia de avaliação dos algoritmos baseia-se em atribuir um grupo de VMs a um grupo de *hosts* (respeitando as restrições) e assim obter métricas a respeito do estado do sistema. Desse modo, não se faz necessário o uso de eventos discretos, uma vez que a simulação demandaria maior tempo de execução e seus resultados pouco corroborariam com o objetivos.

Para resolver este problema foi desenvolvido um programa em linguagem de programação C++ que recebe como entrada VMs com suas respectivas demandas de recursos, grafo de comunicação entre VMs e o *datacenter*, com sua topologia, número de *hosts* e quantidade de recursos disponível em cada um dos *hosts*.

O programa possui três classes principais:

- **PM:** Classe que representa um *host* no sistema, tendo como atributos as quantidades de recursos disponíveis.
- **VM:** Classe que representa uma VM no sistema, tendo como atributos as quantidades de recursos necessários.
- **Alocador:** Classe genérica que representa um algoritmo de alocação de VMs, de forma que algoritmos específicos são implementados como subclasse desta.

O programa possui também classes para armazenar o grafos de comunicação entre VMs e modelar a topologia do *datacenter*.

Assim, o programa busca resolver o problema de alocação com base nos dados de entrada, atribuindo objetos do tipo VM a objetos do tipo PM e obter as métricas com base no modelo 4.1.

5.5 Casos de testes

O testes foram divididos em duas categorias, como já mencionado, caso de pior posicionamento e caso de melhor posicionamento. Em cada categoria foram executados 10 casos de testes, variando a quantidade de VMs em 256, 512, 1024, 2048 e 4096 para *datacenters* em *28-fat-tree* e *48-fat-tree*, totalizando assim 20 casos de testes.

Os resultados são apresentados nas tabelas a seguir. Nelas os resultados para as funções objetivo são dispostos em uma tupla $[p, q]$, em que p representa o número de *hosts* ativos e q o custo de rede em 10^6 Mbps.

5.5.1 Caso de pior posicionamento

Nas Tabelas 3 e 4 são apresentados, respectivamente, testes em *28-fat-tree* e *48-fat-tree* para o caso de pior posicionamento.

Como já mencionado, algoritmos de índice 1 priorizam minimizar o custo de rede enquanto os de índice 2 minimizar o número de *hosts* ativos. Com relação ao custo de rede, ao compararmos algoritmos de índice 1 com índice 1 e índice 2 com índice 2, o algoritmo *KLVMP* obteve o melhor resultado em todos os casos.

Com relação ao número de *hosts* ativos, o algoritmo *KLVMP* obteve um resultado melhor que o algoritmo *EAJOINT* e igual ao *NAFF*. Entretanto, pode-se desempatar esta disputa ao analizarmos o custo de rede, que é menor para o algoritmo *KLVMP*. Examinando-se cada variação separadamente tem-se:

- Não houve diferença entre $KLVMP^1$ e $NAFF^1$ para o número de *hosts* ativos, no entanto, o custo de rede é menor para o algoritmo $KLVMP^1$ a uma taxa de média de 25% com desvio padrão de 17% para *28-fat-tree* e a uma taxa de 19% e um desvio padrão de 10% para *48-fat-tree*

- Para as variações $KLVMP^2$ e $NAFF^2$, também não houve diferença no númer de *hosts* ativos, no entanto, o custo de rede é menor para o algoritmo $KLVMP^2$ a uma taxa de média de 18% com desvio padrão de 14% para *28-fat-tree* e a uma taxa de 16% e um desvio padrão de 3% para *48-fat-tree*.

Quanto ao *EAJOINT*, o algoritmo *KLVMP* superou todos os resultados para ambos os objetivos. Considerando o número de *hosts* ativos, entre $KLVMP^1$ e $EAJOINT^1$ tem-se, em média, um valor melhor para o $KLVMP^1$ próximo a 8%, com um desvio padrão de 3% para *28-fat-tree* e um valor médio aproximado de 7% e desvio padrão de 4% para *48-fat-tree*, também melhor para o $KLVMP^1$. Quanto ao custo de rede, este é menor para o algoritmo $KLVMP^1$ a uma taxa de média de 19% com desvio padrão de 22% para *28-fat-tree* e a uma taxa de 23% e um desvio padrão de 12% para *48-fat-tree*. Enquanto que, comparando $KLVMP^2$ e $EAJOINT^2$, tem-se, em média, um valor menor para o algoritmo $KLVMP^2$ quanto aos *hosts* ativos de 3% e desvio padrão de 2% para *28-fat-tree* e uma diferença média da quantidade de *hosts* ativos de 2% e desvio padrão de 2% para *48-fat-tree*, também menor para o $KLVMP^2$, quanto ao custo de rede, este também é em média menor para o algoritmo $KLVMP^2$ a uma taxa de média

Tabela 3 – Resultados pra o caso de pior posicionamento em uma *28-fat-tree*.

Algoritmo	256 VMs	512 VMs	1024 VMs	2048 VMs	4096 VMs
$NAFF_1$	[21, 0,172]	[33, 0,247]	[79, 0,661]	[174, 1,708]	[338, 2,802]
$NAFF_2$	[10, 0,357]	[19, 0,310]	[41, 1,343]	[83, 3,059]	[163, 4,835]
$KLVMP_1$	[21, 0,089]	[33, 0,152]	[79, 0,621]	[174, 1,353]	[338, 2,464]
$KLVMP_2$	[10, 0,199]	[19, 0,258]	[41, 1,188]	[83, 2,801]	[163, 4,221]
$EAJOINT_1$	[22, 0,183]	[35, 0,229]	[89, 0,659]	[194, 1,445]	[372, 2,467]
$EAJOINT_2$	[10, 0,349]	[20, 0,268]	[43, 1,324]	[86, 2,910]	[168, 4,366]

Tabela 4 – Resultados pra o caso de pior posicionamento em uma *48-fat-tree*.

Algoritmo	256 VMs	512 VMs	1024 VMs	2048 VMs	4096 VMs
$NAFF_1$	[23, 0,107]	[50, 0,204]	[81, 0,314]	[180, 1,354]	[360, 2,328]
$NAFF_2$	[10, 0,262]	[21, 0,298]	[43, 0,712]	[82, 2,151]	[163, 4,256]
$KLVMP_1$	[23, 0,099]	[50, 0,145]	[81, 0,276]	[180, 0,921]	[360, 1,909]
$KLVMP_2$	[10, 0,225]	[21, 0,262]	[43, 0,605]	[82, 1,711]	[163, 3,475]
$EAJOINT_1$	[24, 0,102]	[55, 0,231]	[92, 0,312]	[203, 1,214]	[372, 2,467]
$EAJOINT_2$	[10, 0,261]	[22, 0,295]	[44, 0,676]	[84, 2,102]	[167, 3,961]

de 12% com desvio padrão de 17% para *28-fat-tree* e a uma taxa de 13% e um desvio padrão de 3% para *48-fat-tree*.

Nas Figuras 11a e 11b são respectivamente plotados o custo de rede e o número e *hosts* ativos em *28-fat-tree* e nas Figuras 11c e 11d, respectivamente, o custo de rede e o número e *hosts* ativos em *48-fat-tree*.

Para uma melhor análise do comportamento dos valores objetivos em relação ao tamanho da entrada e a configuração do sistema, foram elaborados gráficos de dispersão tridimensionais dos valores objetivos (custo de rede e número de máquinas ativas) em função do número de VMs dos sistema. Nas Figuras 12a e 12b são plotados gráficos de dispersão tridimensional para *28-fat-tree* e *48-fat-tree* respectivamente.

5.5.2 Caso de melhor posicionamento

Nas Tabelas 5 e 6 são apresentados, respectivamente, testes em *28-fat-tree* e *48-fat-tree* para o caso de melhor posicionamento.

Tabela 5 – Resultados pra o caso de melhor posicionamento em uma *28-fat-tree*.

Algoritmo	256 VMs	512 VMs	1024 VMs	2048 VMs	4096 VMs
NAFF	[10, 0,096]	[19, 0,158]	[41, 0,535]	[83, 1,150]	[163, 1,857]
KLVMP	[10, 0,071]	[19, 0,112]	[41, 0,471]	[83, 1,056]	[163, 1,739]
EAJOINT	[10, 0,092]	[20, 0,153]	[43, 0,485]	[86, 1,082]	[168, 1,819]

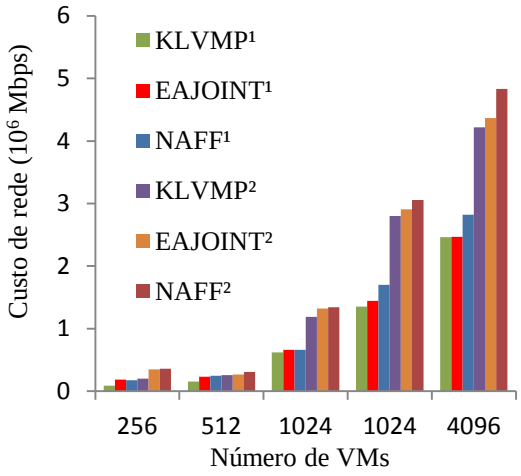
Tabela 6 – Resultados pra o caso de melhor posicionamento em uma *48-fat-tree*.

Algoritmo	256 VMs	512 VMs	1024 VMs	2048 VMs	4096 VMs
NAFF	[10, 0,095]	[21, 0,138]	[43, 0,277]	[82, 0,806]	[163, 1,916]
KLVMP	[10, 0,083]	[21, 0,118]	[43, 0,234]	[82, 0,742]	[163, 1,552]
EAJOINT	[10, 0,095]	[22, 0,136]	[44, 0,262]	[84, 0,819]	[168, 1,735]

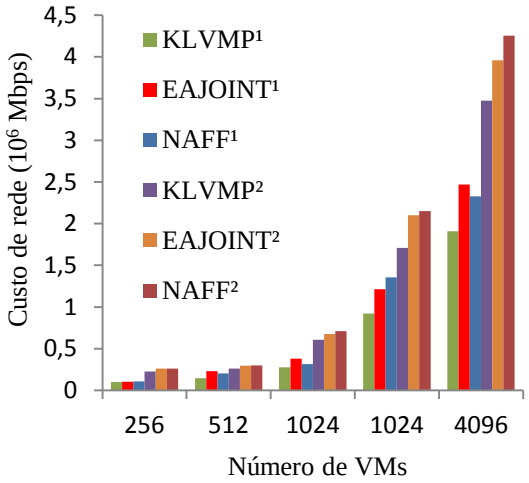
Assim como no caso do pior posicionamento, o algoritmo *KLVMP* apresenta resultados melhores em todos os casos com relação ao custo de rede e

Figura 11 – Resultados obtidos para o pior caso de posicionamento

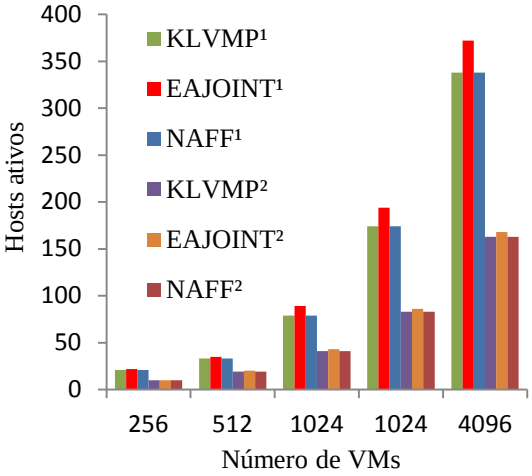
(a) Custo de rede em *28-fat-tree*



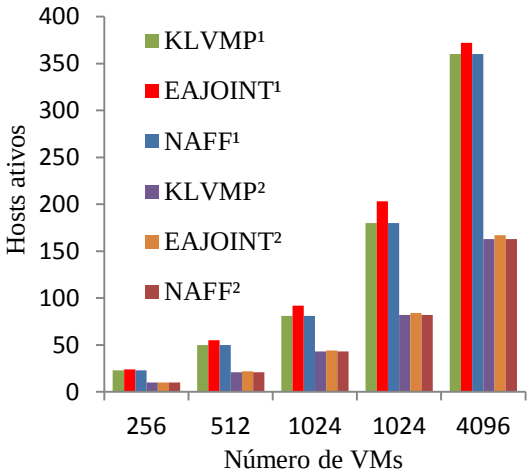
(b) Custo de rede em *48-fat-tree*



(c) Quantidade de *hosts* ativos em *28-fat-tree*



(d) Quantidade de *hosts* ativos em *48-fat-tree*

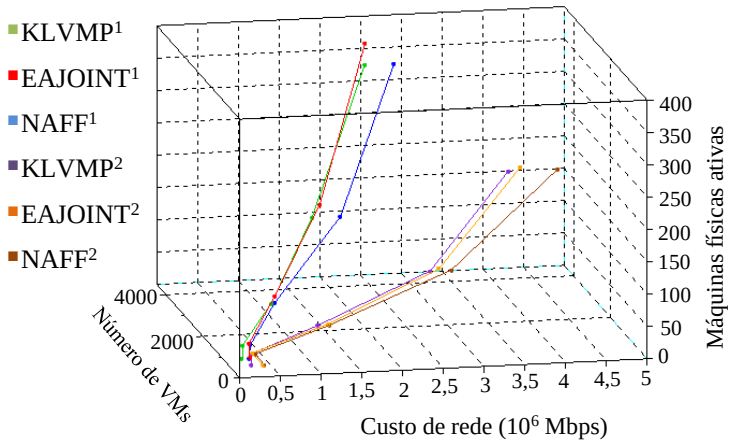


empatando com relação ao número de *hosts* ativos contra o algoritmo *NAFF*. Apesar de empatar em relação o número de *hosts* ativos, o custo de rede é menor para o algoritmo *KLVMP* a uma taxa de média de 16% com desvio padrão de 10% para *28-fat-tree* e a uma taxa de 13% e um desvio padrão de 14% para *48-fat-tree*.

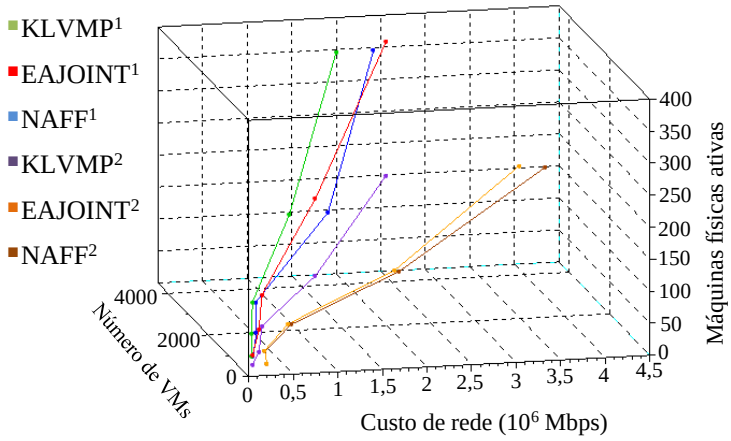
Quanto ao *EAJOINT*, ao compararmos a diferença percentual do número de *hosts* ativos entre *KLVMP* e *EAJOINT* tem-se, em média, um valor melhor para o *KLVMP* próximo a 3%, com um desvio padrão de 1% para *28-fat-tree* e um valor médio aproximado de 2% e desvio padrão de 1% para *48-fat-tree*, também melhor para o *KLVMP*. Quanto ao custo de rede, este é menor para o

Figura 12 – Dispersão tridimensional para o pior caso de posicionamento.

(a) Dispersão tridimensional para *28-fat-tree*



(b) Dispersão tridimensional para *48-fat-tree*



algoritmo *KLVMP* a uma taxa de média de 11% com desvio padrão de 11% para *28-fat-tree* e a uma taxa de 11% e um desvio padrão de 2% para *48-fat-tree*.

Nas Figuras 13a e 13c são respectivamente plotados o custo de rede e o número de *hosts* ativos em *28-fat-tree* e nas Figuras 13b e 13d, respectivamente, o custo de rede e o número de *hosts* ativos em *48-fat-tree*.

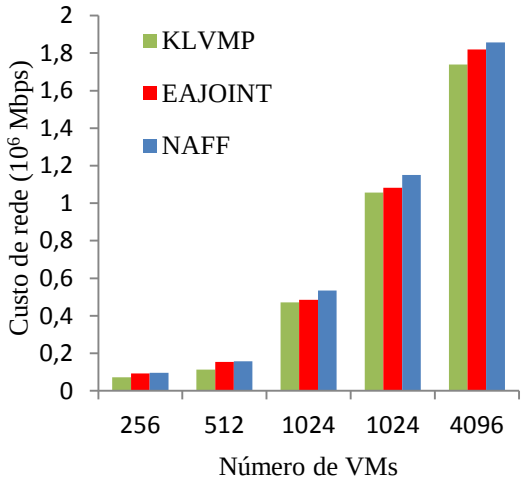
Nas Figuras 14a e 14b são plotados gráficos de dispersão tridimensional das solução obtidas para *28-fat-tree* e *48-fat-tree* respectivamente.

5.6 Análise de resultados

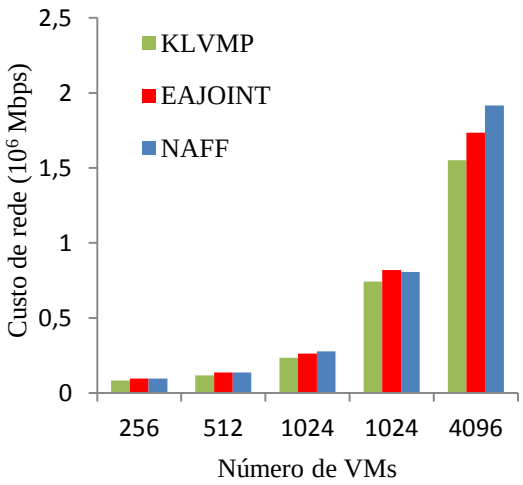
Os testes realizados buscaram comparar o desempenho do algoritmo proposto neste trabalho, *KLVMP*, com outros dois, *NAFF* e *EAJOINT*. Os critérios

Figura 13 – Gráficos de resultados obtidos para o melhor caso de posicionamento

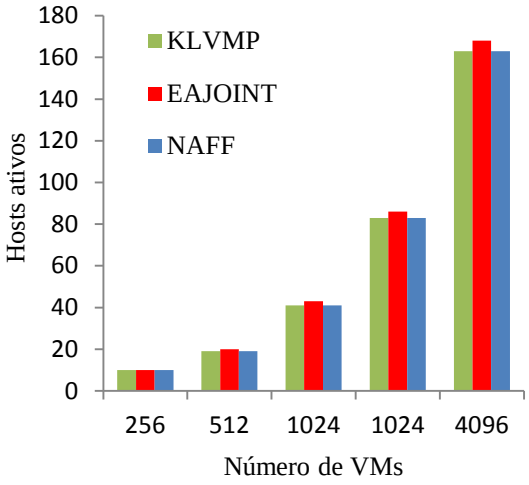
(a) Custo de rede em *28-fat-tree*



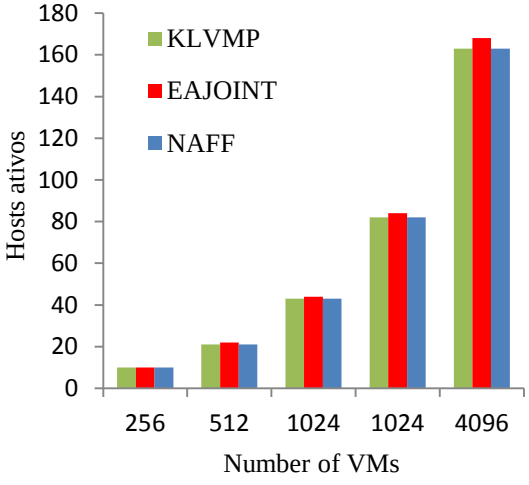
(b) Custo de rede em *48-fat-tree*



(c) Quantidade de *hosts* ativos em *28-fat-tree*



(d) Quantidade de *hosts* ativos em *48-fat-tree*



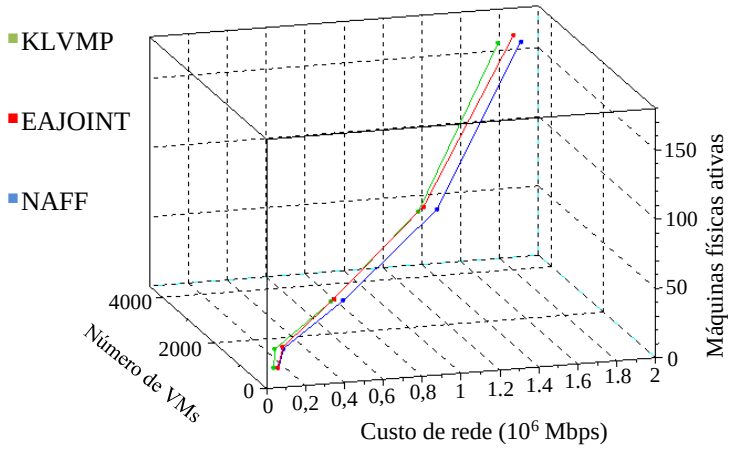
de comparação foram os valores custo de rede e o número de *hosts* ativos produzidos pelas diferentes soluções geradas por estes algoritmos, de forma que, quanto menores esses valores, melhor é a solução produzida. Destes testes é possível observar o seguinte a respeito do algoritmo *KLVMP*:

- **Melhor custo de rede:** Em todos os testes o algoritmo *KLVMP* apresentou melhor custo de rede que os demais, perdendo em apenas 2 cenários dos 20 testados com o algoritmo *EAJOINT* e em apenas 1 cenário dos 20 testados com o *NAFF*.

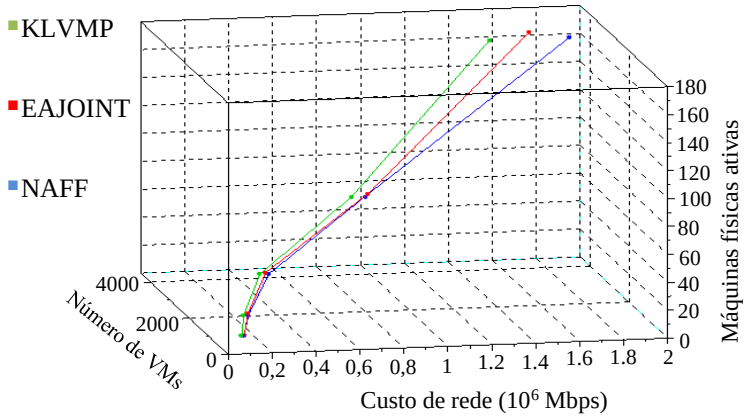
- **Resultado parcialmente melhor na quantidade de *hosts* ativos:** Em

Figura 14 – Dispersão tridimensional para o melhor caso de posicionamento.

(a) Dispersão tridimensional para *28-fat-tree*



(b) Dispersão tridimensional para *48-fat-tree*



todos o testes o algoritmo *KLVMP* apresentou uma quantidade de *hosts* ativos menores que o *EAJOINT* e iguais ao *NAFF*. No entanto, comparando o custo de rede, o algoritmo *KLVMP* obteve um resultado melhor, o que pode ser levado em conta como critério de desempate.

O empate do algoritmo *KLVMP* com o *NAFF* acontece devido ao método construtivo o *KLVMP*, apresentado no Algoritmo 1, que nada mais é que uma abordagem baseada em *first-fit*, assim como o *NAFF*.

Os cenários de teste utilizados permitem também um estudo a respeito de como a heterogeneidade do sistema, do ponto de vista topológico, pode gerar conflito entre esses dois objetivos. Nesse sentido é possível separar as conclusões entre os cenários em que a dispersão é mais heterogênea, isto é, *hosts* de maior capacidade estando distantes um dos outros, ou mais homogênea, com *hosts* de

maior capacidade mais próximos entre si. Iniciando então pelo que seria o pior caso (maior heterogeneidade) se tem:

- Algoritmos com *hosts* ordenados por saltos, que são os de índice 1 ($KLVMP^1$, $EAJOINT^1$ e $NAFF^1$), apresentam um custo de rede mais baixo e uma quantidade de *hosts* ativos mais alta se comparado aos de índice 2, onde se utiliza a ordenação por capacidade.
- Quanto a variação da dimensão do *datacenter*, os algoritmos de índice 1 apresenta um menor custo de rede para *datacenters* com *switches* de 48 portas. Isso ocorre pois aumentando o número de portas aumenta-se também a dimensão de seus *pods*, podendo assim acomodar VMs mais próximas umas das outras.
- Já para as variações de índice 2, em que os *hosts* estão ordenados por capacidade, o custo de rede é ligeiramente maior para *datacenters* com *switches* de 48 portas para cargas de até 2048 VMs, com um aumento mais significativo nos casos com 4096 VMs, o que pode ser explicado pelo fato de que na ordenação por capacidade em redes *datacenters* maiores há uma maior probabilidade de *hosts* com alta capacidade estarem mais distante entre si, devido a heterogeneidade do sistema.

Para o caso de melhor posicionamento, é possível observar o seguinte:

- Ao variar a dimensão do *datacenter*, para cenários com maior demanda tem-se, em média, uma melhora em ambas as funções objetivo em decorrência de *hosts* de maiores capacidade estarem próximos entre si.
- A melhora de resultados é mais rápida para o custo de rede, que é melhor para diâmetros menores já a partir de 512 VMs.
- Quanto ao número de *hosts* ativos a melhora ocorre apenas com mais de 1024 VMs para todos os algoritmos avaliados.
- Comparado ao caso de pior posicionamento, todos os algoritmos testados conseguiram atingir o melhor resultado de cada função objetivo de suas variantes (índice 1 e 2) nos casos de testes de pior posicionamento.

Ao observarmos as Figuras 12 e 14, relativas aos gráficos de dispersão dos valores objetivos, é possível identificar que os três algoritmos implementados, bem como suas variantes, apresentam comportamentos semelhantes dentro de um mesmo caso de teste. Além disso, o algoritmo proposto, *KLVMP*, apresentou resultados melhores que o *EAJOINT* em praticamente todas as situações e obteve resultados melhores quanto ao custo de rede e iguais quanto ao número de *hosts* ativos quando comparado ao *NAFF*.

Ao comparar os diferentes casos de posicionamento, fica claro que, para sistemas heterogêneos, o posicionamento dos *hosts* interfere drasticamente no espaço de busca por soluções, impactando na qualidade das soluções geradas. No caso de pior posicionamento, tem-se duas variantes de um mesmo algoritmo, cada uma focada em otimizar um objetivo em detrimento de outro. Já no caso de melhor posicionamento, é possível otimizar ambos objetivos em apenas uma solução.

5.7 Tempo de execução

Durante as fases de testes foram medidos o tempo de execução dos algoritmos em cada caso. Os algoritmos foram executados em um computador com 32 GB de memória *RAM* e processador *Intel Xeon E5-2420* 1,9 GHz, pertencente ao GSPD – Grupo de Sistemas Paralelos e Distribuídos – da UNESP.

Na Tabela 7 são apresentados os tempos de execução, em segundos, para os algoritmos *KLVMP*[1] e *EAJOINT*[1]. Aqui são apresentados apenas os tempos para os casos de testes usando o cenário *28-fat-tree*. Os tempos de execução para o algoritmo *NAFF*[1] não são apresentados por serem desprezíveis quando comparados aos outros algoritmos, ficando abaixo de um segundo para o maior caso.

O tempo de execução reduzido para o *NAFF* se explica pois sua complexidade computacional é de $O(n^2)$, já que o algoritmo apenas ordena a entrada de VMs e realiza o *first-fit*. A justificativa para a proposição de algoritmos mais lentos que o *NAFF* é de que este algoritmo não produz resultados bons para o custo de rede e que o problema não demanda resposta rápida em sua solução.

A não necessidade de resposta rápida pode ser confirmada a partir do

Tabela 7 – Tempo de execução de algoritmos para *28-fat-tree* no pior caso de posicionamento.

Algoritmo	256 VMs	512 VMs	1024 VMs	2048 VMs	4096 VMs
KLVMP[1]	1	4	25	64	94
EAJOINT[1]	38	24	572	2288	13150

estudo de MENG, PAPPAS e ZHANG (2010), apresentado na Subseção 4.1.1, em que constata, a partir de um estudo em um ambiente real, que as máquinas virtuais apresentam um padrão de comunicação que se mantém por mais de semanas. Assim, apesar do tempo de execução relativamente alto de muitas das abordagens propostas, a solução gerada terá um tempo de vida prolongado, não sendo necessário executar o algoritmo frequentemente.

O tempo total de execução destes algoritmos depende basicamente do tamanho da entrada de VMs e da sua malha de comunicação e essa malha pode ser diferente em cada caso, o crescimento da compĺexidade do problema não é uniforme. Quanto mais arestas possuir o grafo que representa uma malha, maior será o tempo total de busca pela solução do problema. Assim, se o caso envolver *clusters* de grande tamanho, o tempo de execução será maior. A previsão de complexidade da ordem de n^4 assumia a existência de um agrupamento com todas as VMs. Como isso não ocorre, então o tempo de execução segue um crescimento menor.

A grande vantagem que o algoritmo *KLVMP* possui neste sentido é separar a entrada de VMs em *clusters*, de forma que todo *cluster* forme uma malha contínua de comunicação, e assim, tratar cada um desses *clusters* individualmente. Este tipo de estratégia de divisão e conquista provê um ganho significativo no tempo total de execução do algoritmo, como é mostrado na Tabela 7.

5.8 Considerações finais

Os testes executados evidenciaram que as soluções geradas com o *KLVMP* são, de modo geral, melhores do que as soluções geradas pelos outros algoritmos. Os estudos realizados permitiram também verificar que a possível heterogeneidade na disposição topológica das máquinas físicas tem impacto significativo no comportamento dos algoritmos. Fica claro, portanto, que propostas como a

que é feita aqui são ainda necessárias para diminuir a influência de parâmetros externos no desempenho de sistemas de computação em nuvem.

6 Conclusões

Neste trabalho se apresentou um algoritmo para resolver o problema de alocação de máquinas virtuais em sistemas de computação em nuvem, buscando otimizar tanto o número de *hosts* ativos como o custo de rede.

O algoritmo proposto, denominado *Kernighan-Lin based heuristic for Virtual Machine Placement (KLVMP)*, faz uma abordagem heurística que adapta o método de *Kernighan-Lin* para o problema de alocação de máquinas virtuais. Como o algoritmo original, representado na Figura 4, trata o problema quadrático de alocação com n instalações para n locações, foi preciso modificar o componente *KLMC*, que trata a divisão de um grafo em partições, de forma que mais de um vértice possa estar em uma partição, em que há um peso entre cada partição.

É essa modificação que permite mapear um problema quadrático de alocação para uma entrada do algoritmo *KLMC*, atribuindo apenas um vértice para cada partição. Ainda assim, é possível simplificar esta entrada, desde que seja possível agrupar locações que possuam peso zero entre si dentro de uma mesma partição, de forma que entre toda locação l, dentro de uma partição p e k em uma partição q, o peso seja de mesmo valor que o peso entre p e q, para quaisquer partições p e q e quaisquer locações l e k do sistema. No caso do problema de alocação de máquinas virtuais, esta simplificação é válida, pois considerando que cada locação seja uma porção de recursos dentro de uma mesma máquina, e cada máquina seja uma partição, então o peso de cada locação entre cada locação, sera o mesmo peso entre as máquinas que as hospedam.

A eficiência do algoritmo *KLVMP* foi avaliada por meio de simulações, nas quais foram modelados *datacenters* heterogêneos com milhares de nós de processamento conectados em topologia *fat-tree*. Os testes foram aplicados também a outros algoritmos, um do estado da arte e outra heurística comum para a solução do problema, tendo o algoritmo *KLVMP* apresentado resultados interessantes, principalmente quanto ao custo de comunicação.

6.1 Contribuições

Pode-se destacar como contribuições deste trabalho o desenvolvimento de uma nova abordagem para o problema de alocação de máquinas virtuais aplicando o método de *kernighan-Lin*. A nova abordagem foi necessária para resolver o problema considerando múltiplas partições com pesos entre elas, o que não ocorre no algoritmo original.

Além do desenvolvimento do algoritmo, o trabalho também apresenta um estudo de como a heterogeneidade de um *datacenter* pode afetar o balanço entre otimização de rede e da quantidade de *hosts* ativos. Neste sentido, de acordo com os resultados obtidos, quando os *hosts* de maiores capacidades do sistema estiverem o mais próximo possível, é possível atender ambos os objetivos com harmonia. Isso é diferente quando os *hosts* de maiores capacidades estão distantes entre si, em que priorizar um objetivo afeta negativamente o outro.

6.2 Trabalhos futuros

Durante a execução deste trabalho foi possível identificar problemas relacionados que não poderiam ser resolvidos naquele momento. Uma adição que tem sido bem considerada na literatura é a inclusão de parâmetros relacionados ao consumo de energia. Há um consenso de que reduzir os custos de rede e a quantidade de *hosts* ativos pode reduzir também o consumo de energia elétrica. No entanto, neste trabalho não são quantificados gastos em energia elétrica. Dessa forma, como trabalho futuro, é possível modificar o problema incluindo objetivos e restrições relacionados diretamente a gastos em energia elétrica e assim modificar a abordagem para atender estas demandas.

O número de *hosts* ativos é determinado pelo método construtivo (Algoritmo 1), tendo influência direta no espaço de busca pelo aprimoramento no custo de rede. Assim, é possível testar outros métodos, podendo sacrificar o número de máquinas ativas para um melhor custo de rede e vice-versa.

Outra linha de trabalho envolve a aplicação de componentes do *KLVMP* em outros áreas da ciência, que tenham problemas que possam ser resolvidos com partes do algoritmo, como ocorre com o Problema Quadrático de Alocação.

Referências

AL-FARES, M.; LOUKISSAS, A.; VAHDAT, A. A scalable, commodity data center network architecture. In: ACM. *ACM SIGCOMM Computer Communication Review*. [S.l.], 2008. v. 38, n. 4, p. 63–74. Citado 2 vezes nas páginas 26 e 27.

ALAHMADI, A. et al. Enhanced first-fit decreasing algorithm for energy-aware job scheduling in cloud. *Computational Science and Computational Intelligence (CSCI)*, p. 69–74, 2014. Citado na página 42.

AMAZON. Estudo de caso da aws: Netflix. Disponível em <https://aws.amazon.com/pt/solutions/case-studies/netflix/>. 2018. Citado na página 15.

ARMOUR, G. C.; BUFFA, E. S. A heuristic algorithm and simulation approach to relative location of facilities. *Management Science*, INFORMS, v. 9, n. 2, p. 294–309, 1963. Citado na página 31.

BACHIEGA, N. G. et al. Virtual machine instance scheduling in iaas clouds. *Proceedings of the International Conference on Parallel and Distributed Processing Techniques and Applications (PDPTA)*, The Steering Committee of The World Congress in Computer Science, Computer Engineering and Applied Computing (WorldComp), 2014. Citado na página 43.

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. Citado na página 39.

BERTSEKAS, D. P. et al. *Dynamic programming and optimal control*. [S.l.]: Athena scientific Belmont, MA, 1995. v. 1. Citado na página 31.

BOBROFF, N.; KOCHUT, A.; BEATY., K. Dynamic placement of virtual machines for managing sla violations. *Integrated Network Management, 2007. IM'07. 10th IFIP/IEEE International Symposium*, 2007. Citado na página 35.

BUYYA, R.; BROBERG, J.; GOSCINSKI, A. *Cloud Computing: Principles and Paradigms*. Wiley, 2011. (Wiley Series on Parallel and Distributed Computing). ISBN 9781118002209. Disponível em: <http://books.google.com.br/books?id=S1NvRRd77rQC>. Citado 9 vezes nas páginas 9, 15, 19, 20, 21, 22, 23, 27 e 30.

CAO, B. et al. Nice: network-aware vm consolidation scheme for energy conservation in data centers. In: IEEE. *Parallel and Distributed Systems (ICPADS), 2014 20th IEEE International Conference on*. [S.l.], 2014. p. 166–173. Citado na página 40.

CLOUDSTACK. Apache cloudstack - open source cloud computing. Disponível em <https://cloudstack.apache.org/>. 2018. Citado na página 25.

COFFMAN, E. G.; GAREY, M. R.; JOHNSON, D. S. Approximation algorithms for bin-packing — an updated survey. In: *Algorithm design for computer system design*. [S.l.]: Springer, 1984. p. 49–106. Citado na página 31.

DONG, J.; WANG, H.; CHENG, S. Energy-performance tradeoffs in IaaS cloud with virtual machine scheduling. *Communications,China, 12 (2), pp.155-166*, 2015. Citado na página 39.

DORIGO, M.; BIRATTARI, M. Ant colony optimization. In: *Encyclopedia of machine learning*. [S.l.]: Springer, 2011. p. 36–39. Citado na página 34.

GLOVER, F. Tabu search—part i. *ORSA Journal on computing*, INFORMS, v. 1, n. 3, p. 190–206, 1989. Citado na página 35.

GOMORY, R. E.; HU, T. C. Multi-terminal network flows. *Journal of the Society for Industrial and Applied Mathematics*, SIAM, v. 9, n. 4, p. 551–570, 1961. Citado 2 vezes nas páginas 38 e 64.

GONZALEZ, T. F. Clustering to minimize the maximum intercluster distance. *Theoretical Computer Science*, Elsevier, v. 38, p. 293–306, 1985. Citado na página 38.

GOUDARZI, H.; PEDRAM, M. Energy-efficient virtual machine replication and placement in a cloud computing system. In: IEEE. *Cloud Computing (CLOUD), 2012 IEEE 5th International Conference on*. [S.l.], 2012. p. 750–757. Citado na página 36.

GREENBERG, A. et al. Vl2: a scalable and flexible data center network. In: ACM. *ACM SIGCOMM computer communication review*. [S.l.], 2009. v. 39, n. 4, p. 51–62. Citado na página 27.

GUO, C. et al. Bcube: a high performance, server-centric network architecture for modular data centers. *ACM SIGCOMM Computer Communication Review*, ACM, v. 39, n. 4, p. 63–74, 2009. Citado na página 27.

HUANG, D. et al. Energy-aware virtual machine placement in data centers. In: IEEE. *Global Communications Conference (GLOBECOM), 2012 IEEE*. [S.l.], 2012. p. 3243–3249. Citado 3 vezes nas páginas 38, 61 e 63.

HYPER-V. Visão geral do hyper-v. Disponível em <https://msdn.microsoft.com/pt-br/library/hh831531(v=ws.11).aspx>. 2018. Citado na página 24.

KERNIGHAN, B. W.; LIN, S. An efficient heuristic procedure for partitioning graphs. *The Bell system technical journal*, Nokia Bell Labs, v. 49, n. 2, p. 291–307, 1970. Citado 5 vezes nas páginas 9, 40, 50, 53 e 54.

KNAUTH, T.; FETZER, C. Energy-aware scheduling for infrastructure clouds. *2012 IEEE 4th International Conference on Cloud Computing Technology and Science*, 2012. Citado na página 41.

KUHN, H. W. The hungarian method for the assignment problem. *Naval research logistics quarterly*, v. 2, n. 1-2, p. 83–97, 1955. Citado na página 36.

KVM. Kernel based virtual machine. Disponível em <http://www.linux-kvm.org/page/Main_Page>. 2019. Citado na página 24.

LAARHOVEN, P. J. V.; AARTS, E. H. Simulated annealing. In: *Simulated annealing: Theory and applications*. [S.l.]: Springer, 1987. p. 7–15. Citado na página 35.

LAWLER, E. L. The quadratic assignment problem. *Management science*, INFORMS, v. 9, n. 4, p. 586–599, 1963. Citado na página 31.

LAWLER, E. L.; WOOD, D. E. Branch-and-bound methods: A survey. *Operations research*, INFORMS, v. 14, n. 4, p. 699–719, 1966. Citado na página 31.

LEISERSON, C. E. Fat-trees: universal networks for hardware-efficient supercomputing. *IEEE transactions on Computers*, v. 100, n. 10, p. 892–901, 1985. Citado na página 25.

MELL, P.; GRANCE, T. *The NIST Definition of Cloud Computing*. Gaithersburg, MD, 2011. Disponível em: <http://csrc.nist.gov/publications/nistpubs/800-145/SP800-145.pdf>. Citado na página 19.

MENG, X.; PAPPAS, V.; ZHANG, L. Improving the scalability of data center networks with traffic-aware virtual machine placement. *INFOCOM, 2010 Proceedings IEEE*, p. 1–9, 2010. Citado 3 vezes nas páginas 37, 48 e 82.

MI, H. et al. Online self-reconfiguration with performance guarantee for energy efficient large-scale cloud computing data centers. *Services Computing (SCC), 2010 IEEE International Conference*, p. 514–521, 2010. Citado na página 43.

MICROSOFT. Virtual machine manager. Disponível em <https://msdn.microsoft.com/pt-br/library/gg610610(v=sc.12).aspx>. 2018. Citado na página 25.

NURMI, D. et al. The eucalyptus open-source cloud-computing system. in cluster computing and the grid. *CCGRID'09. 9th IEEE/ACM International Symposium*, p. 124–131, 2009. Citado na página 24.

OPENNEBULA. Opennebula - flexible enterprise cloud made simple. Disponível em <https://opennebula.org/>. 2018. Citado na página 24.

OPENSTACK. Openstack open source cloud computing software. Disponível em <https://www.openstack.org/>. 2018. Citado na página 24.

QEMU. Qemu. Disponível em <https://www.qemu.org/>. 2018. Citado na página 24.

RITTINGHOUSE, J.; RANSOME, J. F. *Cloud Computing: Implementation, Management, and Security*. [S.l.]: CRC, 2009. Citado 2 vezes nas páginas 18 e 21.

SHI, W.; HONG, B. Towards profitable virtual machine placement in the data center. *2011 Fourth IEEE International Conference on Utility and Cloud Computing*, p. 138–145, 2011. Citado na página 42.

TAO, F. et al. Bgm-bla: a new algorithm for dynamic migration of virtual machines in cloud computing. *IEEE Transactions on Services Computing*, PP, 2015. Citado na página 36.

VMWARE. WMware ESXi e ESX. Disponível em <http://www.vmware.com/>. 2018. Citado na página 23.

WANG, S. H. et al. Energy-efficient and qos-aware virtual machine placement for software defined datacenter networks. *The International Conference on Information Networking 2014 (ICOIN2014)*, p. 220–225, 2014. Citado na página 41.

WHITLEY, D. A genetic algorithm tutorial. *Statistics and computing*, Springer, v. 4, n. 2, p. 65–85, 1994. Citado na página 34.

XEN. Why Xen Project? Disponível em <http://www.xenproject.org/users/why-the-xen-project.html>. 2018. Citado na página 23.

XU, J.; FORTES, J. Multi-objective virtual machine placementint virtualized data center environments. *2010 IEEE/ACM International Conference on Green Computing and Communications*, 2010. Citado na página 37.

ZHANG, Q.; CHENG, L.; BOUTABA, R. Cloud computing: state-of-the-art and research challenges. *Journal of internet services and applications*, Springer, v. 1, n. 1, p. 7–18, 2010. Citado na página 25.