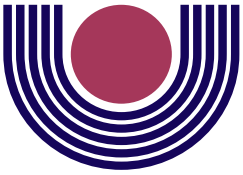

Unioeste - Universidade Estadual do Oeste do Paraná
CENTRO DE CIÊNCIAS EXATAS E TECNOLÓGICAS
Colegiado de Ciência da Computação
Curso de Bacharelado em Ciência da Computação

Análise comparativa entre diferentes algoritmos de alocação de máquinas virtuais em ambientes de computação em nuvens

Gabriel Sanches Silva

CASCAVEL
2015

Gabriel Sanches Silva

Análise comparativa entre diferentes algoritmos de alocação de máquinas virtuais em ambientes de computação em nuvens

Monografia apresentada como requisito parcial para obtenção do grau de Bacharel em Ciência da Computação, do Centro de Ciências Exatas e Tecnológicas da Universidade Estadual do Oeste do Paraná - Campus de Cascavel

Orientador: Guilherme Galante

CASCAVEL
2015

Gabriel Sanches Silva

ANÁLISE COMPARATIVA ENTRE DIFERENTES ALGORITMOS DE ALOCAÇÃO DE MÁQUINAS VIRTUAIS EM AMBIENTES DE COMPUTAÇÃO EM NUVENS

Monografia apresentada como requisito parcial para obtenção do Título de Bacharel em Ciência da Computação, pela Universidade Estadual do Oeste do Paraná, Campus de Cascavel, aprovada pela Comissão formada pelos professores:

Prof. Guilherme Galante (Orientador)
Colegiado de Ciência da Computaçao,
UNIOESTE

Prof. Marcio Seiji Oyamada
Colegiado de Ciência da Computaçao,
UNIOESTE

Prof. Luiz Antonio Rodrigues
Colegiado de Ciência da Computação,
UNIOESTE

Cascavel, 18 de fevereiro de 2016

DEDICATÓRIA

Dedico esse trabalho para toda a minha família que me ajudaram muito durante todo o tempo, a todos os meus amigos que de alguma forma me apoiaram, aos professores que me deram a formação necessária e a minha namorada que me guiou para essa conquista tão importante.

AGRADECIMENTOS

Primeiramente gostaria de agradecer a toda a minha família que está junto comigo todos os dias, eles fazem uma grande parte desse trabalho de conclusão de curso, me apoiaram desde o primeiro dia de aula até o último dia. Vou ser eternamente grato a toda o auxílio e atenção que me prestaram. Quero ainda lembrar da parte da minha família que mora em Maringá, meus avós que sempre me incentivaram a buscar o melhor para mim, aos meus tios e tias por esclarecer os meus pensamentos e aos meus primos pelos momentos bons.

Ainda gostaria de agradecer a todos os meus amigos e colegas do curso que de alguma forma me apoiaram e contribuíram para a conclusão do curso. Em especial aos meus grandes amigos Henrique e Rigon que estão comigo desde antes da faculdade e que com certeza me apoiaram de uma forma muito especial durante esses quatro anos de muito estudo.

Não poderia esquecer de agradecer aos professores do curso de Ciência da Computação da Unioeste campus de Cascavel, que sempre se disponibilizaram a ajudar na minha formação, aos meus tutores de pesquisa que sem dúvida me deram a base para a carreira acadêmica e ao meu orientador que me orientou muito bem para a conclusão desse trabalho de conclusão de curso.

Por fim gostaria de agradecer a pessoa que fez toda a diferença para a minha vida no último ano do curso, faz um ano que estamos juntos e desde então ela vem me ajudando em tudo de forma tão especial que eu só tenho a agradecer por todos os momentos juntos, que foram tantos, realmente foi o melhor ano da minha vida. Obrigado Fer por trazer tanta alegria e felicidade para essa conquista.

Lista de Algoritmos

Lista de Figuras

Lista de Tabelas

Lista de Abreviaturas e Siglas

CEO	*Chief Executive Officer*
CPU	*Central Processing Unit*
IaaS	*Infrastructure as a Service*
NIST	*National Institute of Standards and Technology*
NP	*Non-deterministic Polynomial-time*
PaaS	*Platform as a Service*
PM	*Physical Machine*
SaaS	*Software as a Service*
SLA	*Service Level Agreement*
VM	*Virtual Machine*
VMP	*Virtual Machine Placement*

Sumário

Resumo

Uma arquitetura de computação de nuvem fornece dinamicamente inúmeras máquinas virtuais, utilizadas para diversos tipos de serviços. Nesse contexto a alocação de máquinas virtuais é um dos problemas mais importantes na gestão de uma infraestrutura de nuvem, considerando o grande número de critérios e diferentes formulações para o problema. Nesse trabalho foi realizado um estudo comparativo das soluções encontradas na literatura. A partir do estudo realizado foi proposto um *framework* para implementação de algoritmos de alocação de máquinas virtuais por pesquisadores. O objetivo da proposta é que dado as características da infraestrutura da nuvem em um determinado ambiente de simulação, fosse possível implementar e comparar os algoritmos de acordo com suas métricas. Para a validação do *framework* foram implementados 7 algoritmos de alocação de máquinas virtuais, um dos algoritmos implementados foi escolhido do estudo comparativo. Para a comparação dos algoritmos implementados foram realizados 63 testes diferentes, com diferentes configurações de ambientes de nuvem. Os resultados obtidos mostram como um algoritmo de alocação pode interferir diretamente na eficiência da infraestrutura de nuvem.

Palavras-chave: computação em nuvem, máquinas virtuais, ambiente de simulação, framework

Capítulo 1

Introdução

Segundo Vaquero [36] a computação em nuvem pode ser definida como:

> "Um grande conjunto de recursos virtualizados e compartilhados (hardware, plataformas de desenvolvimento ou software) que podem ser facilmente acessados e utilizados. Esses recursos podem ser dinamicamente reconfigurados para se ajustarem a uma carga variável de trabalho, permitindo uso otimizado dos mesmos. Este conjunto de recursos é tipicamente explorado por um modelo de pagamento por utilização (*pay-per-use*) em que as garantias são oferecidas pelo provedor de infraestrutura por meio de contratos de serviço personalizados (*Service Level Agreement* – SLA)".

Assim, com o surgimento deste modelo de computação, recursos de computação (por exemplo, redes, servidores, armazenamento, aplicações) passam a ser provisionados como serviços sob demanda através de redes com um modelo de pagamento com base no uso. Os recursos podem ser rapidamente alocados e liberados com um esforço mínimo de gerenciamento [9]. O compartilhamento desses recursos é garantido pelo uso de virtualização, que permite que vários ambientes virtuais sejam executados sobre o hardware físico, permitindo a otimização do uso de recursos e a economia de escala [16].

Existem três modelos básicos de negócios relacionados a computação nas nuvens: Infraestrutura como Serviço (*Infrastructure as a Service* – IaaS), Plataforma como Serviço (*Platform as a Service* – PaaS) e Software como Serviço (*Software as a Service* – SaaS). Neste trabalho aborda-se o modelo de infraestrutura como serviço, onde os recursos computacionais como armazenamento, rede e recursos computacionais são provisionados como serviços [26].

Dentro do contexto de prover infraestrutura e recursos necessários para os serviços, entra a ideia da virtualização, com o objetivo de encapsular recursos em uma ou mais máquinas virtuais (*Virtual Machine - VM*), que podem ser alocadas em várias máquinas físicas (*Physical Machine - PM*). Em ambientes de nuvem a demanda de alocação de recursos pode variar dinamicamente, no qual controladores de alocação definem quantas instâncias serão utilizadas para rodar o serviço, onde alocá-los e como executá-los em um ambiente com restrições de recursos. A alocação de VMs nesse contexto pode ser classificada como um problema NP-difícil.

Assim, aperfeiçoar a alocação/realocação de VMs, ou seja, encontrar uma melhor maneira de alocar os recursos e reconfigurar as VMs de acordo com as mudanças de ambientes, torna-se um desafio, pois não existe um modelo genérico que represente vários cenários de alocação de VMs. Dessa forma, encontrar parâmetros adequados ao problema de alocação de VMs é tipicamente formulado como variante do problema da mochila que é conhecido por ser um tipo de problema combinatorial e NP-difícil [26].

Atualmente existem vários estudos e soluções para esse problema, cada um com diferentes objetivos, baseados em diferentes métricas, por exemplo, desempenho, eficiência energética, utilização da rede, redução de custo e utilização de recursos [32]. É possível encontrar na literatura soluções objetivando a minimização da utilização de VMs [2], otimização de alocação de VMs levando em consideração o o conhecimento da topologia de rede [3], a minização do uso de recursos em cada servidor [9], otimização de múltiplos objetivos considerando o SLA [31], e a minimização do congestionamento na rede mantendo o consumo de energia inalterado [37].

Um meio de validar as soluções dos algoritmos encontradas na literatura é realizar uma análise comparativa entre diferentes algoritmos de alocação de VMs de acordo com as suas métricas. Para isso propoe-se uma abordagem para testar algoritmos onde o pesquisador possa implementar facilmente o seu próprio algoritmo e comparar os resultados obtidos com outros algoritmos usando um mesmo ambiente de execução.

Considerando o estado-da-arte sobre o assunto, o objetivo da presente pesquisa é o desenvolvimento de um *framework* para implementação e teste de algoritmos de alocação de VMs. As simulações serão realizadas pelo CloudSim, que é uma ferramenta que permite a modelagem e simulação de sistemas de computação em nuvem e ambientes de provisionamento e alocação de aplicações. Com essa ferramenta é possível avaliar o desempenho das políticas de provisi-

onamento em nuvem sob uma variedade de configurações e requisitos de sistema e de usuário. Além disso, ele expõe interfaces personalizadas para a implementação de políticas e técnicas para a atribuição de VMs provisionadas sob cenários de computação em rede [8].

Para a validação do *framework* proposto foram implementados 7 algoritmos de alocação de VMs, foram realizados 63 testes com esses algoritmos com o objetivo de verificar o desempenho de cada algoritmo sobre diferentes cenários. Com os resultados obtidos foi possível observar a importância de se comparar algoritmos de alocação de VMs em diferentes ambientes de execução.

Esse trabalho será organizado da seguinte maneira: o Capítulo 2 aborda os principais conceitos sobre computação em nuvem, citando uma série de características essenciais de cada ambiente de nuvem, junto aos modelos de serviço e implantação introduzidos pela literatura. O Capítulo 3 introduz o problema da alocação de VMs, bem como as métricas utilizadas para a classificação dos trabalhos e apresenta uma revisão de estado-da-arte sobre o assunto. O Capítulo 4 apresenta o *framework* desenvolvido nesse trabalho, a estrutura do CloudSim e a implementação do *framework*. O Capítulo 5 aborda os testes realizados no *framework* e os resultados obtidos. O Capítulo 6 apresenta as considerações finais desse trabalho.

Capítulo 2

Computação em Nuvem

2.1 Conceitos

O conceito de Computação em Nuvem (*Cloud Computing*) surgiu em 2006, quando o diretor executivo (*Chief Executive Officer* - CEO) da empresa Google, Eric Schmidt, utilizou o termo para definir o modelo de negócio de sua empresa. Atualmente muito tem sido escrito e falado sobre Computação em Nuvem, mas ainda existem muitas contradições relacionadas a esse termo, ainda não existindo uma definição única, o que mostra que a ideia de Computação em Nuvem ainda está em crescimento [16]. Existem algumas definições bastante abrangentes que podem ser tomadas como referência, tais como a de Foster [15]:

> "*Cloud Computing* é um paradigma de computação em larga escala que possui foco em proporcionar economia de escala, em que um conjunto abstrato, virtualizado, dinamicamente escalável de poder de processamento, armazenamento, plataformas e serviços são disponibilizados sob demanda para clientes externos através da Internet."

Outra definição conceituada é a de Vaquero [36] onde o autor utiliza várias definições diferentes para formalizar um conceito unificado. Segundo ele, a Computação em Nuvem é um grande conjunto de recursos virtualizados e compartilhados (hardware, plataformas de desenvolvimento ou software) que podem ser utilizados e acessados por clientes por meio da rede. Esses recursos podem ser dinamicamente alocados e reconfigurados para se adaptarem a uma carga variável de trabalho, sendo possível otimizar os mesmos. Este conjunto de recursos é explorado por um modelo de pagamento por uso (*pay-per-use*) onde as garantias são ofereci-

das pelo provedor dos recursos por meio de contratos de serviço personalizados (*Service Level Agreement* - SLA).

De acordo com Mell et al. [28], o modelo computacional de nuvem tem como base o uso de uma infraestrutura composta por um conjunto (*pool*) de servidores e de toda a infraestrutura necessária para seu funcionamento (eletricidade, refrigeração, rede de interconexão e armazenamento), que são compartilhados entre inúmeros usuários (*multi-tenancy*). O compartilhamento é feito através do uso de virtualização, que permite que vários ambientes virtuais sejam executados sobre o hardware físico, permitindo a otimização do uso de recursos e a economia de escala. O acesso aos recursos fornecidos é feito remotamente por meio de uma rede ou Internet.

Seguindo essas definições e outras vistas na literatura [4] podemos determinar que a computação em nuvem é um modelo de computação, uma infraestrutura de rede que possui recursos para aplicações de multi-usuários/clientes em um ou mais servidores (física ou virtual), particularmente usada para aplicações de demanda imprevisíveis. Logo essa ideia passa a ser um modelo de negócios que é compartilhado entre vários clientes. Esse compartilhamento de recursos é realizado através do uso de virtualização, onde vários ambientes com diferentes recursos são executados em *datacenters* (centro de dados). O provisionamento desses recursos podem ser otimizados seguindo diferentes critérios/métricas que serão discutidas no Capítulo 3.

2.2 Características essenciais

Seguindo os conceitos do *National Institute of Standards and Technology* (NIST) a computação em nuvem possui cinco características essenciais reconhecidas na Figura 2.1 e descritas a seguir:

- Auto-serviço sob demanda (*On-demand self-service*): é o aspecto em que o provedor pode fornecer recursos computacionais (processamento, memória e armazenamento), conforme a demanda do usuário, sem haver a necessidade da intervenção de administradores do serviço.

- Amplo acesso via rede (*Broad network access*): os recursos da nuvem ficam disponíveis através da rede e é possível acessá-los através de mecanismos padronizados que promovem o uso por dispositivos clientes de diversas plataformas tais como, smartphones,

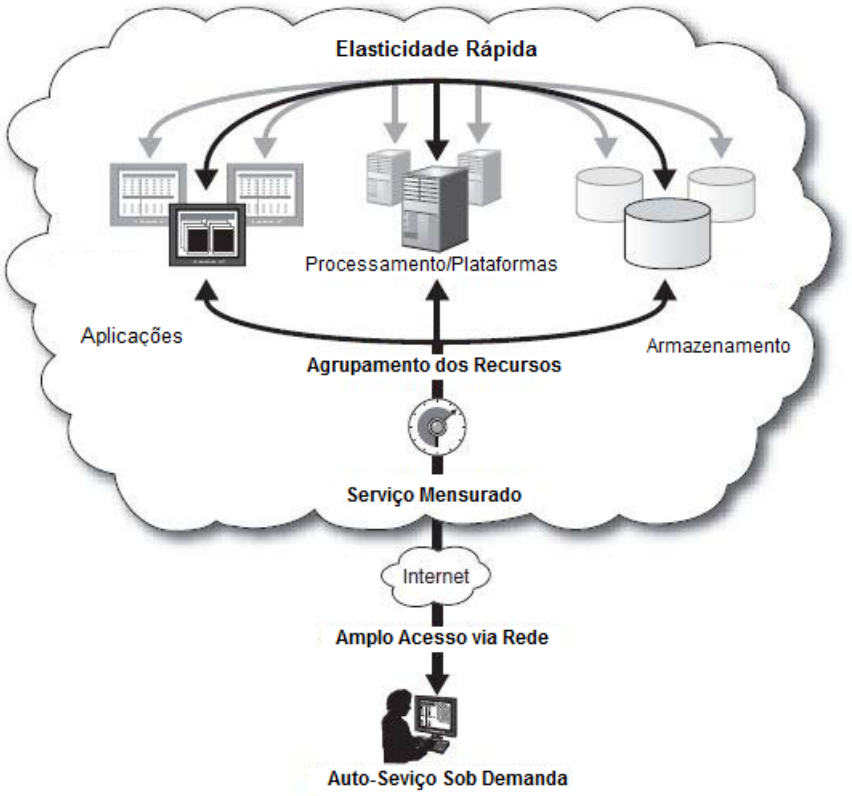

Figura 2.1: Características essencias da computação em nuvem. Adaptada de Bolin [4].

tablets, laptops ou desktops.

- Agrupamento de recursos (*Resource pooling*): é a característica onde os recursos computacionais do provedor são agrupados para atender a múltiplos usuários, com recursos físicos e virtuais distintos que são atribuídos e distribuídos dinamicamente conforme a demanda.

- Elasticidade rápida (*Rapid elasticity*): define a capacidade de alocar recursos tanto para aumentá-los como diminuí-los. Logo, os recursos podem ser provisionados e liberados elasticamente, quase que automaticamente de acordo com a demanda, passando a ideia de recursos ilimitados que podem ser alocados em qualquer quantidade e a qualquer tempo.

- Serviço mensurado (*Measured Service*): refere-se ao monitoramento e controle de todos os aspectos do serviço da nuvem. O controle e a otimização dos recursos são automáticos e fazem parte do sistema da nuvem, que é transparente tanto para o provedor quanto para o consumidor. Isso permite que os serviços sejam associado ao método pagamento

conforme o uso (*pay-as-you-use*) que relaciona o ajuste de custo de acordo com o uso dos serviços pela nuvem, que compreendem Infraestrutura, Plataforma e Software, como apresentado na seção 2.3.

2.3 Modelos de Serviço

De acordo com o nível de abstração e pelos tipos dos recursos fornecidos pelo provedor, pode-se dividir os modelos de serviço de computação em nuvem em três classes: Infraestrutura como Serviço (*Infrastructure as a Service* - IaaS), Plataforma como Serviço (*Platform as a Service* - PaaS) e Software como Serviço (*Software as a Service* - SaaS) [4, 16].

O modelo de serviço IaaS é caracterizado pelo fornecimento dos recursos fundamentais de um computador como, processamento, servidores, armazenamento, componentes de rede e largura de banda como um serviço. É importante ressaltar os diferentes níveis de controle entre provedor e usuário no modelo de negócios, onde o usuário não possui controle direto sobre os recursos físicos da nuvem, mas pode requisitar VMs ao provedor e possui controle total sobre suas aplicações, sistemas operacionais e VMs. Já o provedor não possui controle das aplicações, sistemas e/ou VMs providas ao usuário, mas monitora o gerenciamento da virtualização, controlando os recursos físicos da nuvem.

O modelo de serviço PaaS prove uma plataforma com sistema operacional, linguagens de programação e ambientes para o desenvolvimento de aplicações, que permite que desenvolvedores desenvolvam e implementem aplicativos Web e sistemas de software. Assim como no modelo SaaS, o usuário não gerencia e nem possui controle a infraestrutura, mas tem total controle sobre as aplicações implantadas. Esse tipo de serviço é referido na literatura como uma pilha de soluções.

O modelo de serviço SaaS tem como objetivo prover o uso de aplicações específicas disponíveis para os usuários que rodam em uma determinada infraestrutura de nuvem. Essas aplicações geralmente são acessadas a partir de vários dispositivos do usuário por meio de uma interface com o cliente, por exemplo, um navegador Web [7]. Neste modelo de serviço o usuário não possui qualquer controle sobre a infraestrutura, incluindo rede, servidores, sistemas operacionais, armazenamento, exceto algumas configurações específicas da aplicação. Temos como exemplo do modelo SaaS os serviços de *Customer Relationship Management* (CRM) da Salesforce, o

Google Docs e o Dropbox [13].

Este trabalho tem como objetivo estudar o modelo IaaS que é responsável pelo provisionamento de recursos físicos da nuvem, junto as funcionalidades fornecidas pela virtualização. Os recursos são fornecidos por um conjunto de recursos interligados, como servidores, armazenamento e dispositivos de rede [16]. A IaaS permite ainda que os recursos sejam fornecidos de forma portável e flexível, onde o usuário pode obter fácil acesso a esses recursos através da internet e pode rapidamente solicitar quantas VMs forem necessárias, pois a escalabilidade dos recursos ocorre quase que em tempo real.

2.4 Modelos de Implantação

Independentemente da classe de serviço prestado, uma nuvem pode ser classificada quanto ao seu modelo de implantação como publica, privada, comunitária ou híbrida [4].

A nuvem privada é um tipo de ambiente de nuvem utilizada unicamente por uma organização, esta nuvem pode estar dentro ou fora das instalações da organização e pode ser administrada tanto pela própria empresa, quanto por terceiros. A infraestrutura de nuvem pública é disponibilizada para o uso aberto ao público em geral, em que o termo "público" não quer dizer que seja isento de pagamento todas as vezes, mas sim que esse ambiente pode ser acessado por qualquer usuário que conheça a localização do serviço. Nuvem comunitária é um modelo de implantação é compartilhada entre uma comunidade de organizações que possuem interesses em comum. O controle e administração dessa infraestrutura pode ser realizada por uma ou mais organizações da comunidade ou terceiros. A nuvem híbrida é uma composição de duas ou mais nuvens, que podem ser do tipo privada, pública ou comunidade, onde cada uma dessas nuvens ainda são tratadas como entidades únicas. A diferença é que cada uma dessas nuvens são conectadas por meio de tecnologia proprietária ou padronizada que permite a portabilidade de dados e aplicações.

A escolha de um determinado modelo depende das necessidades particulares de cada organização, levando-se em consideração questões relacionadas ao custo, segurança e controle. De um lado, as nuvens públicas oferecem recursos sob demanda mas pouco controle. Do outro lado, nuvens privadas, mais seguras e com maior controle, mas toda a aquisição, gerência, e manutenção é responsabilidade da organização. Entre as duas soluções estão as nuvens híbri-

das, que podem trazer benefícios dos dois modelos, ou ainda as nuvens comunitárias, voltadas para projetos colaborativos [16].

2.5 Virtualização

Virtualização é um termo amplamente utilizado na computação em nuvem que se refere à abstração de recursos computacionais, ocultando dos seus usuários algumas de suas características tais como programas, sistemas operacionais e hardware. Através do uso de virtualização, um único computador físico, ou servidor, pode compartilhar seus recursos de hardware entre múltiplos sistemas operacionais hospedados, onde cada hospedeiro possui controle total sobre sua VM ou ainda, em alguns casos, é como se estivesse instalado isoladamente em uma máquina física. Por tanto, uma VM é um ambiente totalmente isolado, com capacidade de executar sistemas operacionais e aplicativos como se fosse uma computador físico [38].

A virtualização de recursos foi inventada pela IBM em 1960, onde naquele tempo os usuários utilizavam cartões perfurados para submeter certos comandos a um sistema coordenado por compartilhamento de tempo. Atualmente a virtualização é reconhecida como um mecanismo de acesso para múltiplos usuários poderem acessar um mesmo servidor de modo escalável, elástico e de acordo com as demandas do usuário. Isso significa que o usuário pode alocar suas aplicações de modo automático. Este processo de requisitar ou liberar recursos é realizado pelo software que serve como mediador entra as múltiplas instâncias de sistemas operacionais em uma mesma máquina física chamado de monitor de VM ou mesmo hipervisor (*hypervisor*), que é o responsável por garantir o isolamento entre as VMs, limitar os recursos de cada uma e definir o modo de acesso [4].

Portanto, a virtualização é uma técnica bastante utilizada em ambientes com diversidade de plataformas de sistemas operacionais sem que seja necessário o aumento no número de máquinas físicas. Cada aplicação pode executar em uma VM própria, incluindo as bibliotecas e sistema operacional que executam em uma plataforma de hardware comum. Assim, a virtualização proporciona um alto nível de portabilidade e flexibilidade, pois, permite que várias aplicações de sistemas operacionais diferentes executem em uma mesma máquina física isoladamente das demais VMs. Além de que ao se executar múltiplas instâncias de VMs em uma mesma VM, também resulta em um uso mais eficiente de processamento [10].

É importante ressaltar que existem duas formas básicas de uso de virtualização de uma VM. O primeiro modo é a virtualização sobre hardware, que executam diretamente sobre o próprio hardware de uma máquina física, onde as VMs são postas sobre o hardware. O objetivo desse modo de virtualização é que os recursos de hardware (processador,memória,meios de armazenamento e dispositivos) sejam compartilhados entra diferentes máquinas físicas de forma que cada uma tenha a ilusão de que os recursos são privados a ela [23]. Exemplos de virtualização em hardware são a tecnologia da Intel chamada *Intel Virtualization Technology* e a AMD com uma tecnologia semelhante chamada *AMD Virtualization*.

O segundo modo é caracterizado por executar as VMs sobre um sistema operacional nativo como na Figura 2.2. Nesse modo de virtualização o hipervisor fornece uma camada de virtualização composta por um sistema operacional hóspede, que pode ser diferente do sistema operacional nativo, é composto por um hardware virtual criado sobre os recursos de hardware do sistema operacional nativo [23]. Alguns exemplos de virtualização sobre um sistema operacional são o software Xen criada pela Universidade de Cambridge[1], o VMWare[2] e o VirtualBox[3].

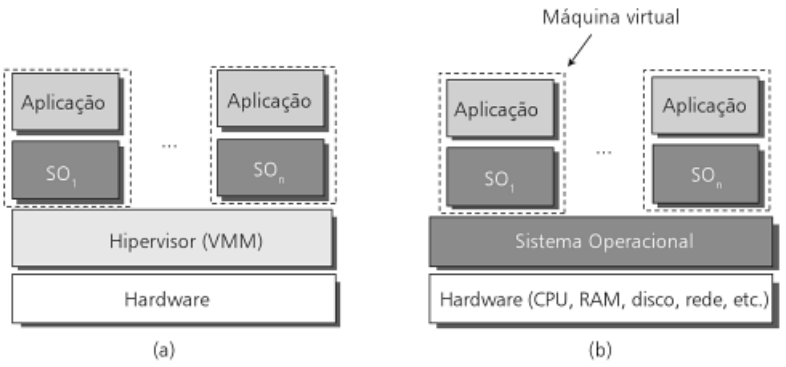

Figura 2.2: Modos de virtualização: a) virtualização direta no hardware, b) virtualização sobre sistema operacional [30].

Existem duas técnicas para a implementação de VMs. A primeira delas é a virtualização total que consiste em fornecer uma camada de abstração de todos os componentes físicos e lógicos de alto e baixo nível, gerando um ambiente virtual onde o sistema operacional visitante é executado e não é necessária nenhuma alteração no sistema hóspede, pois a VM é percebida como a própria máquina física. Porém, isso apresenta um custo adicional de processamento,

[1]https://www.citrix.com.br/products/xenserver/overview.html
[2]http://www.vmware.com/br
[3]https://www.virtualbox.org/

uma vez que cada instrução deve ser testada pelo hipervisor [6]. Esse tipo de arquitetura foi implementada pela VMWare e pela KVM[4].

Na segunda técnica, chamada de paravirtualização, é fornecido uma camada de hardware similar, mas não idêntica ao hardware adjacente. O hipervisor permite ao sistema operacional convidado o acesso direto ao hardware, impondo alguns limites e monitorando a VM. Essa permissão de acesso garante um desempenho similar a de uma máquina física. Um inconveniente é que há uma redução na portabilidade, já que é exigido que o sistema virtualizado seja modificado para funcionar sobre a VM fornecida [6]. Muitos softwares de virtualização utilizam a arquitetura de paravirtualização, por exemplo, o software Xen [38]. É possível observar a diferença das duas técnicas na Figura 2.3, no qual a paravirtualização necessita de uma camada a mais para a tradução dos comandos do sistema virtualizado para comandos do sistema nativo.

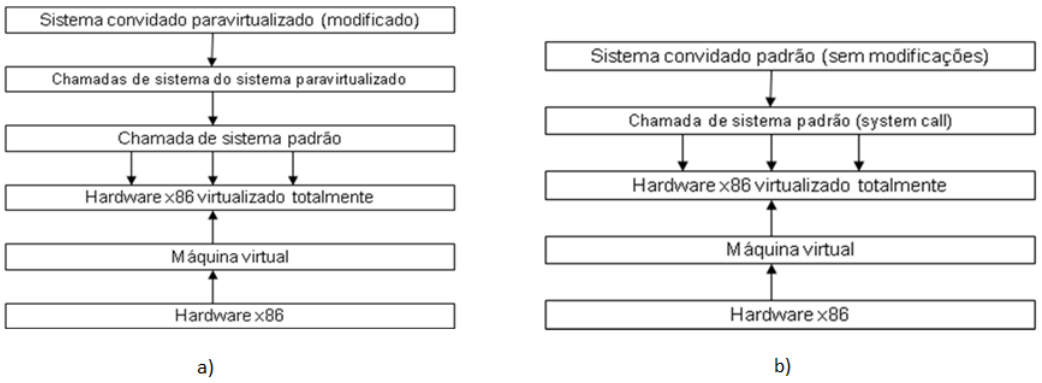

Figura 2.3: Técnicas de virtualização: a) Paravirtualização, b) Virtualização total. Adaptada de Laureano [23].

Devido ao fato de que cada VM executa em uma mesma arquitetura de hardware, percebe-se que a utilização de VMs junto à virtualização estão associadas ao modelo de negócio IaaS, ou seja, as VMs são um tipo de infraestrutura ou ambiente virtual providas para executar aplicações do usuário.

Dadas as definições de virtualização, várias questões importantes são levantadas, como alocar VMs em ambientes de hardware de modo a obter uma performance eficiente, um menor consumo de energia, uma minimização de recursos, diminuir o uso de rede, entre outros. Essa questão será tratada no Capítulo 3, onde discuti-se diferentes maneiras de se alocar VMs em

[4]http:/www.linux-kvm.org/page/Main_Page

uma infraestrutura de acordo com o objetivo em questão.

Capítulo 3

Alocação de Máquinas Virtuais

O problema de alocação de VMs em nuvem está relacionado ao processo de mapeamento de instâncias virtuais em máquinas físicas de acordo com regras de alocação. Vários trabalhos tem sido publicados recentemente na busca de otimizar algoritmos de alocação de VMs [32]. Muitos problemas são de ordem combinatorial e não possuem uma solução matemática ótima, e por essa razão algoritmos heurísticos são geralmente utilizados para solucionar esse problema. Apesar disso, muitos provedores de nuvem ainda encontram problemas na aplicação de um método eficiente para a alocação de VMs, que consideram consumo de energia de servidores, balanceamento do uso recursos, e alocação de VMs em grande escala [12].

No problema de alocação de VMs consideram-se um conjunto de máquinas físicas com diferentes recursos de hardware (processador, memória, meios de armazenamento e dispositivos), nas quais pode-se alocar diferentes VMs com diferentes requisitos de recursos. Considerando esse cenário, deve-se determinar o mapeamento de VMs para diferentes máquinas físicas, considerando um determinado objetivo [5].

Segundo Pires e Barán [32], podemos classificar os algoritmos de alocação em dois tipos. O primeiro tipo é aquele em que é possível que uma VM seja alocada em uma máquina física e realocada futuramente em outra, que é chamada de alocação dinâmica. O segundo tipo não permite que VMs realizem realocação para outras máquinas físicas, chamada de alocação estática. Essas duas definição também podem ser encontradas como *online* e *offline*, respectivamente.

3.1 Métricas

O centro de dados de um ambiente de nuvem possui um conjunto considerável de PMs e VMs. Existem diversos critérios que podem ser considerados para uma solução eficiente do problema de alocação de VMs, dependendo das políticas de controle e objetivos de otimização. Esses critérios podem mudar de tempo em tempo, isso implica na variedade de possíveis formulações do problema e diferentes objetivos para serem otimizados. Considerando que existem muitos trabalhos com várias funções objetivas semelhantes, os critérios utilizados para a classificação dos trabalhos foram divididos em cinco grupos de funções objetivas [32].

- Desempenho: com o objetivo de melhorar a utilização de processamento, os serviços de nuvem investem cada vez mais em virtualização e consolidação, buscando um meio para suprir a demanda de diversas aplicações diferentes executando simultaneamente em uma mesma plataforma. Nesse cenário que envolve grande número de serviços de nuvem, o desempenho é uma preocupação adicional e um desafio com alta complexidade.

- Eficiência energética: essa métrica está em consonância com tecnologias do tipo eco-eficiência, que tem o objetivo de reduzir o consumo de energia em ambientes de nuvem. Para isso são utilizadas técnicas como melhor controle de distruibuição de sistemas, sistemas refrigeradores eficientes, otimização de hardware e mecanismos de balanceamento de recursos. Como exemplo de aplicação temos um caso em que desligando um único servidor x86 de um centro de dados é possível economizar aproximadamente 400 dólares por ano [33].

- Rede: essa métrica está muito relacionada com o consumo de energia, tem o objetivo de diminuir a utilização de rede. A maioria dos trabalhos se baseiam na otimização de custo de comunicação de rede, migração em tempo real, métricas de rede (*delay*, tempo de transferência e acesso de dados, congestionamento, latência).

- Redução de custo: com o surgimento dos serviços de nuvem foi estabelecido um preço fixo de uso. Entretanto, muitos serviços de nuvem tem aplicado esquemas de preço dinâmicos onde o preço pago é proporcional a utilização do serviço. Outra técnica utilizada é o redimensionamento das VMs de acordo com o uso, sem prejudicar o desempenho, isso

pode gerar uma diminuição nos custos de implantação do serviço. Além disso, custos internos para o escalonamento de VMs são gerados devido a interferência e o sobrecarregamento do sistema causado entre várias VMs executando na mesma máquina física. Neste trabalho serão citados alguns trabalhos que tem como objetivo a redução do custo.

- Utilização de recursos: normalmente os ambientes de nuvem são compostos por multiplas máquinas físicas e recursos virtuais como CPU, memória, armazenamento, banda de rede e em alguns casos *Graphical Process Units* (GPU). Nesse contexto, uma eficiente e balanceada utilização desses recursos passa a ser uma importante questão. A maioria dos trabalhos focam em maximizar a utilização de recursos, outros trabalhos tem como objetivo o balanceamento de cada recurso [32].

3.2 Técnicas de Solução

Segundo Pires e Barán [32] existem diferentes técnicas propostas para a solução do problema de alocação de VMs. Essas técnicas podem ser classificadas em quatro tipos:

- Algoritmos Determinísticos: podem ser definidos como aqueles tipos de algoritmos onde em qualquer que seja o passo de execução exista apenas um próximo passo predeterminado. Algumas técnicas determinísticas clássicas são citadas como solução para a alocação de VMs, tais como programação por restrição, programação linear, programação linear inteira, programação linear mista, otimização pseudo-booleana e programação dinâmica.

- Heurística: esse tipo de técnica tem como objetivo a quantificação de proximidade a um determinado critério. Essa abordagem não é viável para solucionar problemas com um grande número de máquinas físicas e virtuais, pois é uma técnica baseada na busca exaustiva pela solução ótima do problema e isso faz com que o resultado possua um baixo desempenho e um custo computacional muito alto. Alguns algoritmos que utilizam essa técnica são: *First-Fit*, *First-Fit Decreasing*, *Best-Fit*, *Best-Fit Decreasing*, *Worst-Fit* e *Haviest-Fit*.

- Meta-Heurística: essa técnica de solução geralmente utiliza de um conhecimento histórico dos resultados anteriores adquiridos pelos algoritmos para se guiarem na otimização do

problema de alocação de VMs. Com essa técnica também é possível obter bons resutados em um tempo prático. Alguns exemplos de algoritmos que aplicam essa técnica são: os algoritmos meméticos, algoritmos genéticos, otimização por nuvem de partículas, busca pela vizinhança e *Simulated Annealing*.

- Algoritmos de aproximação: são algoritmos utilizados para encontrar uma solução aproximada, na maioria das vezes são associados a problemas considerados NP-difícil. Poucos trabalhos são publicados utilizando esse tipo de solução para otimizar o problema de alocação de VMs, já que as técnicas heurísticas e meta-heurísticas nos fornece uma qualidade boa o suficiente para a solução do problema.

3.3 Estado-da-Arte

Diversos trabalhos têm sido elaborados sobre o problema de alocação de VMs em ambientes de nuvem. Esta seção tem como objetivo apresentar alguns trabalhos da literatura escolhidos de acordo com a métrica aplicada e qual algoritmo foi utilizado para solucionar o problema de alocação de VMs. Ao todo foram selecionados 20 trabalhos com diferentes métricas e diferentes aplicações. Esses trabalhos serão comparados entre si utilizando a métrica e o tipo de algoritmo aplicado em cada trabalho [26]. Esse estudo compartivo é utilizada como base ao restante do trabalho, onde pretende-se escolher alguns algoritmos para serem implementados no simulador CloudSim [8].

Segundo o estudo realizado por López e Barán [32], ao se estudar a otimização do problema de alocação de VMs são encontrados três tipos de algoritmos.O primeiro tipo inclui os algoritmos que têm o objetivo de otimizar apenas uma métrica, ou seja, que possuem apenas um objetivo de otimização. Existem também algoritmos com vários objetivos de otimização que possuem vários critérios de otimização combinados em um só objetivo. Por fim os algoritmos que trabalham com múltiplos objetivos em várias funções objetivas.

O trabalho de López e Barán [31] é um exemplo de algoritmo envolvendo múltiplos objetivos aplicados em diferentes funções considerando o acordo de nível de serviço (*Service Level Agreement*), que busca minimizar o consumo de energia, minimizar o tráfego de rede e maximizar a redução de custo. Os autores propoem um novo algoritmo de multi-objetivos memético, onde a entrada é composta pelos recursos de cada máquina física junto aos recursos da rede e pe-

las VMs contendo seus requisitos de recursos. Algoritmos meméticos são um tipo de algorítmo genético que utiliza indivíduos que podem evoluir autonomamente acrescentando unidades de informação cultural [29].

O estudo realizado por Biran et al. [3] é um exemplo de algoritmo que possui apenas um objetivo. O trabalho propoe uma variação para o problema de otimização de alocação de VMs levando em consideração as características de rede. Para isso é proposto um algoritmo *Min Cut Ratio-Aware* que é utilizado para achar soluções de alocação para o tráfego na rede constante de serviços implantados, minizando a carga máxima para o corte de rede. Outro trabalho aplicado para minimizar o uso de rede é proposto por Trong e Hwang [37]. Os autores propoem um algoritmo chamado *Traffic and Power aware Virtual Machine Placement* (TPVMP) que combina as técnicas *traffic aware* que tem o objetivo de reduzir o custo de tráfego na rede e a técnica *energy aware* escolhe a máquina física destino para cada VM de modo a CPU usar o menor consumo possível de energia.

Outro algoritmo com múltiplos objetivos é apresentado no artigo [12]. Esse algoritmo se baseia no método de k-meios (*K-means*). O método é utilizado para realizar a quantização do vetor de recursos, onde para um parametro k divide-se um conjunto de objetos n em m servidores. O algoritmo visa otimizar a alocação de recursos e minimizar o consumo de energia.

Sarma et al. [34] desenvolveram um novo modelo de algoritmo com múltiplos objetivos, o qual foi criado pela combinação de algoritmos genéticos com o método de otimização da colônia de formigas (*Ant Colony Optimization*), visando melhorias na utilização de recursos e a eficiência energética. A técnica de otimização da colônia de formigas é uma técnica probabilística utilizada para solucionar problemas computacionais. O objetivo desta técnica é encontrar um caminho ótimo. Os algoritmos genéticos podem ser definidos como procedimentos probabilísticos de busca em larga escala envolvendo múltiplos estados [17].

Um estudo recente sobre a minimização de consumo de energia foi publicado por Sait e Shahid [33]. Neste artigo é apresentado uma técnica heurística não determinística iterativa conhecida como Evolução Simulada (*Simulated Evolution*) que combina melhoria iterativa a perturbação construtiva e busca sempre não ficar preso em mínimos locais, onde a busca é realizada por movimentos inteligentes utilizados para resolver o problema da alocação de VMs. Portanto, essa técnica tem o objetivo de encontrar uma solução quase ótima em um tempo

relativamente curto. Seu desempenho é semelhante a técnica *Simulated Annealing* citada na seção 3.2 como uma técnica meta-heurística.

Camati, Calsavara e Lima [9] propuseram uma aproximação para a solução do problema de alocação de VMs utilizando a técnica do problema de múltiplas mochilas multidimensional onde dado um conjunto inicial de VMs de diferentes tamanhos e valores, é retirado um subconjunto desse conjunto e alocado em um grupo de servidores com o objetivo de maximizar o uso dos recursos de cada servidor. Assim o trabalho possui como principais objetivos minimizar o consumo de energia e maximizar o uso de recursos.

Le et al. [24] estudaram o impacto da alocação de VMs em virtude da temperatura nos centro de dados. Com base nesse estudo propuseram uma política de distruibuição de carregamento que considera todos os custos com eletricidade e resfriamento, de modo que se aplique estratégias de resfriamento em servidores levando em consireção os picos de processamento. Portanto, o objetivo do trabalho é tanto minimizar o custo quanto o consumo de energia da estrutura como um todo.

Goudarzi e Pedram [18] apresentaram uma possível solução para o problema de eficiência energética na alocação de VMs. Eles utilizam um processo em que inicialmente são criadas cópias de VMs e só então usa-se programação dinâmica para procura de recursos para alocar as cópias. O método de programação dinâmica determina o número de cópias para cada VM e realiza o processo de alocação das mesmas. As cópias possuem os mesmos recursos que a VM original, são utilizadas para não deixar que um servidor fique ocioso, assim sempre há uso de processamento, consequentemente o sobrecarregamento dos servidores diminui, essa técnica pode reduzir o custo de energia em até 20%.

Erickson et al. [14] estudaram um possível modo de otimizar a desempenho de servidores conhecendo o tráfego e a topologia de rede. Eles introduzem um novo tipo de algoritmo chamado *Mixed-Integer Model* que explora as características das máquinas físicas e da rede, buscando uma nova alocação de uma VM que minimize o uso de processamento pelos servidores ou rede que estiverem sobrecarregados.

Gupta e Kalé [20] realizaram um estudo sobre um escalonador de computadores de alta performance e implementaram isso sobre o escalonador do *Open Stack*, um sistema de código aberto para a criação de nuvens privadas e pública. O objetivo do trabalho foi introduzir duas

novas técnicas para a otimização de computadores de alta perfomance, a *Topology Awareness* que busca alocar as VMs em máquinas físicas mais próximas possíveis umas das outras respeitando a topologia do servidor e a *Homogeneity* que garante o mesmo uso de processamento para cada aplicação, neste caso os autores selecionam k VMs e garante que cada uma use o mesmo tipo de processador. O principial objetivo dessas técnicas são minimizar o uso de rede e melhorar a performance de processamento.

Gupta, Milojicic e Balle [19] também implementam um escalonador de computadores de alta performance sobre a plataforma *Open Stack*. O escalonador é implementado usando como base o método heurístico *Multi-dimensional Online Bin Packing* com o objetivo de selecionar o servidor para alocar o serviço considerando os recursos de todas as dimensões (CPU, memória, armazenamento, rede) aplicada a diversas máquinas físicas. Esse trabalho é uma extensão do trabalho anterior e possui os mesmos objetivos principais que são minimizar o uso de rede e melhorar o desempenho de processamento.

É comum que muitos servidores e grupos de servidores estejam conectados por múltiplos *switches* e múltiplas VMs executem nesses grupos de servidores. Como as VMs podem migrar entre os servidores isso pode resultar em um desequilíbrio no uso de rede, pois, alguns *switches* podem estar sobrecarregados e outros podem ficar ociosos. Anthony [1] que registrou a patente US 2015/0074262 A1 pela VMWare, descreve uma técnica em que inicialmente é identificada uma rede com menos uso do que um limiar do sistema e seleciona os servidores presentes nessa rede para alocarem VMs. Um segundo passo da técnica é analisar se o tráfego de uma rede está acima do limiar do sistema, caso aconteça é iniciado a migração das VMs dos servidores desse rede para servidores de outro rede com tráfego a baixo do limiar do sistema.

Li, Zhang e Wu [25] propuseram um método de alocação de VMs *offline* que simula a migração de VMs entre máquinas físicas e um outro método de alocação de VMs *online* com migração real. O método possui uma aproximação heurística em que uma VM é diretamente alocada na melhor máquina física que possua recursos para recebe-lá. Como o objetivo do trabalho é maximizar o uso de recursos, eles propuseram também um algoritmo chamado *migration-based VM placement* onde é verificada uma VM em uma máquina física que possa ser migrada para outra máquina física melhor, de modo a liberar recuaros a uma nova VM.

Com o objetivo de minimizar o número de servidores Jin et al. [21] formularam um método

chamado *Max-Min Multidimensional Stochastic Bin Packing*, onde a ideia básica é aumentar a relação mínima de utilização de todos os recursos de um servidor, para satisfazer a demanda determinística e estocástica de VMs. A ideia do algoritmo é que para cada servidor recentemente ligado, é encontrado um conjunto de VMs que maximize a utilização mínima de recursos por servidores, em consequência o número de servidores ligados necessários diminui.

Bellur, Rao e Kumar [2] introduziram duas aproximações para a otimização de alocação de VMs com o objetivo de minimizar o número de máquinas físicas utilizadas em ambientes de nuvem. Inicialmente os autores utilizam o método *Vector Bin Packing*, formulado como uma programação linear. Esse método é semelhante ao problema da mochila, pois, um vetor nesse caso é considerado como uma requisição de recursos de uma VM, que possui n dimensões interpretados como recursos computacionais, assim como cada máquina física é um vetor que possui d dimensões referentes aos recursos computacionais. O objetivo principal é minimizar cada dimensão de recursos. A segunda aproximação utiliza o problema de maximização dual que é um tipo de programação não linear aplicada na primeira aproximação.

Shi et al. [35] propuseram duas aproximações para a solução ao problema de alocação de VMs. A primeira é a aproximação pela programação inteira linear, esse modelo produz alocações ótimas de requisições de VMs. Já a segunda aproximação classifica as requisições de VMs em diferentes categorias satisfazendo o algoritmo *First-Fit Decreasing* para o problema conhecido como *multidimensional vector bin packing* que é um problema heurístico utilizado para reduzir as soluções que não são ideais. Ambas as aproximações tem como principal objetivo a redução de custo.

Li et al. [27] estudaram diversos métodos conhecidos na literatura para achar um custo ótimo de implantação de servidores utilizando cenários de preços dinâmicos, como o método *Round-Robin*, *Greedy*, *Random*, a permutação e o *First-fit*. Depois de analisada as qualidades e defeitos de cada algoritmo, foi possível analisar que uma aproximação por busca exaustiva é boa para achar solução ótimas, mas sua execução são normalmente demoradas. Em contrapartida a aproximação utilizando *Greedy* possui uma rápida execução e soluções aceitáveis. Portanto, essa publicação se encaixa na métrica de redução de custo.

Chaisiri, Lee e Niyato [11] introduziram um algoritmo chamado alocação ótima de VMs, o objetivo do algoritmo é minimizar o custo total para comprar uma reserva e planos sobre

demanda do provedor de recursos. Esse algoritmo é utilizado no modelo de serviço IaaS, para descobrir o número de VMs ideais em cada ambiente de nuvem, levando em consideração uma otimização entre a incerteza de futuras demandas e o preço dos recursos. A solução otima é obtida através da programação inteira estocástica.

Lago, Madeira e Bittencourt [22] desenvolveram um algoritmo de alocação de VMs, cujo o principal objetivo é a minimização do consumo de energia para a execução de tarefas em *datacenters*. Para isso o algoritmo realiza o desligamento de máquinas físicas subutilizados e a migração de cargas dessas máquinas que operam a baixo de um nível definido. A escolha de qual máquina física irá receber as tarefas transferidas é baseada no conceito de maior eficiência de energia entre as máquinas físicas da estrutura de computação em nuvem, que é dada pela relação do número de milhões de execuções realizadas por segundo pela energia consumida de cada máquina física.

Após a realização do estado-da-arte, utilizamos as características dos trabalhos para montar a Tabela 3.1, que contém o tipo (mono-objetivo ou multi-objetivo), as métricas (desempenho, rede,eficiência energética, utilização de recursos e redução de custo) e as técnicas de solução (algoritmos determinísticos, heurística, meta-heurística e algoritmos de aproximação). Com as métricas de cada um dos trabalhos definidas é possível observar quantos trabalhos são destinados a cada métrica na Figura 3.1.

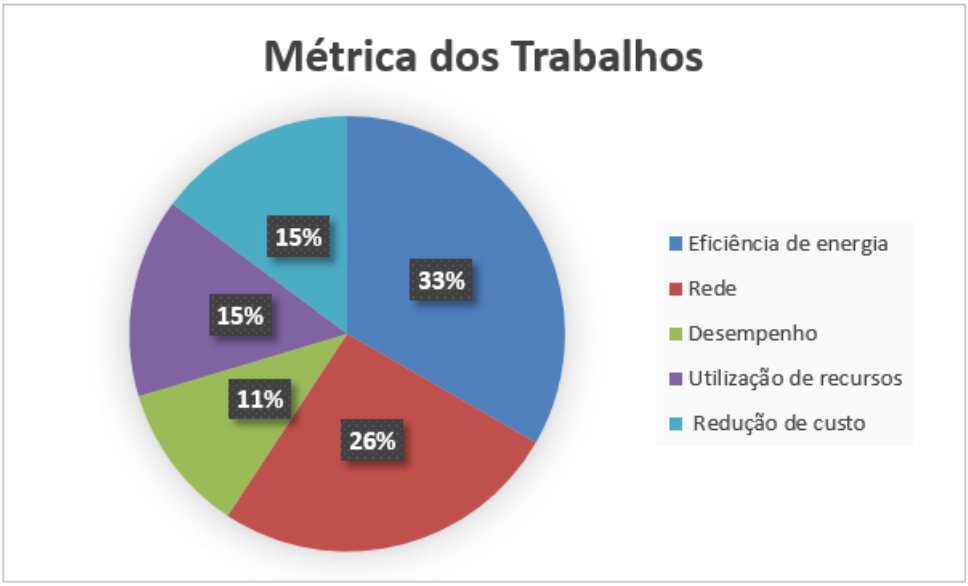

Figura 3.1: Gráfico representando a porcentagem das métricas encontradas nos trabalhos.

Tabela 3.1: Tabela comparativa de soluções para alocação de VMs.

Refência bibliográfica	Tipo	Métricas	Técnica de Solução
Lópes e Barán [31]	multi-objetivo	eficiência energética, rede e redução de custo	meta-heurística
Biran et al. [3]	mono-objetivo	rede	heurística
Trong e Hwang [37]	multi-objetivo	rede e eficiência energética	meta-heurística
Chen et al. [12]	multi-objetivo	eficiência energética, desempenho e utilização de recursos	heurística
Sarma et al. [34]	multi-objetivo	eficiência energética e utilização de recursos	meta-heurística
Sait e Shahid [33]	mono-objetivo	eficiência energética	meta-heurística
Camati, Calsavara e Lima [9]	multi-objetivo	eficiência energética e utilização de recursos	meta-heurística
Le et al. [24]	multi-objetivo	redução de custo e eficiência energética	heurística
Goudarzi e Pedram [18]	mono-objetivo	eficiência energética	heurística
Erickson et al. [14]	mono-objetivo	desempenho e rede	heurística
Gupta e Kalé [20]	mono-objetivo	desempenho e rede	heurística
Gupta, Milojicic e Balle [19]	multi-objetivo	desempenho e rede	heurística
Anthony [1]	mono-objetivo	rede	heurística
Li, Zhang e Wu [25]	mono-objetivo	utilização de recursos	heurística
Jin et al. [21]	mono-objetivo	utilização de recursos	meta-heurística
Bellur, Rao e Kumar [2]	mono-objetivo	utilização de recursos	meta-heurística
Shi et al. [35]	mono-objetivo	redução de custo	heurística
Li et al. [27]	mono-objetivo	redução de custo	heurística
Chaisiri, Lee e Niyato [11]	mono-objetivo	redução de custo	meta-heurística
Lago, Madeira e Bittencourt [22]	mono-objetivo	eficiência energética	heurística

Capítulo 4

Framework para o escalonamento de VMs em nuvem

Concluídos os estudos dos artigos sobre a alocação de VMs, pode-se verificar que existem muitos estudos e trabalhos que buscam melhorar o fornecimento de VMs por métricas diferentes entre diversas máquinas físicas de um ambiente de computação em nuvem. Nesse trabalho, o objetivo é criar um *framework* para a implementação desses algoritmos de alocação de VMs, de modo que fosse possível aplicar/testar esses algoritmos em um ambiente de simulação com máquinas físicas implantadas em *datacenters*. A ideia principal é que o *framework* possibilite a um pesquisador implementar e testar os algoritmo de alocação de VMs, em que, dado um conjunto de VMs e um conjunto de máquinas físicas, possa ser possível alocar essas VMs nas máquinas físicas testando as métricas utilizadas para o algoritmo.

O *framework* proposto é um ambiente de simulação, no qual pode-se configurar a quantidade de VMs utilizadas e suas configurações de hardware, juntamente pode-se configurar a infraestrutura física a ser utilizada para alocar as VMs. Pode-se ainda analisar algumas métricas, tais como o número de máquinas utilizadas, a energia total consumida, total de banda utilizada. Essas informações são úteis para o gestor da infraestrutura do ambiente de nuvem manter um acompanhamento do sistema, assim ele pode usar essas informações para tomar alguma ação e manter um acompanhamento do ambiente de nuvem, por exemplo verificar que como há um excedente de máquinas físicas sem utilização, seria possível desligar algumas máquinas físicas e economizar custos com energia e processamento, ou até mesmo realizar migrações para balanceamento de carga entre máquinas físicas pouco utilizadas.

4.1 Cloudsim

Para a implementação de um ambiente de simulação que execute algoritmos implementados pelo usuário utilizando a linguagem de progração Java, foi utilizada uma ferramenta de modelagem e simulação de ambientes de computação em nuvem chamada CloudSim [8].

O CloudSim é uma ferramenta extensível que nos permite modelar e simular sistemas de computação em nuvem. Essa ferramenta implementa componentes de um sistema de nuvem como *datacenters*, VMs, políticas para o fornecimento de recursos, tudo isso sob configurações do usuário. As políticas são a parte principal para esse trabalho, que podem ser implementadas por algoritmos genéricos e estendidas na execução da simulação da ferramenta [8].

A Figura 4.1 mostra o projeto de multi camadas do CloudSim e seus componentes arquiteturais. A ferramenta incluí um controle dedicado de interfaces para VMs, memória, armazenamento e largura de banda. As funcionalidades fundamentais que serão utilizados no *framework* como provisionamento de máquinas físicas para VMs, controle da execução da aplicação, monitoramento do estado dinâmico do sistema, são localizados na camada do CloudSim. Como o foco do trabalho está no estudo da eficiência de diferentes políticas de alocação de VMs para máquinas físicas, foi utilizado então o serviço de *VM provisioning*, que é um serviço realizado pela entidade Broker onde dado um determinado algoritmo de alocação de VM configurado no Datacenter, o Broker a utiliza para definir a máquina física em que uma VM requisitada deve ser alocada. A camada mais acima do CloudSim é a do código do usuário (*User Code*) que é responsável pelas entidades básicas da ferramenta como *hosts* (número de máquinas, memória, largura de banda, número de núcleos de processamento), em nível da aplicação são utilizados os *cloudlets* que são as tarefas designadas para cada VM com suas especificações de tamanho de armazenamento e número de núcleos de processamento utilizados. Ainda nessa camada são especificadas as VMs, o número de usuários, os *datacenters* e o *broker* que é responsável pelo provisionamento de VMs no *datacenter* [8].

4.2 Metodologia

O *framework* utiliza tanto a camada de usuário quanto a camada do próprio CloudSim, para a execução da simulação, deve-se definir previamente as características básicas das entidades,

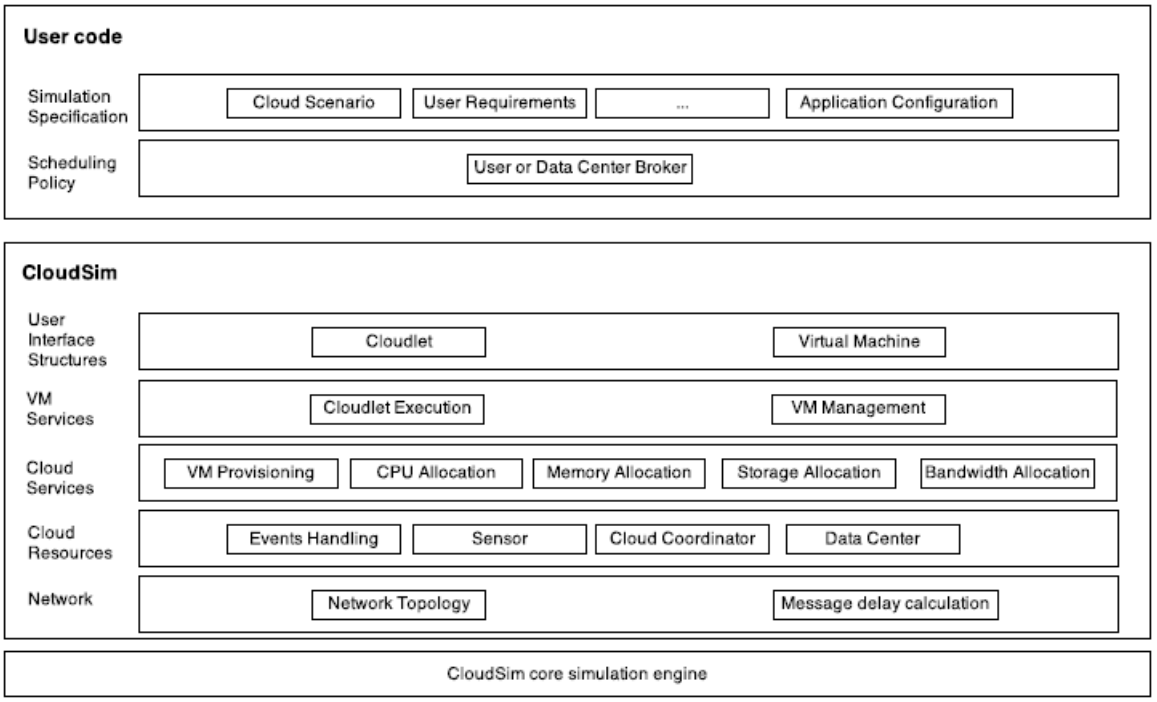

Figura 4.1: Arquitetura do CloudSim. Adaptada de Calheiros et al. [8].

como a quantidade de memória, armazenamento, núcleos de processamento, largura de banda, número de milhões de instruções executadas por segundo (MIPS) e o modelo energético de cada *host*. As características de cada VM a ser alocada é semelhante às características dos *hosts*, por convenção aloca-se apenas um cloudlet para cada VM. Considerando que o foco do trabalho concentra-se na alocação das VMs e não nas múltiplas tarefas executadas sobre uma mesma VM, assim como foi utilizado apenas um *datacenter* e um usuário no ambiente de simulação.

Na Figura 4.2 é possível identificar as principais classes do CloudSim que foram utilizadas na implementação do ambiente de simulação. O Diagrama de classe representa toda as classes utilizadas para a implementação do ambiente de simulação, seus respectivos atributos e funções utilizadas. As classes utilizadas foram a classe *Host* que representa as máquinas físicas, a classe *Datacenter* que representa o conjunto das máquinas físicas, a classe *Vm* que representa as máquinas virtuais, a classe *DatacenterCharacteristics* que representa as características do *datacenter*, a classe *Pe* que é o núcleo de processamento de cada *host* e a classe *Cloudlet* que executa uma certa carga em uma VM.

A classe principal para o *framework* é a classe *VmAllocationPolicy* responsável pela implementação do algoritmo de alocação de uma VM em um determinado *host*. Os algoritmos para

a alocação de VMs em máquinas físicas devem ser implementados em uma classe criada pelo usuário que implemente os métodos da classe *VmAllocationPolicy* passados por herança, nesta classe o algoritmo para alocação de uma VM em um determinado *host* deve ser implementado no método *allocateHostForVm* essa função é utilizada pela classe *DatacenterBroker* realizar o serviço de *VM provisioning* aos *hosts* situados na classe *Datacenter*. A entrada para o método é a VM que está na fila para ser alocada e o *host* para a alocação da VM. Esse método retorna um valor lógico que indica a possibilidade de caso seja possível essa alocação retorna valor verdadeiro caso contrário retorna valor falso. O processo é repetido até que todas as VMs sejam alocadas. Caso os *hosts* do ambiente de execução não consigam alocar todas as requisições de VMs, o próprio simulador emite erro de execução e a simulação é finalizada.

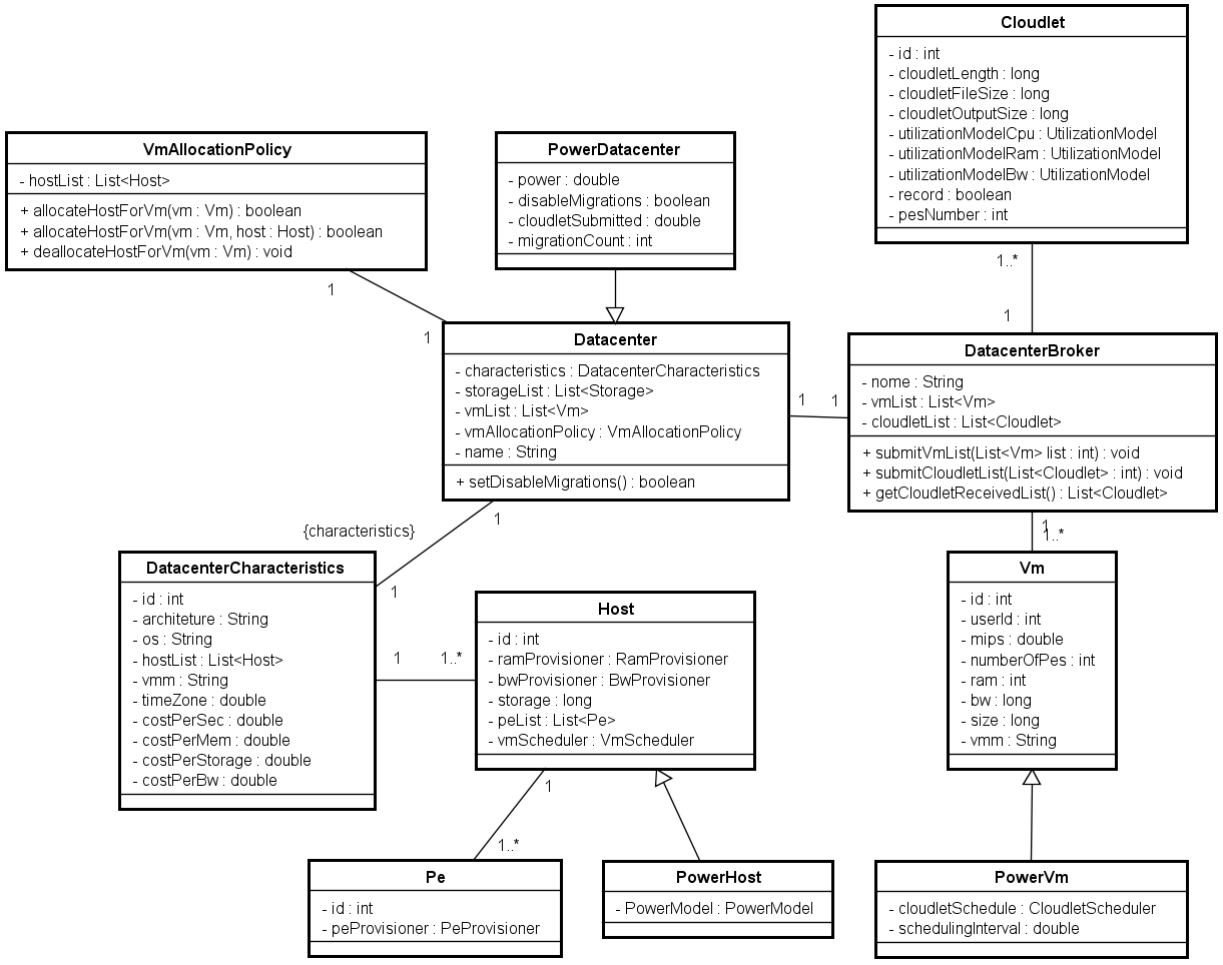

Figura 4.2: Diagrama de classe implementadas no *framework*.

O algoritmo para a alocação de VMs pode ser implementado em outro método na mesma

classe, de modo que o algoritmo implementado retorne o *host* definido para a alocação da VM dentre os *hosts* disponíveis. Para isso é necessário realizar toda a verificação se o *host* definido pode ou não alocar os recursos requisitados pelo VM. O objetivo é que essa classe do algoritmo implementado seja passada como parâmetro na criação do *datacenter*. Assim o algoritmo fica no nível de *datacenter* determinando em qual de suas máquinas físicas as requisições de recursos das VMs serão alocadas, que pode ser verificado na Figura 4.2.

4.3 Implementação

Como o objetivo do *framework* é viabilizar o teste de diferentes algoritmos de alocação de dados, utilizamos o algoritmo *FirstFit* para a alocação de VMs como exemplo para a implementação de um algoritmo de alocação de VMs no *framework* desenvolvido. Para a configuração do ambiente e implementação de um algoritmo de alocação de VM, pode ser aplicada a seguinte metodologia.

Inicialmente são definidas as características básicas das entidades principais do CloudSim para o ambiente de simulação encontradas no Algoritmo 4.1. Para os *hosts* são definidos quantos tipos diferentes de máquinas físicas serão utilizados na simulação (linha 13), a quantidade de milhões de instruções executadas por segundo por núcleo de processamento (linha 14), quantos núcleos de processamento a máquina física possui (linha 15), quanto de memória (linha 16), quantidade de largura de banda (linha 17) e a quatidade de armazenamento na unidade de megabytes (linha 18) e também é definido o modelo energético do *host* baseado em um modelo de máquinas física real (linha 20), que nesse caso de teste utilizamos o modelo HP ProLiant Ml110 G4. Da linha 29 a 32 são definidos o custo por segundo de processamento, o custo para uso do recurso de memória, o custo para armazenamento e o custo para uso do recurso de banda de rede.

Algoritmo 4.1: Código com as características básicas das VMs, *hosts*, *cloudlets* e dados da simulação.

```
1       public final static double SCHEDULING_INTERVAL = 300;
2       public final static double SIMULATION_LIMIT = 24 * 60 * 60;
3       public final static boolean ENABLE_OUTPUT = true;
4
5       public final static int VM_TYPES = 1;
6       public final static int[] VM_MIPS = {1000};
7       public final static int[] VM_PES = {2};
8       public final static int[] VM_RAM = {2048}; //2GB
9       public final static int VM_BW = 100000; // 100 Mbit/s
10      public final static int VM_SIZE = 2500; // 2.5 GB
11
12      public final static int NUMBER_OF_HOSTS = 15;
13      public final static int HOST_TYPES = 1;
14      public final static int[] HOST_MIPS = {2000};
15      public final static int[] HOST_PES = {4};
16      public final static int[] HOST_RAM = {16384}; // 16GB
17      public final static int HOST_BW = 1000000; // 1 Gbit/s
18      public final static int HOST_STORAGE = 1000000; // 1 TB
19
20      public final static PowerModel[] HOST_POWER = {
21         new PowerModelSpecPowerHpProLiantMl110G4Xeon3040()
22      };
23
24      public final static int CLOUDLET_LENGTH = 2500 *
25                           (int) SIMULATION_LIMIT;
26      public final static int CLOUDLET_PES = 1;
27      public final static int NUMBER_0F_CLOUDLETS = 20;
28
29      public final static double cost = 3.0;
30      public final static double costPerMem = 0.05;
31      public final static double costPerStorage = 0.001;
32      public final static double costPerBw = 0.0;
```

Para os cloudlets são definidos apenas um tamanho estático que depende do limite de tempo da execução da simulação (linha 24), o número de núcleos de processamento utilizados para a execução da tarefa. O número de cloudlets define o número de VMs resultando um cloudlet para cada VM. Além de algumas características da simulação como o intervalo de execução da simulação medido na unidade de segundos (linha 1), o tempo limite da execução da simulação também medido na unidade de segundos (linha 2) e uma variável definindo o valor lógico da geração de *log* para verdadeiro (linha 3).

A classe que implementa a política de alocação de VMs deve implementar os métodos her-

dados da classe *VmAllocationPolicy*, como mostra a Figura 4.2. Os métodos são *allocateHostForVm* (linha 4) e *deallocateHostForVm* (linha 38) encontradas no Algoritmo 4.2. Pode-se observar que existem dois métodos chamados *allocateHostForVm* (linha 4 e 19). Por convenção do simulador, a classe *DatacenterBroker* utiliza o método que possui somente a VM como parâmetro (linha 4) para a aplicação do *VM provisioning* com o objetivo de alocar a VM em um dos *hosts* disponíveis. Portanto, nos algoritmos implementados utilizou-se o método *allocateHostForVm* para os parâmetros de VM e *host*, já que é necessário definir o *host* que irá alocar a Vm em análise. No caso do algoritmo *FirstFit* o método *findHostForVm* encontra o primeiro *host* que consiga alocar a VM passada como parâmetro. De modo semelhante podem ser implementados diferentes algoritmos de alocação de VMs seguindo as configurações citadas.

Algoritmo 4.2: Código da classe FirstFitAllocationPolicy.

```
1  public class FirstFitAllocationPolicy extends
       PowerVmAllocationPolicyAbstract {
2
3      @Override
4      public boolean allocateHostForVm(Vm vm) {
5       return allocateHostForVm(vm, findHostForVm(vm));
6    }
7
8      public PowerHost findHostForVm(Vm vm) {
9
10        for (PowerHost host : this.<PowerHost>getHostList()) {
11           if (host.isSuitableForVm(vm) &&
                (getFreePes().get(host.getId())>0)) {
12              return host;
13           }
14        }
15        return null;
16     }
17
18     @Override
19     public boolean allocateHostForVm(Vm vm, Host host) {
20        if (host.vmCreate(vm)) {
21           getVmTable().put(vm.getUid(), host);
22
23           int requiredPes = vm.getNumberOfPes();
24           int idx = getHostList().indexOf(host);
25           getUsedPes().put(vm.getUid(), requiredPes);
26           getFreePes().set(idx, getFreePes().get(idx) -
                requiredPes);
27
28           Log.formatLine(
```

```
29                  "%.2f: VM #" + vm.getId() + " has been allocated
                       to the host #" + host.getId(),
30                  CloudSim.clock());
31              return true;
32          }
33
34          return false;
35      }
36
37      @Override
38      public void deallocateHostForVm(Vm vm) {
39      }
40  }
```

Com as características pré-definidas e a classe do algoritmo que herda os métodos da classe *VmAllocationPolicy* implementada, deve-se criar um objeto da classe que implementa o algoritmo de alocação de VMs e setá-lo como parâmetro para a classe do *Datacenter* no seu ambiente de simulação como mostra o Algoritmo 4.3 (linha 5). Feito isso basta executar o ambiente de simulação que seria a classe principal do projeto. A saída exibe um relatório em log que pode ser passado para um arquivo de texto utilizando um parser, esse log contém as informações de números de máquinas desligadas, o número de máquinas migradas durante a simulação, o consumo energético total de todas as máquinas físicas, o tempo total da simulação e o acompanhamento do consumo energético dos *hosts* nos intervalos da simulação criados anteriormente.

Algoritmo 4.3: Código da criação do objeto *datacenter*

```
1  PowerDatacenter datacenter = (PowerDatacenter) createDatacenter(
2                  "Datacenter",
3                  PowerDatacenter.class,
4                  hostList,
5                  new FirstFitAllocationPolicy(hostList));
```

Capítulo 5

Simulação

Com o objetivo de realizar análises sobre a ferramenta CloudSim e para validar a implementação do *framework*, foram implementados 7 algoritmos de alocação de VMs junto ao ambiente de execução descritos a seguir:

- *FirstFit*: o primeiro e mais simples algoritmo implementado, ao receber a requisição de alocação de uma VM em um *host*, apenas seleciona o primeiro *host* que possa comportar a VM passada por parâmetro.

- *BestFit*: o objetivo da política é encontrar o *host* que ao receber a requisição da VM utilize a maior quantidade possível de recursos que os demais *hosts*, de modo que sobre a menor quantidade de recursos possível sem uso. Como os recursos de uma VM podem variar, esse algoritmo foi dividido entre *BestFitCPU* que foca no recurso de núcleo de processamento e o *BestFitRam* que foca no recurso da utilização da memória RAM.

- *WorstFit*: é o inverso da política do *BestFit*, assim o objetivo do algoritmo é encontrar um *host* que ao receber a requisição de VM utilize a menor quantidade possível de recursos, fazendo com que sobre a maior quantidade de recursos possível entre os *hosts*. Assim como no algoritmo do *BestFit* o *WorstFit* também foi dividido entre em dois focos de recursos diferentes o *WorstFitCPU* e o *WorstFitRAM*.

- *RandomFit*: escolhe o *host* em que a VM será alocada de forma aleatória, o *host* escolhido é testado para avaliar se é possível ou não comportar a VM.

- *Lago*: o algoritmo testa dinamicamente o processamento necessário para alocar a VM e o principal objetivo é minimizar o consumo de energia em tempo real que pode ser en-

contrado na Figura 5.1. Para cada VM a ser alocada são testados todos os *hosts*, de modo a definir qual irá alocar a VM (linha 3). A primeira condição do algoritmo é checar se o *host* possui uma quantidade suficiente de MIPS disponível para a VM ser alocada (linha 5). Para cada *host* que possui a capacidade de MIPS suficiente é então calculada a sua eficiência (linha 6), que é medida pela quantidade total de MIPS dividida pela capacidade máxima de energia da máquina física. O algoritmo escolhe o *host* que tiver a maior eficiência entre os demais (linha 7), caso ocorra um empate é escolhido a máquina física que consuma menos energia (linha 13), caso ainda ocorra um empate entre os menores consumos de energia é escolhido o *host* que tiver o maior uso de CPU (linha 16), caso ainda assim ocorra um empate o *host* com maior poder de processamento (MIPS) é escolhido como a máquina física que irá receber a requisição da VM (linha 19) [22].

```
Lago Allocator Algorithm: findHostForVm(vm)
1.  bestEfficiency = -∞
2.  bestHost = null
3.  foreach host in datacenter do
4.      utilization = getCurrentMIPSUtilization(host) +
        getMIPS(vm)
5.      if (utilization < getMIPS(host)) then
6.          efficiency = getMIPS(host) / getMaximumPower(host)
7.          if (efficiency > bestEfficiency) then
8.              bestEfficiency = efficiency
9.              bestHost = host
10.         else if ( efficiency == bestEfficiency) then
11.             pw_vm_at_host = getPower(bestHost) +
                getPowerAfterAllocation(host, vm)
12.             pw_vm_at_bestHost = getPower(host) +
                getPowerAfterAllocation(bestHost, vm)
13.             if (pw_vm_at_host < pw_vm_at_bestHost) then
14.                 bestHost = host;
15.             else if (pw_vm_at_host == pw_vm_at_bestHost)
                then
16.                 if (getUtilizationOfCpu(host) >
                    getUtilizationOfCpu(bestHost)) then
17.                     bestHost = host
18.                 else if (getUtilizationOfCpu(host) ==
                    getUtilizationOfCpu(bestHost)) then
19.                     if (getTotalMips(host) >
                        getTotalMips(allocatedHost)) then
20.                         bestHost = host
21.                     end if
22.                 end if
23.             end if
24.         end if
25.     end if
26. end foreach
27. return bestHost
```

Figura 5.1: Algoritmo de alocação Lago. Adaptada de Lago, Madeira e Bittencourt [22].

5.1 Cenários

Os testes foram realizados utilizando três modelos de características de VMs e *hosts* diferentes. Cada um dos algoritmos foi executado para cada um dos modelos, ainda cada um desses testes foram divididos em três tamanhos de *datacenters*, o que da um total de 63 testes executados no *framework*. O teste em *datacenters* de tamanho pequeno utiliza de 15 *hosts*, 20 VMs e como foi citado no capítulo anterior por convenção utilizamos um *cloudlet* para cada VM. O tamanho médio de *datacenter* utiliza 180 *hosts* e 250 VMs, e o tamanho grande utiliza 2000 *hosts* e 3000 VMs.

Algumas características são comuns entre os diferentes modelos de testes, todos os *hosts* utilizam 1 Gbit/s de largura de banda, 1 TB de armazenamento em disco, sistema operacional Linux e são baseados no modelo energético do HP ProLiant Ml110 G4 Xeon 3040[1] implementado no CloudSim. Todas as VMs utilizam 100 Mbit/s da largura de banda das máquinas físicas, ocupam um tamanho em disco de 2,5 GB e utilizam arquitetura de sistema operacional x86.

O primeiro modelo é caracterizado por ser um modelo homogêneo, portanto todas as VMs e todos os *hosts* possuem as mesmas características. Os *hosts* possuem 16 GB de memória RAM e 4 núcleos de processamento de 2000 MIPS cada. Já as requisições das VMs possuem 2 núcleos de processamento de 1000 MIPS cada e 2 GB de memória RAM cada. O segundo modelo utiliza duas configurações de *hosts* caracterizado como semi-heterogêneo já que as requisições de VMs ainda permanecem iguais a do primeiro modelo, uma possui 4 núcleos de processamento de 2000 MIPS cada e 16 GB de memória RAM como no primeiro modelo e o segundo tipo de *host* possui 2 núcleos de processamento de 2000 MIPS cada e 2 GB de memória RAM. O terceiro modelo é caracterizado por ser um modelo heterogêneo, em que foi utilizado os mesmos dois tipos de *hosts* do segundo modelo e as requisições de VMs foram divididas em três modelos, uma possui 4 GB de memória RAM e 4 núcleos de processamento de 600 MIPS cada, a segunda possui 2 núcleos de processamento de 300 MIPS cada e 2 GB de memória RAM e o terceiro tipo de VM tem 1 núcleos de processamento de 200 MIPS cada e 2 GB de memória RAM. As características citadas podem ser mais facilmente visualizadas nas Tabelas 5.1 e 5.2.

Nas baterias de alguns modelos de teste são utilizados diferentes tipos de *hosts* e VMs, assim o número total de *hosts* e VMs definidas pelo tamanho de cada cenário são divididas

[1] https://h10057.www1.hp.com/ecomcat/hpcatalog/specs/provisioner/05/432126-031.htm

proporcionalmente entre os tipos dos modelos, como mostra a Tabela 5.3.

Tabela 5.1: Tipos de máquinas físicas utilizadas nos modelos de simulação.

Modelo	Tipo	MIPS	PES	RAM	Largura de Banda	Armazenamento
Homogêneo	1	2000	4	16 GB	1 Gbit/s	1 TB
Semi-Heterogêneo	1	2000	4	16 GB	1 Gbit/s	1 TB
	2	1000	2	16 GB	1 Gbit/s	1 TB
Heterogêneo	1	2000	4	16 GB	1 Gbit/s	1 TB
	2	1000	2	16 GB	1 Gbit/s	1 TB

Tabela 5.2: Tipos de VMs utilizadas nos modelos de simulação.

Modelo	Tipo	MIPS	PES	RAM	Largura de Banda	Armazenamento
Homogêneo	1	1000	2	2 GB	100 Mbit/s	2,5 GB
Semi-Heterogêneo	1	1000	2	2 GB	100 Mbit/s	2,5 GB
Heterogêneo	1	600	4	4 GB	100 Mbit/s	2,5 GB
	2	300	2	2 GB	100 Mbit/s	2,5 GB
	3	200	1	2 GB	100 Mbit/s	2,5 GB

Tabela 5.3: Quantidades de *host* e VM para cada tamanho em cada um dos modelos utilizados.

Tamanho	Homogêneo		Semi-Heterogêneo			Heterogêneo				
	Host1	VM1	Host1	Host2	VM1	Host1	Host2	VM1	VM2	VM3
Pequeno	15	20	8	7	15	8	7	5	5	5
Médio	180	250	90	90	250	90	90	84	83	83
Grande	2000	2500	1000	1000	2500	1000	1000	834	833	833

5.2 Resultados

Dadas as configurações de cada modelo de teste, foram implementados e executados todos os cenários apresentados na seção anterior. Para cada teste foram coletados o consumo energé-

tico em kWh de todas as máquinas físicas do *datacenter* e foi também coletado o número de máquinas físicas utilizadas em cada execução para cada algoritmo de alocação de VMs. Ainda é importante ressaltar que os teste são totalmente determinísticos, ou seja, no início da execução é criada a lista de requisição de VMs e logo após são selecionados os *hosts* para cada uma das VMs. Como ambas as VMs e os *hosts* são representados por uma lista, o número de cada uma das entidades para cada tipo seguindo a Tabela 5.3, são ordenados na lista em sequência pelo tipo de *host* ou VM, por exemplo, para o teste do modelo heterogêneo de tamanho médio as primeiras 84 VMs são do tipo VM1, as próximas 83 VMs são do tipo VM2 e as últimas 83 VMs são do tipo VM3, o mesmo acontece para os *hosts* os primeiros 1000 *hosts* são do tipo Host1 e os últimos 1000 *hosts* são do tipo Host2.

Na Figura 5.2 se encontra o resultado das execuções da bateria de teste para o modelo mais simples, o modelo homogêneo, envolvendo um *datacenter* de tamanho pequeno, médio e grande. Pode-se verificar que os algoritmos *FirstFit* e as variações do *BestFit* foram bem satisfatórios em relação aos demais algoritmos para um ambiente com requisições de VMs iguais e máquinas físicas com a mesma configuração, podendo chegar a uma redução de consumo energético e número de *hosts* utilizados em aproximadamente 40% em relação ao *WorstFit* e o algoritmo *Lago*. Pode-se observar que os resultados entre as variações do WorstFit e do BestFit dão o mesmo resultado entre si, isso acontece uma vez que nesse modelo são utilizados apenas um tipo de VM e um tipo de *host*. E ainda pode-se notar que para esse modelo de teste o tamanho do teste não demonstrou uma grande alteração na eficiência dos algoritmos. Pode-se ainda constatar como a ordem de distribuição das requisições de VMs e *hosts* influenciam diretamente na forma em que as VMs serão alocadas, onde o *WorstFit* que sempre escolhe o *host* que possui o maior recurso disponível e o *RandomFit* que escolhe os *hosts* aleatoriamente resultaram em uma eficiência ruim se comparado ao *FirstFit* que escolhe o próximo *host* disponível que pode alocar a VM.

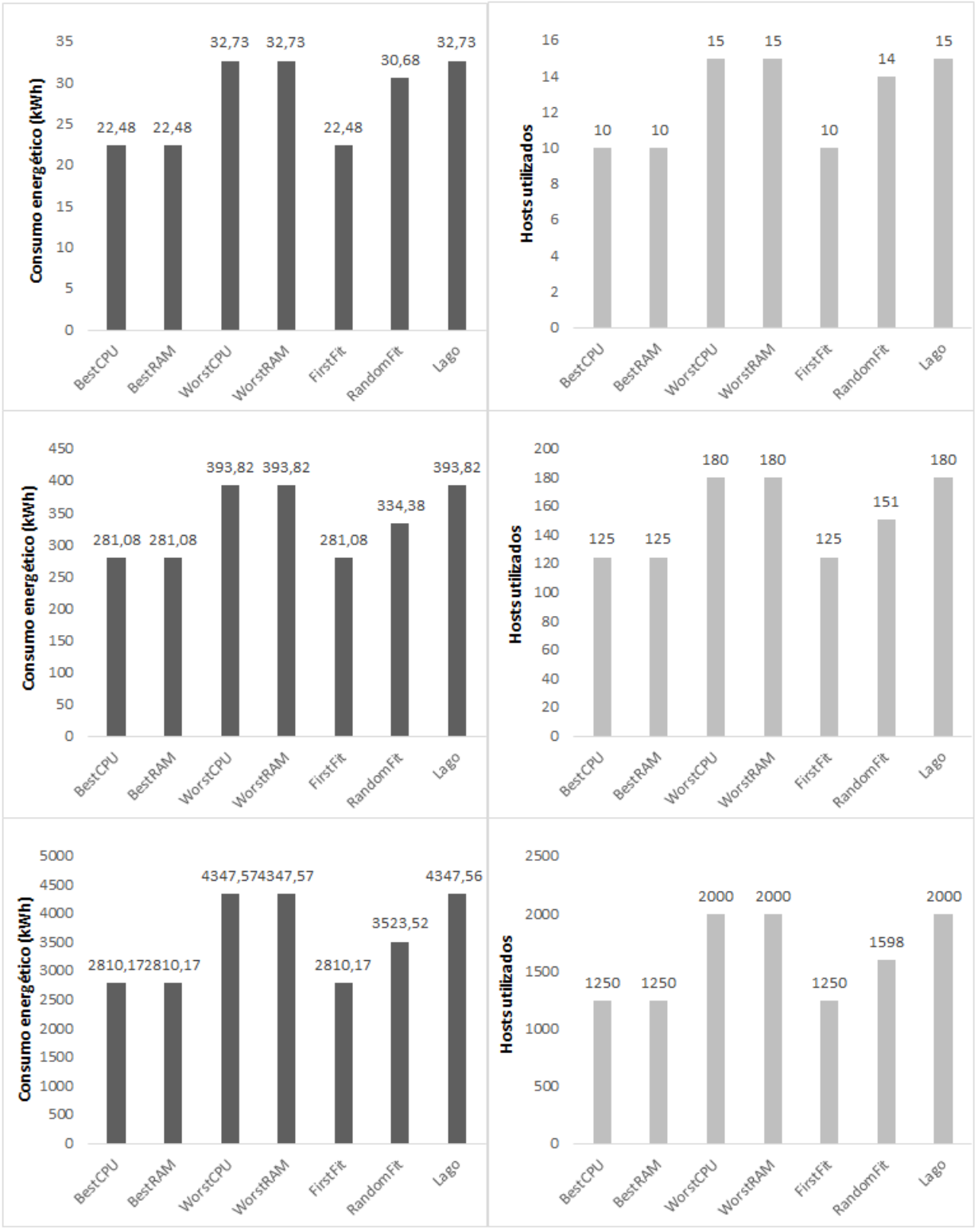

Figura 5.2: Resultados da simulação do modelo homogêneo para os tamanhos *datacenter* pequeno, médio e grande.

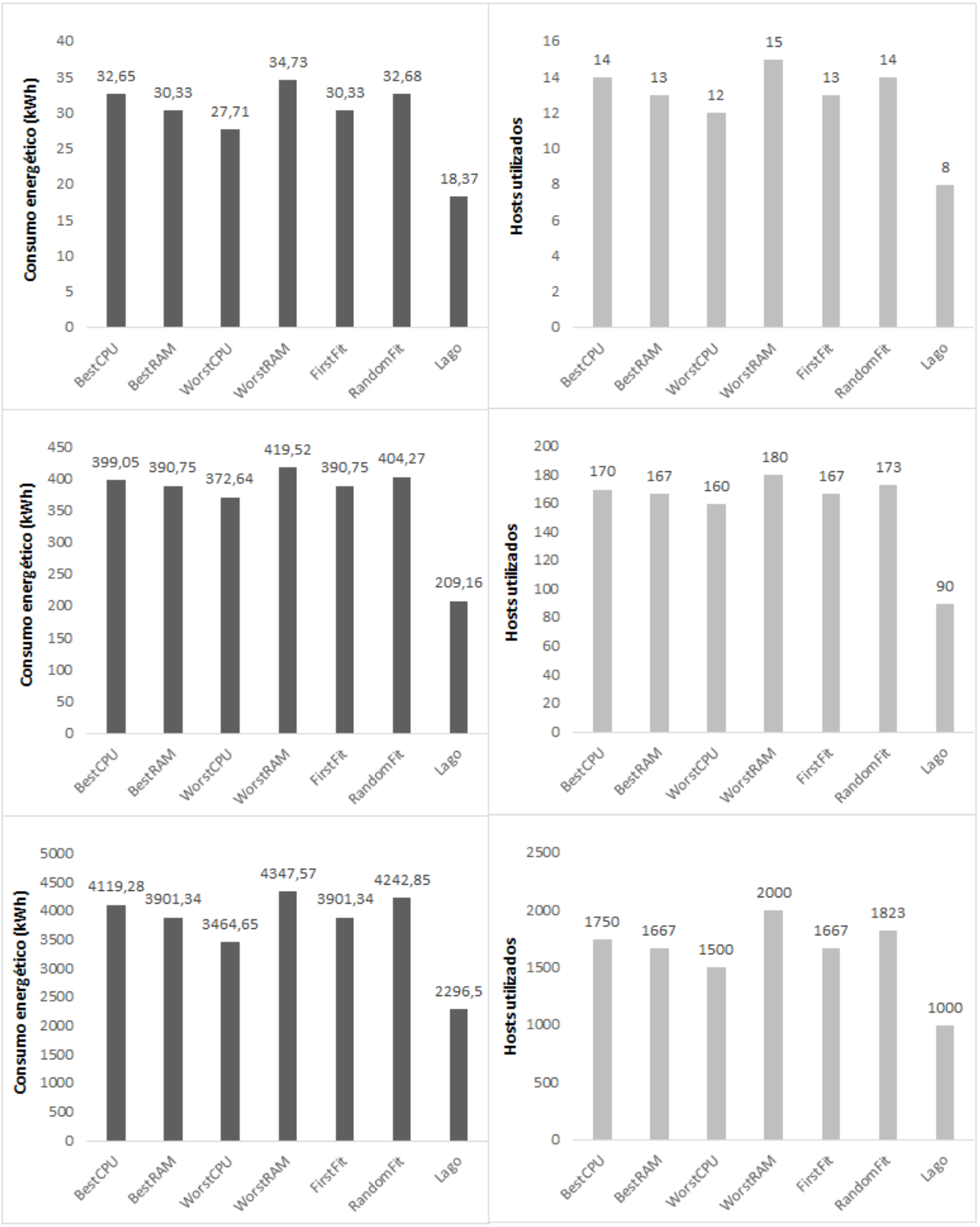

Figura 5.3: Resultados da simulação do modelo semi-heterogêneo para os tamanhos *datacenter* pequeno, médio e grande.

Analisando a Figura 5.3 é possível observar como o tamanho dos *datacenters* para um modelo semi-heterogêneo do ambiente de simulação implica diretamente nos algoritmos de alocação de recursos das VMs. Observa-se que nesse modelo, o algoritmo *Lago* obteve um resultado muito superior que os demais algoritmos com uma eficiência de até 100% se comparado ao resultado do *WorstFit*, reduzindo significamente o consumo energético. Além disso, o algoritmo *WorstFitCPU* que havia sido o algoritmo menos eficiente na Figura 5.3, obteve um desempenho mais eficientemente energético que os algoritmos *FirstFit*, *BestFit* e o *RandomFit* para esse modelo. O mesmo obteve ainda uma eficiência de cerca de 25% a mais que o *WorstFitRAM*, devido ao fato de que os *hosts* possuem uma diferença em número de núcleos de processamento e o *WorstFitCPU* prioriza a escolha de um determinado *host* de modo que com a alocação da VM sobre um maior número possível de núcleos de processamento. Portanto são alocadas mais VMs em um mesmo *host* diminuindo o número de *hosts* utilizados no algoritmo *WorstFitCPU* que no caso do *WorstFitRAM*, onde os *hosts* não possuem diferença em memória RAM, consequentemente o consumo energético também é diminuído.

Na Figura 5.4 pode-se notar que a colunas do algoritmo *WorstFitCPU* estão zeradas ao se executar alguns testes para o modelo heterogêneo, com diferentes tipos de máquinas físicas e VMs. Neste caso, a ferramenta CloudSim retornou erro na alocação das VMs e interrompeu as execução ds teste devido ao fato de que para aquele ambiente de simulação não foi possível alocar todas as VMs em máquinas físicas. Assim é possível verificar que a ordem de distribuição das VMs nos *hosts* é muito importante, e considerando que esse algoritmo não possui migração entre máquinas físicas, a ordem com que as VMs foram alocadas fez com que não houvessem *hosts* suficientes para a alocação de todas as VMs. Portanto, o algoritmo *WorstFitCPU* não obteve sucesso na alocação das VMs para um cenário de modelo heterogêneo de tamanhos pequeno e médio. Mas para um tamanho grande a execução do teste ocorreu normalmente, e ainda superou a eficiência dos demais algoritmos implementados, exceto para o algoritmo *Lago*. Assim como nos resultados da Figura 5.3 o algoritmo *Lago* mostrou-se muito mais eficiente energeticamente que os demais algoritmos de alocação de VMs, essa eficiência energética pode variar em cerca de 65% até 100% em relação aos demais algoritmos.

Pode-se observar que há uma relação entre as duas métricas dos gráficos do consumo energético e o número de *hosts* utilizados em cada um dos modelos de teste. Isso ocorre por que

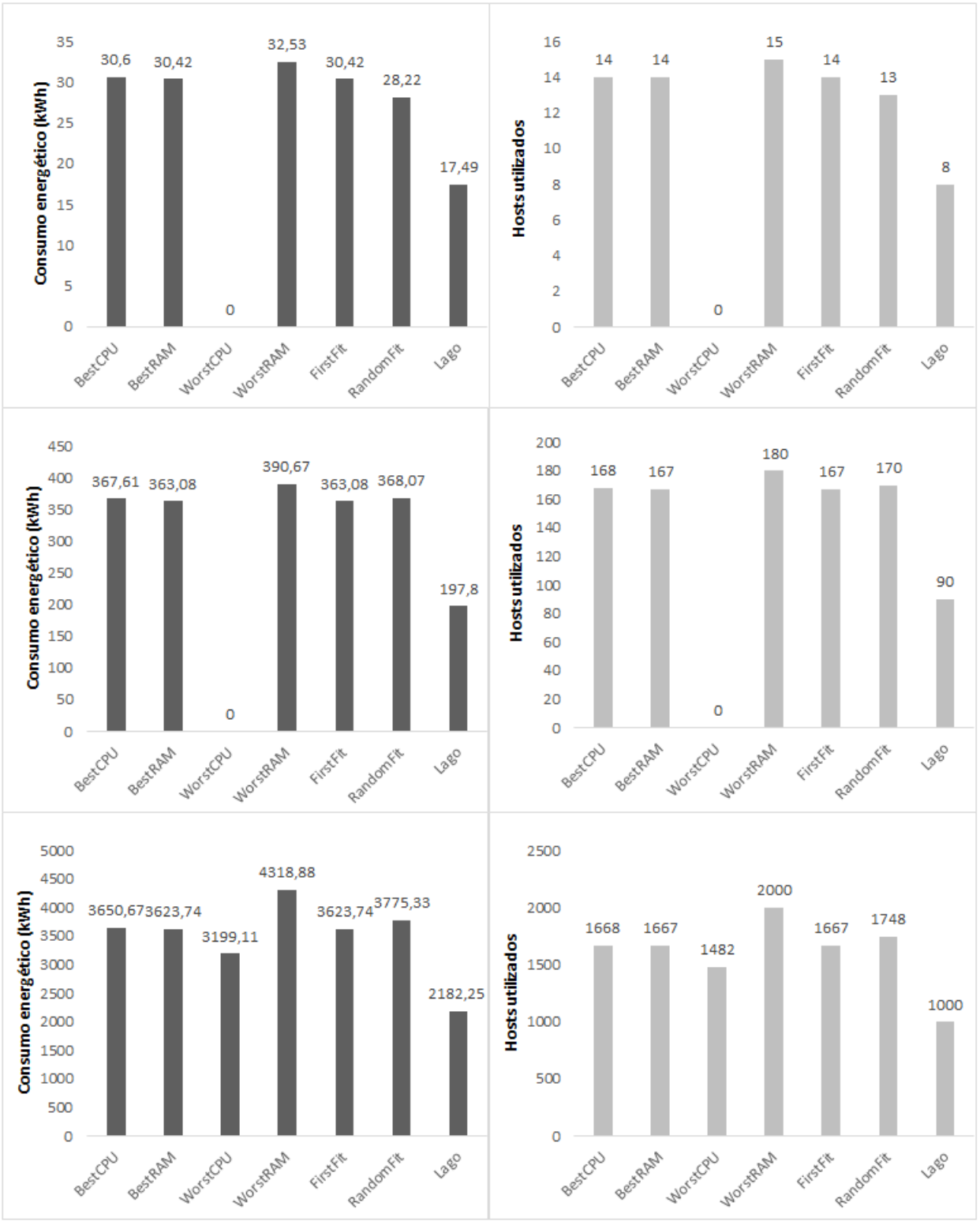

Figura 5.4: Resultados da simulação do modelo heterogêneo para os tamanhos *datacenter* pequeno, médio e grande.

o consumo energético está diretamente interligado com o número de máquinas utilizadas, dado que cada máquina possui um modelo energético que define seu consumo de energia. Ainda é possível observar que para o mesmo número de máquinas utilizadas para diferentes algoritmos gera diferentes consumos energéticos. Isso ocorre devido ao fato de que o consumo energético vária de acordo com a utilização do *host*. Ainda pode-se notar que para cada o modelo os resultados para cada tamanho testados são muito semelhantes, o que mostra que para alguns casos o tamanho do ambiente de simulação configurado no *framework* o resultado final não é muito alterado.

O que mais se destaca nos resultados obtidos é que o algoritmo escolhido da literatura para implementação realmente possui um resultado muito superior se comparado a eficiência energética e a utilização de recursos entre os algoritmos convencionais implementados, e o quanto esse resultado afeta a infraestrutura da nuvem como um todo. Como o foco do algoritmo *Lago* era a métrica de eficiência energética, foi testado e observado que para ambientes de diferentes tipos de VMs e *hosts* realmente o algoritmo possui uma eficiência superior aos algoritmos utilizados para teste. Portanto, através do framework foi possível validar a eficiencia do algoritmo testado.

Capítulo 6

Considerações finais

Como visto no trabalho, a computação em nuvem pode ser considerada como um grande conjunto de recursos virtualizados e compartilhados (hardware, plataformas de desenvolvimento ou software) que podem ser utilizados e acessados por clientes por meio da rede. Esse tipo de tecnologia tem sido muito utilizada por diversas empresas como é o caso da Google, do Dropbox e da Amazon.

O que torna a computação em nuvem uma alternativa interessante é a facilidade de adquirir mais recursos de acordo com a demanda necessária pagando de acordo com uso de recursos, o fácil acesso dos recursos pela rede é outra característica determinante, o agrupamento de recursos para acesso de vários usuários, a elasticidade no fornecimento de recursos, e a garantia do controle e otimização dos recursos. Ainda de acordo com o nível de abstração e pelos tipos de recursos fornecidos pelo provedor, a computação em nuvem pode ser dividida em diferentes tipos de serviço, com diferentes níveis de controle e acesso entre usuário e provedor. Os modelos de serviços são: infraestrutura como serviço (IaaS), plataforma como um serviço (PaaS) e software como um serviço (SaaS).

O compartilhamento desses recursos é feito através do uso de virtualização, que permite que vários ambientes virtuais sejam executados sobre o hardware físico chamados de máquinas virtuais, permitindo a otimização do uso de recursos e a economia de escala. Este processo de requisitar ou liberar recursos é realizado pelo software que serve como mediador entra as múltiplas instâncias de sistemas operacionais em uma mesma máquina física chamado de monitor de VM ou mesmo hipervisor (*hypervisor*), que é o responsável por garantir o isolamento entre as VMs, limitar os recursos de cada uma e definir o modo de acesso.

Um ambiente de nuvem que utiliza um modelo de serviço IaaS possui um conjunto conside-

rável de PMs e VMs, existem diversos critérios que podem ser considerados para uma solução eficiente do problema de alocação de VMs, dependendo das políticas de controle e objetivos de otimização. Portanto, a comparação e teste de algoritmos de alocação de VMs é um grande desafio da área da computação em nuvem, por isso a importância desse trabalho que teve o objetivo de implementar uma abordagem em que um pesquisador possa facilmente testar seu algoritmo e compará-lo com outros algoritmos usando um mesmo ambiente de simulação.

Foram selecionados alguns trabalhos da literatura para a revisão bibliográfica, cada trabalho foi classificado utilizando um conjunto de critérios. Os critérios escolhidos para a classificação dos trabalhos foram: desempenho, eficiência, rede, redução de custos e redução de recursos. Neste trabalho foi implementado o *framework* na linguagem Java utilizando a ferramenta CloudSim. Foram implementados sete algoritmos para a validação do *framework*, foram realizados 63 testes com essses algoritmos que foi o suficiente para compreender a diferença que cada algoritmo pode gerar na alocação de VMs dependendo do cenário utilizado, pode-se notar o quanto é importante esse estudo e acompanhamento em ambientes de computação em nuvem.

Os resultados foram muito satisfatórios. Foi possível implementar facilmente os algoritmos e compará-los. Os testes mostraram a eficiência de cada algoritmo que depende da arquitetura do ambiente de nuvem, das características das máquinas físicas e das requisições das VMs. Portanto, o algoritmo escolhido para a alocação de VMs em um ambiente de computação em nuvem pode fazer uma grande diferença na eficiência da infraestrutura do sistema. O resultado também mostrou que o algoritmo implementado da literatura é realmente mais eficiente que os demais algoritmos convencionais para a sua métrica de aplicação.

Como trabalhos futuros podem ser implementados diversos algoritmos de alocação de VMs para outras métricas no *framework*, por exemplo, a implementação de algoritmos que utilizam largura de banda e a implementação de migração de VMs entre diferentes máquinas físicas com o objetivo de balanceamento de carga.

Referências Bibliográficas

[1] J. Antony. Placement of virtual machines in a virtualized computing environment, Sept. 12 2013. US Patent App. 14/024,652.

[2] U. Bellur, C. S. Rao, and S. D. M. Kumar. Optimal placement algorithms for virtual machines. *CoRR*, abs/1011.5064, 2010.

[3] O. Biran, A. Corradi, M. Fanelli, L. Foschini, A. Nus, D. Raz, and E. Silvera. A stable network-aware vm placement for cloud systems. In *Proceedings of the 2012 12th IEEE-E/ACM International Symposium on Cluster, Cloud and Grid Computing (ccgrid 2012)*, pages 498–506. IEEE Computer Society, 2012.

[4] J. S. Bolin. *Use case analysis for adopting cloud computing in Army test and evaluation*. PhD thesis, Monterey, California. Naval Postgraduate School, 2010.

[5] D. Bonde. Techniques for virtual machine placement in clouds. *Department of Computer Science and Engineering, Indian Institute of Technology, Bombay, Mumbai*, 2010.

[6] A. Bosing and E. R. Kaufmann. Virtualização de servidores e desktops. *Unoesc & Ciência-ACET*, 3(1):47–64, 2012.

[7] K. K. Breitman. *Arcabouço para desenvolvimento de serviços baseados na Simulação de Monte Carlo na Cloud*. PhD thesis, PUC-Rio, 2012.

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

[9] R. S. Camati, A. Calsavara, and L. Lima Jr. Solving the virtual machine placement problem as a multiple multidimensional knapsack problem. *ICN 2014*, page 264, 2014.

[10] A. Carissimi. Virtualização: da teoria a soluções. *Minicursos do Simpósio Brasileiro de Redes de Computadores–SBRC*, 2008:173–207, 2008.

[11] S. Chaisiri, B.-S. Lee, and D. Niyato. Optimal virtual machine placement across multiple cloud providers. In *Services Computing Conference, 2009. APSCC 2009. IEEE Asia-Pacific*, pages 103–110. IEEE, 2009.

[12] L. Chen, J. Zhang, L. Cai, R. Li, T. He, and T. Meng. Mtad: A multitarget heuristic algorithm for virtual machine placement. *International Journal of Distributed Sensor Networks*, pages 1–14, 2014.

[13] E. Coutinho, F. R. Sousa, D. G. Gomes, and J. Souza. Elasticidade em computaçao na nuvem: Uma abordagem sistemática. *XXXI Simpósio Brasileiro de Redes de Computadores e Sistemas Distribuıdos (SBRC 2013)-Minicursos*, 2013.

[14] D. Erickson, B. Heller, N. McKeown, and M. Rosenblum. Using network knowledge to improve workload performance in virtualized data centers. In *2014 IEEE International Conference on Cloud Engineering (IC2E)*, pages 185–194. IEEE, 2014.

[15] I. Foster, Y. Zhao, I. Raicu, and S. Lu. Cloud computing and grid computing 360-degree compared. In *Grid Computing Environments Workshop, 2008. GCE'08*, pages 1–10. Ieee, 2008.

[16] G. Galante. *Explorando a elasticidade em nível de programação no desenvolvimento e na execução de aplicações científicas*. PhD thesis, Universidade Federal do Paraná, 2014.

[17] D. E. Goldberg and J. H. Holland. Genetic algorithms and machine learning. *Machine learning*, 3(2):95–99, 1988.

[18] H. Goudarzi and M. Pedram. Energy-efficient virtual machine replication and placement in a cloud computing system. In *Cloud Computing (CLOUD), 2012 IEEE 5th International Conference on*, pages 750–757. IEEE, 2012.

[19] A. Gupta, L. V. Kale, D. Milojicic, P. Faraboschi, and S. M. Balle. Hpc-aware vm placement in infrastructure clouds. In *Cloud Engineering (IC2E), 2013 IEEE International Conference on*, pages 11–20. IEEE, 2013.

[20] A. Gupta, D. Milojicic, and L. V. Kalé. Optimizing vm placement for hpc in the cloud. In *Proceedings of the 2012 workshop on Cloud services, federation, and the 8th open cirrus summit*, pages 1–6. ACM, 2012.

[21] H. Jin, D. Pan, J. Xu, and N. Pissinou. Efficient vm placement with multiple deterministic and stochastic resources in data centers. In *Global Communications Conference (GLOBECOM), 2012 IEEE*, pages 2505–2510. IEEE, 2012.

[22] D. G. d. Lago, E. R. Madeira, and L. F. Bittencourt. Power-aware virtual machine scheduling on clouds using active cooling control and dvfs. In *Proceedings of the 9th International Workshop on Middleware for Grids, Clouds and e-Science*, page 2. ACM, 2011.

[23] M. Laureano. *Máquinas virtuais e emuladores: conceitos, técnicas e aplicações*. Novatec Editora, 2006.

[24] K. Le, R. Bianchini, J. Zhang, Y. Jaluria, J. Meng, and T. D. Nguyen. Reducing electricity cost through virtual machine placement in high performance computing clouds. In *Proceedings of 2011 International Conference for High Performance Computing, Networking, Storage and Analysis*, page 22. ACM, 2011.

[25] K. Li, H. Zheng, and J. Wu. Migration-based virtual machine placement in cloud systems. In *Cloud Networking (CloudNet), 2013 IEEE 2nd International Conference on*, pages 83–90. IEEE, 2013.

[26] W. Li. Virtual machine placement in cloud environments. Tese de licenciatura, UMEA – Umeå University, Umeå - Suécia, 2012.

[27] W. Li, P. Svärd, J. Tordsson, and E. Elmroth. Cost-optimal cloud service placement under dynamic pricing schemes. In *Proceedings of the 2013 IEEE/ACM 6th International Conference on Utility and Cloud Computing*, pages 187–194. IEEE Computer Society, 2013.

[28] P. Mell and T. Grance. Draft the nist definition of cloud computing. *Nist Special Publication*, 145(6):7, 2009.

[29] P. Moscato et al. On evolution, search, optimization, genetic algorithms and martial arts: Towards memetic algorithms. *Caltech concurrent computation program, C3P Report*, 826:1989, 1989.

[30] R. S. Oliveira, A. S. Carissimi, and S. S. Toscani. *Sistemas Operacionais-Vol. 11: Série Livros Didáticos Informática UFRGS*. Bookman.

[31] F. L. Pires and B. Barán. Multi-objective virtual machine placement with service level agreement: A memetic algorithm approach. In *Proceedings of the 2013 IEEE/ACM 6th International Conference on Utility and Cloud Computing*, pages 203–210. IEEE Computer Society, 2013.

[32] F. L. Pires and B. Barán. A virtual machine placement taxonomy. In *Cluster, Cloud and Grid Computing (CCGrid), 2015 15th IEEE/ACM International Symposium on*, pages 159–168. IEEE, 2015.

[33] S. M. Sait and K. S. Shahid. Engineering simulated evolution for virtual machine assignment problem. *Applied Intelligence*, pages 1–12, 2015.

[34] V. A. K. Sarma, R. Rajendra, P. Dheepan, and K. S. Kumar. An optimal ant colony algorithm for efficient vm placement. *Indian Journal of Science and Technology*, 8(S2):156–159, 2015.

[35] L. Shi, B. Butler, D. Botvich, and B. Jennings. Provisioning of requests for virtual machine sets with placement constraints in iaas clouds. In *Integrated Network Management (IM 2013), 2013 IFIP/IEEE International Symposium on*, pages 499–505. IEEE, 2013.

[36] L. M. Vaquero, L. Rodero-Merino, J. Caceres, and M. Lindner. A break in the clouds: towards a cloud definition. *SIGCOMM Comput. Commun. Rev.*, 39:50–55, 2008.

[37] H. T. Vu and S. Hwang. A traffic and power-aware algorithm for virtual machine placement in cloud data center. *International Journal of Grid & Distributed Computing*, 7(1):350–355, 2014.

[38] R. Wottrich, T. Genez, and W. Pereira. Uma visão geral sobre sistemas virtualizados. *Minicurso de arquitetura de computadores. UNICAMP, São Paulo, Brasil*, 2012.