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

Cláudio Henrique Gramulha

Comparação de algoritmos para alocação de máquinas virtuais em sistemas de computação em nuvem

São José do Rio Preto
2022

Cláudio Henrique Gramulha

Comparação de algoritmos para alocação de máquinas virtuais em sistemas de computação em nuvem

Monografia apresentada ao Programa de graduação em Ciência da Computação da UNESP para obtenção do título de Bacharel.

Orientadora: Profa. Dra. Renata Spolon Lobato

São José do Rio Preto - SP
2022

G747c

Gramulha, Cláudio Henrique

Comparação de algoritmos para alocação de máquinas virtuais em sistemas de computação em nuvem / Cláudio Henrique Gramulha. -- São José do Rio Preto, 2022

51 p. : il., tabs.

Trabalho de conclusão de curso (Bacharelado - Ciência da Computação) - Universidade Estadual Paulista (Unesp), Instituto de Biociências Letras e Ciências Exatas, São José do Rio Preto

Orientadora: Renata Spolon Lobato

1. Computação em nuvem. 2. Algoritmos de alocação. 3. Máquinas virtuais. 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).

Essa ficha não pode ser modificada.

Cláudio Henrique Gramulha

Comparação de algoritmos para alocação de máquinas virtuais em sistemas de computação em nuvem

Monografia apresentada ao Programa de graduação em Ciência da Computação da UNESP para obtenção do título de Bacharel.

Banca examinadora:
Profa. Dra. Renata Spolon
UNESP – São José do Rio Preto
Orientadora

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

Prof. Dr. Eng. Rodrigo Capobianco Guido
UNESP – São José do Rio Preto

São José do Rio Preto - SP
21 de Dezembro de 2022

Agradecimentos

Agradeço a meus amigos e familiares por todo o apoio durante o curso de graduação. Também estendo meus agradecimentos à Profa. Dra. Renata Spolon Lobato e ao Prof. Dr. Aleardo Manacero Junior pela orientação e pelos conselhos oferecidos durante o desenvolvimento desse trabalho. Por fim registro também meu agradecimento ao Prof. Dr. Eng. Rodrigo Capobianco Guido, que fez parte da banca avaliadora.

"Nenhum homem nem nenhuma nação podem existir sem uma ideia sublime."
Fiódor Dostoiévski

Resumo

Sistemas de computação em nuvem oferecem uma gama de serviços amplamente utilizados por vários setores da indústria e da academia. Essa tecnologia oferece acesso a recursos de computação, como redes, servidores e armazenamento de maneira compartilhada. Suas principais características podem ser resumidas ao autoatendimento sob demanda, amplo acesso à rede, agrupamento de recursos, elasticidade e medição de uso de serviço. Nuvens podem ser implementadas de diversas maneiras, oferecendo serviços variados, como SaaS, PaaS e IaaS. Sistemas em nuvem são uma emulação de um sistema de computador. Logo, virtualização e alocação de máquina virtuais são processos-chave para que esses ambientes sejam bem sucedidos. Muitas classes de algoritmos que procuram aprimorar tais processos foram desenvolvidos, buscando otimizar desempenho, usabilidade, disponibilidade e redução de custos operacionais. Esse trabalho objetiva fornecer uma visão geral desses sistemas, discutir virtualização, alocação de VMs e introduzir algoritmos do estado da arte que cumprem com essa tarefa. Comparando resultados obtidos da implementação de alguns desses algoritmos com dados coletados no simulador *CloudSim* para encontrar desempenhos que se destacam entre os métodos testados.

Palavras-chave: Computação em nuvem, algoritmos de alocação, máquinas virtuais.

Abstract

Cloud computing systems offer a range of services that are widely used by various sectors of industry and academia. This technology provides access to computing resources such as networks, servers and storage on a shared basis. Its main characteristics can be summarized as on-demand self-service, broad network access, resource pooling, elasticity and measured service. Clouds can be implemented in different ways, offering different services, with SaaS being the most known and used model. Cloud systems are an emulation of a computer system. Therefore, virtualization and virtual machine placement are key processes for these environments to be successful. Many classes of algorithms that seek to improve such processes were developed, pursuing performance optimization, usability, availability and operational cost reduction. This paper aims to provide an overview of these systems, discuss virtualization, VM placement and to introduce state of art algorithms that fulfill this task. Comparing results obtained from the implementation of such algorithms with data collected in the simulator *CloudSim* to find performances that stand out among the tested methods.

Key-words: Cloud computing, placement algorithms, virtual machine.

Lista de Figuras

Lista de Tabelas

List of Algorithms

Lista de abreviaturas e siglas

- WAN: *Wide Area Network*
- SaaS: *Software as a Service*
- PaaS: *Platform as a Service*
- IaaS: *Infrastructure as a Service*
- VM: *Virtual Machine*
- PM: *Physical Machine*
- QoS: *Quality of Service*
- NIST: *National Institute of Standards and Technology*
- TI: Tecnologia da Informação
- AWS: *Amazon Web Services*
- VMM: *Virtual Machine Monitor*
- GA: *Genetic Algorithm*
- ACO: *Ant Colony Optimization*
- KLVMP: *Kernighan-Lin based heuristic for Virtual Machine Placement*
- PSO: *Particle Swarm Optimization*
- SA: *Simulated Annealin*
- TS: *Tabu Search*
- BF: *Best-Fit*
- FF: *First-Fit*
- NF: *Next-Fit*
- WF: *Worst-Fit*
- SLA: *Service-level agreement*
- MFPED: *Medium-fit power efficient decreasing algorithm*
- PEBFD: *Power efficient first-fit decreasing algorithm*
- IQR: *Inter quartile range algorithm*

Sumário

1 Introdução

1.1 Contextualização

Um sistema distribuído é um sistema no qual componentes localizados em computadores em rede se comunicam e coordenam suas ações através da troca de mensagens (COULOURIS; DOLLIMORE; KINDBERG, 2005). O termo *distribuído* refere-se ao fato de que diferentes partes do sistema estão espalhadas por uma rede, geralmente uma rede de longa distância (WAN), onde seus componentes interagem entre si para resolver tarefas (LYNCH, 1996).

A computação em nuvem é frequentemente usada para se referir a *software* como serviço (SaaS), o qual é um tipo de serviço onde os usuários acessam aplicativos hospedados na nuvem. Os sistemas de computação em nuvem normalmente alocam máquinas virtuais (VMs), que são imagens de *software* de máquinas reais, aos usuários. Elas podem ser criadas, destruídas e movidas conforme necessário. Com a computação em nuvem, indivíduos e organizações podem obter acesso de rede sob demanda a um conjunto compartilhado de recursos como servidores, armazenamento e aplicativos (SUNYAEV, 2020).

A alocação de VMs nesses sistemas é um problema complexo que deve levar em consideração as necessidades dos usuários, os recursos do sistema e as restrições do sistema. O objetivo da alocação de máquinas virtuais é encontrar um mapeamento de VMs para máquinas físicas que atenda às necessidades dos usuários, minimizando o custo dos recursos utilizados. O problema de alocação dessas máquinas tem sido estudado extensivamente nos últimos anos. Muitos algoritmos foram propostos, mas nenhum algoritmo único garante encontrar a solução ótima. A melhor abordagem é usar uma combinação de heurísticas e meta-heurísticas (MANN; SZABÓ, 2017).

1.2 Motivação

Em sistemas de computação em nuvem, a alocação de máquinas virtuais é um problema chave de gerenciamento de recursos. O objetivo é minimizar o custo dos recursos alocados e, simultaneamente, atender aos requisitos de qualidade de serviço (QoS) dos usuários.

Há uma grande variedade de algoritmos propostos para a alocação de VMs em sistemas em nuvem. Esses algoritmos podem ser classificados em duas categorias principais: estáticos e dinâmicos. Algoritmos estáticos são aqueles que não consideram a natureza variável do tempo das cargas de trabalho, enquanto que, em contrapartida, os algoritmos dinâmicos o fazem.

A importância de algoritmos eficientes para alocação de máquinas virtuais em sistemas em nuvem vem aumentando nos últimos anos devido à crescente popularidade da computação em nuvem. No entanto, mais estudos são necessários para que se possa estabelecer qual é o melhor deles. Ou, pelos menos, saber onde e como poderiam ser melhorados. Dessa maneira, torna-se essencial o entendimento e conhecimento das diversas técnicas que se pode utilizar para a realização da tarefa de alocar VMs. Assim, para que se possa tomar decisões educadas na escolha de um método para sua implementação, visando a obtenção dos melhores resultados.

1.3 Objetivo

O objetivo desta pesquisa é comparar o desempenho de diferentes algoritmos de alocação de VMs em sistemas de computação em nuvem. Algoritmos estáticos e dinâmicos são considerados e tem seu desempenho analisado em termos de utilização de recursos, principalmente quanto ao consumo de energia e garantia do SLA (do inglês, *Service-level agreement*, ou acordo de nível de serviço) por meio da minimização da quantidade de migrações realizadas no processo de alocação. O estudo ajudará a identificar o algoritmo mais eficiente e eficaz para essa tarefa.

Ainda, ajudará a entender as limitações de cada um dos métodos e os fatores que precisam ser levados em consideração na escolha de um algoritmo para alocação de VMs em sistemas *cloud*.

Dessa forma, são listados os seguintes objetivos:

1. Compreender as implementações de cada algoritmo e os fatores que devem ser considerados na escolha de um algoritmo para alocação de máquinas virtuais em sistemas de computação em nuvem.

2. Comparar o desempenho dos algoritmos a serem testados para alocação de VMs.

3. Identificar o algoritmo mais eficiente e eficaz dentre os algoritmos testados para alocação de máquinas virtuais em sistemas de computação em nuvem.

1.4 Metodologia

O estudo será realizado comparando o desempenho de diferentes algoritmos para alocação de máquinas virtuais em sistemas de computação em nuvem. O desempenho dos algoritmos será medido em termos de consumo de energia para alocar as máquinas virtuais, da quantidade migrações realizadas e da qualidade dos resultados.

Para esse fim, será utilizado o *CloudSim*(CALHEIROS et al., 2011), *software* de simulação que permite a modelagem e simulação de ambientes de computação em nuvem. Esse software traz facilidades na construção de modelos de nuvem, escalonadores e cargas de trabalho. Apresentando um motor de simulação que oferece métricas de desempenho e permite que os modelos simulados sejam analisados com precisão.

2 Revisão bibliográfica

2.1 Conceitos iniciais de sistemas distribuídos

A definição de sistema distribuído é um sistema que consiste em diversos componentes interconectados, cada um executando em um computador diferente e se comunicando por troca de mensagens. Esses sistemas podem ser definidos livremente como uma rede de computadores, cada um dos quais executando independentemente uma instância de um aplicativo. Os sistemas distribuídos são caracterizados pelo fato de cada computador no sistema ser independente, o que significa que cada um pode mudar seu estado sem afetar os outros (STEEN; TANENBAUM, 2002).

O termo *sistema distribuído* pode se referir a várias coisas diferentes, mas o ponto comum é que todas elas envolvem a distribuição de algum tipo de recurso ou serviço em uma rede de computadores. Isso pode ser algo tão simples quanto um sistema de arquivos, onde os arquivos são armazenados em um servidor e acessados por clientes, ou algo mais complexo como um banco de dados distribuído.

Nesses sistemas, cada computador tem sua própria memória privada e armazenamento local. Isso contrasta com um sistema centralizado, onde todos os computadores do sistema compartilham uma memória e armazenamento comuns. A vantagem de um sistema distribuído é que ele é mais escalável e pode lidar com mais tráfego do que um sistema centralizado. Isso se dá pelo fato de que, nesses tipos de sistemas, os recursos são compartilhados.

Existem alguns conceitos-chave que são importantes de se entender para que se possa projetar e implementar um sistema distribuído.

- Computadores do sistema devem ser capazes de se comunicar entre si. Essa comunicação pode ser alcançada de várias maneiras, como por meio de uma rede ou de um sistema de armazenamento compartilhado (STEEN; TANENBAUM, 2002).

- Computadores do sistema devem ser capazes de coordenar suas ações. Essa coordenação pode ser alcançada por meio de vários meios, como um relógio compartilhado ou um sistema que permite a passagem de mensagens entre os computadores (STEEN; TANENBAUM, 2002).

- Computadores do sistema devem ser conseguir detectar e se recuperar de falhas. Isso é necessário para que o sistema possa continuar operando mesmo se um dos computadores falhar (STEEN; TANENBAUM, 2002).

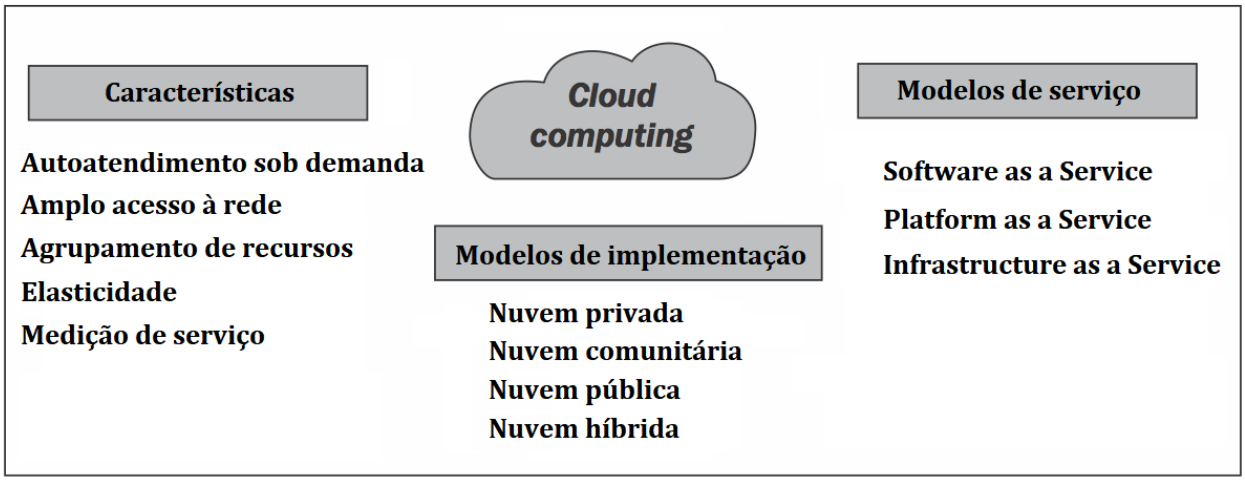
- Finalmente, os computadores do sistema devem ser capazes de se adaptar às mudanças no ambiente. Isso pode incluir alterações na rede ou alterações na carga de trabalho (COULOURIS; DOLLIMORE; KINDBERG, 2005).

Os sistemas distribuídos estão se tornando mais predominantes à medida que a necessidade por escalabilidade e por disponibilidade aumenta. A rápida adoção da internet em escala global tornou os recursos de computação mais baratos, poderosos e mais presentes do que nunca. Potencializando a busca por armazenamento, hospedagem e entrega de serviços pelo meio *online* (ZHANG; CHENG; BOUTABA, 2010). Desse ecossistema, surge um novo paradigma que visa saciar as demandas do mercado, a computação em nuvem.

2.2 Sistemas de computação em nuvem

De acordo com o NIST (*National Institute of Standards and Technology*, Estados Unidos), O termo *computação em nuvem* refere-se a um modelo de tecnologia que permite o acesso a um

Figura 1: Essenciais da computação em nuvem



Adaptado de: *https://www.ibm.com/blogs/cloud-computing/2014/01/31/cloud-computing-defined-characteristics-service-levels/*

conjunto compartilhado de recursos de computação (por exemplo, redes, servidores, armazenamento, aplicativos e serviços) por meio de uma conexão de rede. Esse modelo de tecnologia oferece várias vantagens, como acesso sob demanda, conveniência e flexibilidade. A agência americana ainda lista cinco características essenciais da computação em nuvem: autoatendimento sob demanda, amplo acesso à rede, *pool* de recursos, elasticidade ou expansão rápida e medição de serviço. Lista, também, três modelos de serviço (*software*, plataforma e infraestrutura) e quatro modelos de implementação (privado, comunitário, público e híbrido) que juntos categorizam maneiras de fornecer serviços em nuvem (BADGER et al., 2012), vide figura 1.

Uma das principais características desse tipo de sistema é que ele permite que os usuários acessem e usem recursos sob demanda. Para fornecer acesso a esse tipo de serviço, os sistemas de computação em nuvem precisam ser capazes de alocar recursos dinamicamente aos usuários. E, uma maneira de fornecer tal acesso é por meio da virtualização para criar máquinas virtuais (BOSS et al., 2007) que podem ser alocadas aos usuários conforme e quando necessário.

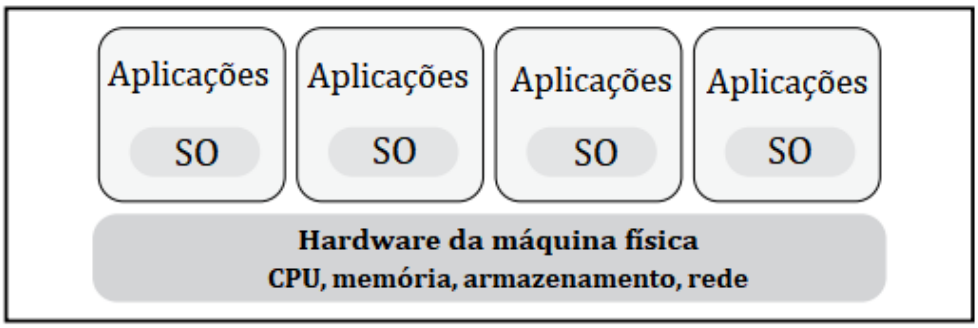
A virtualização é uma técnica para ocultar as características físicas dos recursos de computação dos usuários e apresentá-los como recursos lógicos. Ele permite que vários recursos virtuais sejam criados em um único recurso físico (por exemplo, um servidor), como ilustra a figura 2. A virtualização é amplamente utilizada em sistemas de computação em nuvem para criar VMs.

Uma VM é uma emulação de um sistema de computador. As VMs são criadas executando um *hypervisor* em um servidor físico. Um *hypervisor* é um programa de *software* que permite que vários sistemas operacionais sejam executados em um único servidor físico. Cada VM criada pelo *hypervisor* é isolada das demais e possui seu próprio sistema operacional, aplicativos e dados (DESAI et al., 2013).

Algoritmos de alocação de VMs são usados para alocar VMs a usuários em sistemas de computação em nuvem de maneira a atender às necessidades do usuário e, ao mesmo tempo, maximizar a utilização de recursos. Tais algoritmos de alocação precisam ser capazes de alocar VMs de forma rápida e eficiente. Eles também precisam ser escaláveis para que possam lidar com um grande número de usuários e inúmeras instâncias de VMs.

Existem vários algoritmos de alocação de VMs que foram propostos na literatura. No próximo capítulo, será abordada uma visão geral dos algoritmos de alocação de VMs, assim como serão descritas suas principais características e modelos de implementação.

Figura 2: Máquinas virtuais rodando no hardware físico



Fonte: (HOOPES, 2009), traduzido pelo autor

2.2.1 Características essenciais dos sistemas de computação em nuvem

A computação em nuvem é um modelo para fornecer serviços de tecnologia da informação usando a *internet*. Em vez de se conectar diretamente a um servidor, os usuários usam ferramentas e aplicativos baseados na web para acessar os recursos. Esse modelo de sistema permite que os usuários acessem aplicativos e dados armazenados em servidores remotos, possibilitando trabalhar de qualquer lugar com conexão à *internet* (RASHID; CHATURVEDI, 2019).

O modelo de computação em nuvem é baseado em uma abordagem de pagamento conforme o uso, no qual os usuários pagam apenas pelos recursos que utilizam. Isso o torna uma solução econômica para empresas de todos os tamanhos. Além disso, a computação em nuvem é escalável, para que usuários possam adicionar ou remover recursos facilmente conforme suas necessidades mudam.

Os sistemas de computação em nuvem oferecem muitos recursos essenciais, incluindo:

- **Autoatendimento sob demanda** (*On-Demand Self-Service*): usuários podem acessar recursos de nuvem de maneira automática sem precisar entrar em contato com um provedor de serviços (SRINIVAS; REDDY; QYSER, 2012)(RASHID; CHATURVEDI, 2019).

- **Amplo acesso à rede** (*Broad Network Access*): recursos de nuvem podem ser acessados de qualquer lugar com conexão à Internet. Isso possibilita o trabalho de qualquer lugar, aumentando a flexibilidade e a produtividade, com o acesso sendo feito através de dispositivos pessoais do usuário (BUYYA; BROBERG; GOSCINSKI, 2010).

- **Agrupamento de recursos** (*Resource Pooling*): recursos da nuvem são agrupados e compartilhados entre os usuários. São exemplos desses recursos: processamento, memória e armazenamento. Por conta disso, o usuário não tem conhecimento da localização desses recursos, apenas em situações onde, deliberadamente, escolher um determinado *datacenter* para operar (SAJID; RAZA, 2013)(BUYYA; BROBERG; GOSCINSKI, 2010).

- **Elasticidade** (*Rapid Elasticy*): recursos de nuvem podem ser ampliados ou reduzidos de maneira rápida e fácil para atender às demandas em constante mudança. Isso possibilita que usuários possam responder rapidamente às mudanças na demanda, sem incorrer no custo de manter a capacidade não utilizada (ZISSIS; LEKKAS, 2012).

- **Medição de serviço** (*Measured Service*): provedores de nuvem oferecem serviços medidos, para que os usuários paguem apenas pelos recursos que utilizam. Permitindo o controle de custos e evitando gastos excessivos com recursos de TI (SAJID; RAZA, 2013).

2.2.2 Principais modelos de serviço em sistemas de computação em nuvem

Cloud computing é um modelo de computação que garante acesso a um conjunto compartilhado de recursos de computação que podem ser rapidamente provisionados(MELL; GRANCE et al., 2011), eximindo o uso de dispositivos pessoas e a necessidade de servidores locais para acessar serviços de computação.

Esse modelo é atraente para organizações, porque pode economizar significativamente em custos de hardware e software. Os principais modelos de serviços nesse tipo de computação são: infraestrutura como serviço (IaaS), plataforma como serviço (PaaS) e software como serviço (SaaS).

- **IaaS** é o modelo mais básico de computação em nuvem, onde pode-se acessar recursos de computação virtualizados, como servidores, armazenamento e rede. Isso pode ser feito sob demanda, o que significa que os usuários pagam apenas pelos recursos que usam. Os provedores de IaaS geralmente oferecem uma variedade de serviços, como computação, armazenamento, rede, segurança e monitoramento. A IaaS é atraente para as organizações porque pode economizar significativamente em custos com *hardware* e com *software*. Além disso, o IaaS pode ser usado para aumentar ou diminuir rapidamente os recursos de TI conforme necessário, proporcionando grande flexibilidade (ZISSIS; LEKKAS, 2012).

- **PaaS** é um modelo de computação em nuvem que fornece uma plataforma para desenvolvedores criarem, testarem e implantarem aplicativos, oferecendo uma variedade de ferramentas e serviços que facilitam o gerenciamento dos mesmos. Alguns provedores de PaaS oferecem ambiente gerenciado para hospedagem de aplicativos *Web*, enquanto outros oferecem uma plataforma de uso mais geral que pode ser usada para qualquer tipo de aplicação, por exemplo. No PaaS o consumidor tem controle sobre os aplicativos implantados e, comumente, sobre as configurações do ambiente de hospedagem de aplicativos. No entanto, não gerencia ou controla a infraestrutura de nuvem subjacente (ZISSIS; LEKKAS, 2012).

- **SaaS** é um modelo de computação em nuvem que fornece aplicativos pela *internet*. Os aplicativos SaaS geralmente são acessados por meio de aplicações *Web* ou aplicativo móvel (ZISSIS; LEKKAS, 2012). Os provedores de SaaS geralmente oferecem um modelo de preços baseado em assinatura, em que os usuários pagam pelo uso do aplicativo mensalmente ou anualmente. O SaaS possibilita uma alternativa ao método tradicional de aquisição de *softwares* (RODRIGUES, 2019).

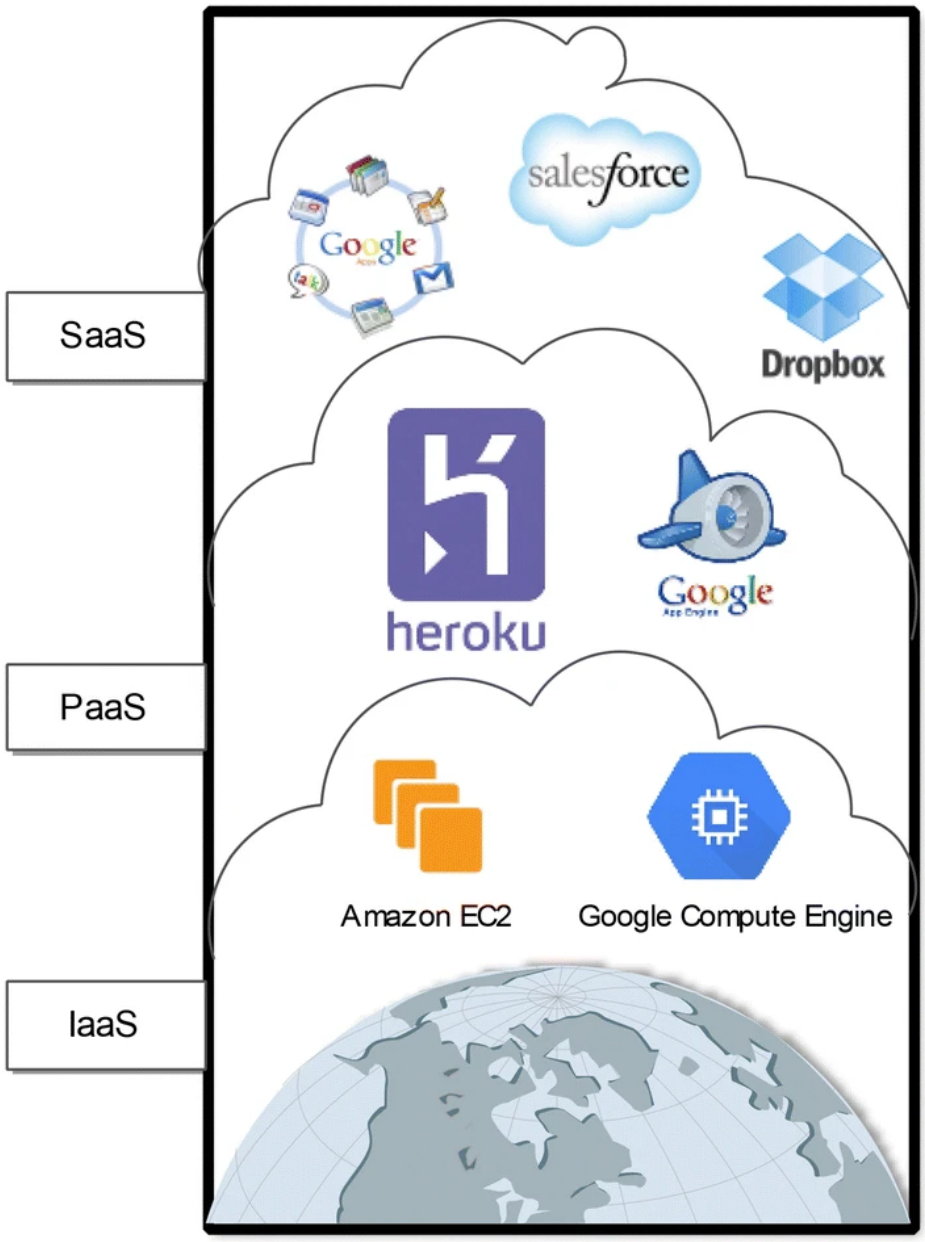
A figura 3 ilustra os principais tipos de serviços oferecidos e alguns dos maiores prestadores desses serviços.

2.2.3 Modelos de implementação em sistemas de computação em nuvem

Os modelos de implementação de computação em nuvem são as maneiras pelas quais os serviços em nuvem são entregues aos clientes. Existem três modelos principais para serviços em nuvem: nuvens públicas, nuvens privadas e nuvens híbridas. Vale a pena citar, também, nuvens de comunidade, por sua relevância em certos contextos. Cada modelo tem suas próprias vantagens e desvantagens que o tornam mais ou menos adequado para diferentes organizações e casos de uso.

- **Nuvens Públicas**: provedores de serviços de nuvem operam e oferecem planos de acesso às nuvens públicas. Os clientes podem acessar esses serviços pela Internet e pagar apenas pelos recursos que utilizarem. Nuvens públicas são o tipo mais comum de modelo de

Figura 3: Diferentes modelos serviços em computação em nuvem



Fonte: (TALEBIAN et al., 2020)

implementação de nuvem e, geralmente, são usadas para aplicações que não exigem um alto grau de segurança ou privacidade. Esse modelo de implementação, geralmente, tem custo reduzido em comparação a, por exemplo, nuvens privadas, pois o provedor de serviços pode distribuir o custo da infraestrutura e das operações entre muitos clientes. Além disso, são flexíveis e escaláveis, o que significa que podem acomodar facilmente aumentos ou diminuições na demanda. Tornando-as ideais para aplicativos com cargas de trabalho altamente variáveis ou desconhecidas.

Contudo, deve-se levar em consideração que esse tipo de serviço pode deixar a desejar em situações onde exista a necessidade de se ter um controle mais refinado sobre controle de dados, rede e configurações de seguranças, uma vez que, nesse tipo de nuvem, essas configurações ficam sob a supervisão dos provedores do serviço (ZHANG; CHENG; BOUTABA, 2010).

São exemplos desse tipo de serviço *Google Cloud* e *AWS* (*Amazon Web Services*).

- **Nuvem de comunidade**: uma nuvem comunitária é um modelo de implementação de nuvem que usa infraestrutura compartilhada para atender a uma comunidade específica de consumidores. É semelhante a uma nuvem pública, pois usa recursos agrupados, mas é diferente por ser projetada para atender a um grupo bem definido. O maior exemplo de implementação desse tipo de serviço é a nuvem governamental, que utiliza infraestrutura compartilhada para atender as agências governamentais.

  Esse tipo de nuvem compartilhada é bastante poderosa, pois ao mesmo tempo que permite reduções de custo de infraestrutura, melhoram questões de segurança ao dar aos membros que compõem o grupo em controle da nuvem maior agência sobre as suas configurações. No entanto, a maior liberdade no gerenciamento dos recursos vem ao custo de se ter que lidar com maior complexidade de configurações, manutenção e questões de escalabilidade.

- **Nuvens Privadas**: nuvens privadas, também conhecidas como nuvens internas, são de propriedade e operadas por uma única organização. Nuvens privadas oferecem mais controle do que nuvens públicas, mas a um custo mais alto. Elas são frequentemente usadas para aplicações que exigem um alto grau de segurança ou privacidade, oferecendo o maior grau de controle sobre desempenho e confiabilidade (ZHANG; CHENG; BOUTABA, 2010).

  As desvantagens desse tipo de nuvem incluem o custo *upfront*, assim como custo de infraestrutura, operações e manutenção. Bem como também apresentam menor potencial de escalabilidade.

- **Nuvens híbridas**: nuvens híbridas são uma composição de nuvens públicas, privadas ou comunitárias que permanecem como entidades únicas, mas são unidas para permitir a portabilidade de dados e aplicativos entre os ambientes (BADGER et al., 2012). As nuvens híbridas oferecem o melhor dos dois mundos, mas podem ser mais complexas de implantar e gerenciar do que apenas nuvens públicas ou privadas. Problemas com compatibilidade também tornam-se um desafio a ser tratado.

2.3 Virtualização

O termo *virtualização* pode ser definido como um processo que cria uma camada de abstração entre os modelos de *hardware* e *software* de um computador. Em outras palavras, cria-se uma versão não física de recursos que variam desde sistemas operacionais, servidores, unidades de armazenamentos, etc. Esse processo é essencial para sistemas de computação em nuvem, pois isola o sistema operacional do *hardware* físico e dos aplicativos que nele estão presentes.

Nessas máquinas virtuais existe um conjunto de *hardware* virtual. Ou seja, elas possuem unidades de armazenamento, CPU, memória e interfaces de rede individuais, no qual um sistema operacional e aplicativos podem ser carregados (HOOPES, 2009). Esse processo permite que diversas máquinas virtuais com sistemas operacionais diferentes sejam executadas separadamente e concorrentemente em uma única máquina física. A virtualização garante que características-chaves da computação em nuvem, anteriormente citadas, como elasticidade e agrupamento de recursos sejam garantidas. Pois, de acordo com *Merrill Lynch*, computação em nuvem, aproveita a virtualização para maximizar o poder de computação (LYNCH, 2008).

Uma máquina física pode virtualizar mais de uma VM, dependendo da capacidade de seu conjunto de *hardware*. Máquinas físicas que hospedam máquinas virtuais são chamadas de *host*, enquanto que uma VM alocada em determinado computador é chamada de *guest*. Os sistemas operacionais das máquinas *guest* tem acesso ao *hardware* do *host*. Porém, não tem conhecimento sobre outras VMs alocadas na mesma máquina *host* e, assim, não interagem com as outras. Ou seja, cada instância de máquina virtual é isolada.

O componente responsável pela comunicação entre os componentes do servidor *host* e das VMs *guests* é nomeado VMM (*Virtual Machine Monitor*), ou *hypervisor*. O *hypervisor* aloca recursos para as VMs e gerencia tanto a execução real que ocorre no *hardware* da PM (máquina física) hospedeira quanto a interação das chamadas do sistema operacional para o *hardware* virtual (cujas tarefas são agendadas no hardware físico real), fornecendo a ilusão de que cada VM está sendo executada em seu próprio hardware (PEREZ-BOTERO; SZEFER; LEE, 2013).

Na figura 4 pode-se acompanhar de maneira simplificada como os componentes dos sistemas de computação em nuvem interagem entre si.

2.4 Alocação de VMs

O processo de alocação de máquinas virtuais em máquinas físicas é conhecido como *virtual machine placement*. A alocação da VM é o processo de determinar quais máquinas físicas devem hospedar quais VMs (GAO et al., 2013). À medida que os *data centers* continuam a crescer em tamanho e complexidade, o mesmo acontece com o desafio de gerenciar VMs nesses ambientes.

O objetivo desse processo é otimizar o desempenho e a utilização dos servidores, além de garantir que as VMs sejam protegidas no caso de uma falha de máquina física (TALEBIAN et al., 2020). Em um ambiente de computação em nuvem, onde os recursos são compartilhados, a alocação e o gerenciamento de recursos torna-se uma parte crítica das operações da VM.

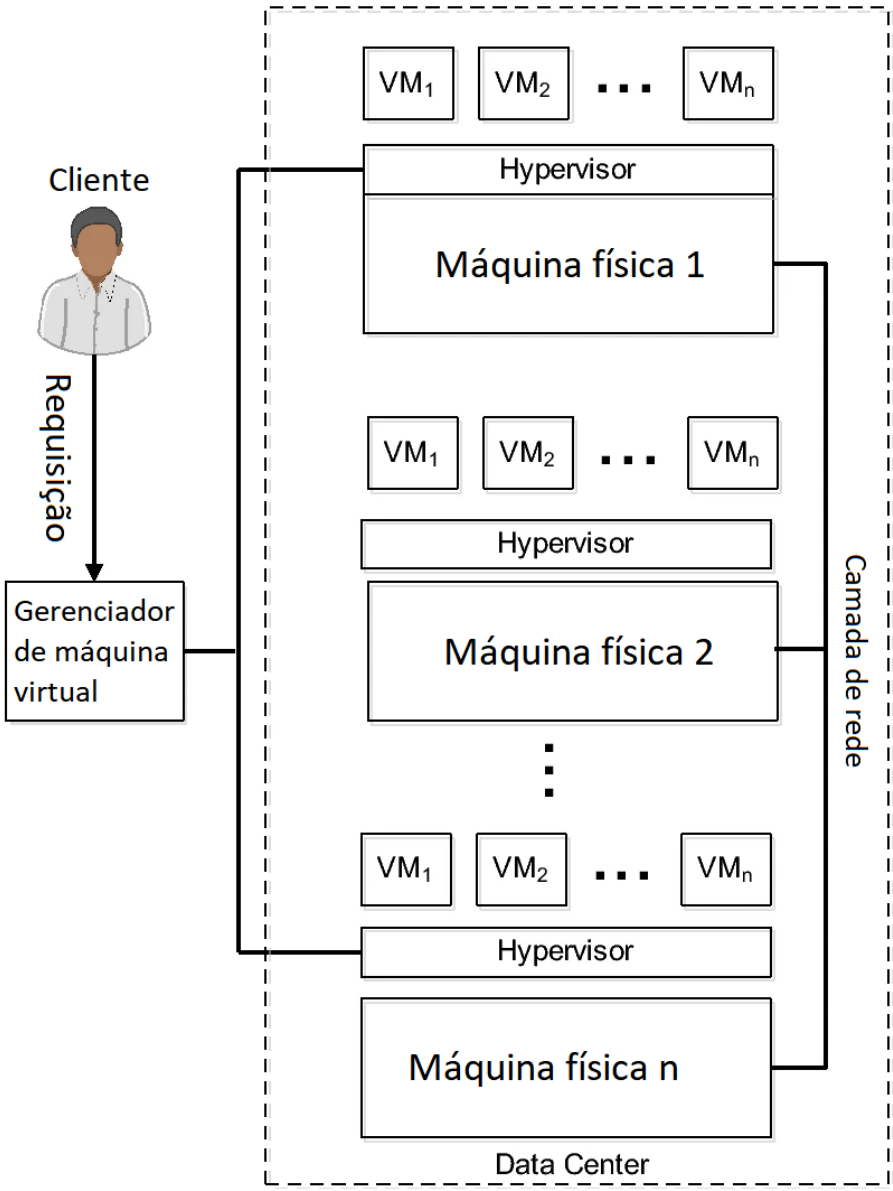
Há vários fatores a serem considerados ao realizar a alocação da VM. São esses:

- **Desempenho**: o desempenho da VM é afetado diretamente pelos recursos que estão disponíveis para ela. Ao alocar VMs, é importante considerar as cargas de trabalho que cada VM executará e garantir que a VM tenha acesso aos recursos necessários para executar essas cargas de trabalho com eficiência (DASHTI; RAHMANI, 2016).

- **Utilização**: um dos objetivos da alocação de VMs é otimizar a utilização de recursos. Ao alocar VMs em máquinas físicas que não estão totalmente utilizadas, é possível aumentar a utilização geral e reduzir o desperdício de recursos e custos de energia (MEISNER; GOLD; WENISCH, 2009).

- **Disponibilidade**: no caso de uma falha de máquina física, é importante que as VMs ainda possam ser executadas em outras máquinas. Ao alocar VMs, é importante considerar como as VMs podem ser protegidas contra falhas de PM. Isso pode incluir o uso de *hardware* redundante ou a implantação de VMs em mais de um local (YANAGISAWA; OSOGAMI; RAYMOND, 2013).

Figura 4: Arquitetura de um sistema de computação em nuvem



Fonte: (TALEBIAN et al., 2020), traduzido pelo autor

- **Custo operacional**: o custo de execução de uma VM pode variar dependendo dos recursos exigidos pela VM. Ao alocar VMs, é importante considerar o custo de execução de cada máquina e garantir que o *down time* para realizar migrações não penalize as aplicações que estejam hospedadas no servidor (GOUDARZI; GHASEMAZAR; PEDRAM, 2012).

Uma das principais operações de virtualização em sistemas de computação em nuvem é a migração de máquinas virtuais. Esse processo permite a movimentação da VM de um *host* para o outro. A migração tem o objetivo de permitir a adaptação no procedimento de alocação de VMs para que haja flexibilidade em se cumprir com requisitos de desempenho, localidade, comunicação, redução de consumo de energia, entre outros (MASDARI; NABAVI; AHMADI, 2016).

De acordo com (MASDARI; NABAVI; AHMADI, 2016), o processo de migração de VMs consiste em quatro passos:

1. Seleciona-se uma PM que esteja sobrecarregada ou sub-carregada, isto é, máquinas que não estiverem usando seus recursos eficientemente.

2. Seleciona-se as VMs alocadas na máquina física encontrada anteriormente para ser realizada a migração para uma nova PM.

3. A alocação das VMs é realizada para a nova máquina física que melhor atenda os requisitos de recursos das máquinas virtuais selecionadas.

4. A migração é realizada para a nova máquina física

Técnicas modernas permitem que a migração entre *hosts* seja realizada enquanto os servidores estiverem em operação (*live migration*) sem uma interrupção significativa do serviço (CLARK et al., 2005). Assim como, quando os servidores estiverem desligados (*non-live migration*).

2.4.1 Formulação do problema de alocação de máquinas virtuais

O problema de alocação de máquinas virtuais é definido na literatura como um problema de otimização combinatória *NP-Hard* (SPEITKAMP; BICHLER, 2010), pois ainda não se tem uma solução algorítmica eficiente para se resolver essa questão (TALBI, 2009). Soluções atuais envolvem a utilização de algoritmos com solução quase ótima para que se possa alcançar resoluções em um tempo computacional razoável, uma vez que o escopo de mapeamentos possíveis de VMs para máquinas *host* torna-se muito grande (TALEBIAN et al., 2020).

Comumente, tem-se como objetivo principal diminuir a quantidade de PMs trabalhando simultaneamente. Assim, reduzindo custos de energia. No entanto, existem também métodos que buscam alcançar, de maneira satisfatória, a otimização de mais de um recurso em simultâneo. Como, por exemplo, tráfego de rede, distribuição térmica, comunicação entre certos conjuntos de VMs, etc (TALEBIAN et al., 2020).

Uma série de algoritmos e métodos foram desenvolvidos para tratar do problema de alocação de máquinas virtuais, tanto em abordagens com objetivo único, quanto em abordagens que visam a otimização de mais de um recurso.

3 Estado da arte

No contexto da computação em nuvem, onde recursos são compartilhados entre um grande número de usuários, o problema de alocação de máquinas virtuais em máquinas físicas tem sido amplamente estudado. Nesse tipo de ambiente, a alocação eficiente de VMs torna-se essencial para garantir que recursos sejam usados sem desperdício e que os níveis acordados de serviço sejam bem atendidos (MASDARI; NABAVI; AHMADI, 2016).

Na literatura encontra-se uma ampla gama de classes de algoritmos com diversas implementações. A seguir segue uma breve apresentação e explicação do funcionamento de algoritmos usados para tratar do problema de alocação de máquinas virtuais.

3.1 *Genetic Algorithm* (GA)

Inspirado no processo de seleção natural, o algoritmo genético (MITCHELL, 1998) (WHITLEY, 1994) é uma meta-heurística bioinspirada que tem por objetivo solucionar problemas de otimização, busca, modelagem, entre outros. Podendo, ainda, ser aplicado a ambientes distribuídos para resolver o problema de alocação de recursos.

O GA consiste em três componentes:

1. Uma população de soluções potenciais (cromossomos);
2. Um conjunto de critérios de avaliação (funções de aptidão);
3. Um conjunto de operadores (*operadores genéticos*) usados para gerar novos cromossomos a partir dos já existentes.

O primeiro passo na execução desse algoritmo é inicializar a população de cromossomos, que pode ser feito aleatoriamente ou de acordo com algum parâmetro predeterminado. Uma vez que a população tenha sido inicializada, cada cromossomo é avaliado usando as funções de aptidão. Essa função atribui um valor numérico a cada cromossomo, que representa o quão próximo aquele cromossomo está de uma solução para o problema.

O próximo passo é selecionar um par de cromossomos da população para realizar o cruzamento. Esse processo é chamado de *Crossover*, processo no qual dois cromossomos pais trocam informações genéticas para produzir cromossomos descendentes que vão herdar características de seus pais. Esses novos cromossomos são, tipicamente, mais aptos que pais e avançam a solução.

Feito isso, os cromossomos descendentes resultantes sofrem mutação, processo aleatório pelo qual são feitas alterações na estrutura de um cromossomo. Essas mudanças podem melhorar ou degradar a aptidão de um dado cromossomo qualquer.

Por fim, os cromossomos recém-gerados são avaliados usando as funções de aptidão e substituem os cromossomos menos aptos na população. Este processo é, então, repetido iterativamente até que uma solução satisfatória seja encontrada, ou algum critério de parada seja atendido.

3.2 *Ant Colony Optimization* (ACO)

A otimização de colônias de formigas (DORIGO; BIRATTARI; STUTZLE, 2006) é uma meta-heurística baseada em população para resolver problemas computacionais que podem ser reduzidos a encontrar bons caminhos através de grafos. A ideia-chave por trás desse algoritmo é que soluções para problemas complexos de otimização podem ser encontradas fazendo com que uma população de soluções potenciais (as *formigas*) criem soluções de forma incremental, enquanto compartilham informações sobre boas soluções entre si (via *trilhas de feromônio*).

Cada formiga escolhe independentemente um caminho através do espaço do problema, tomando decisões locais baseadas em trilhas de feromônios e heurísticas; à medida que uma formiga explora seu ambiente, ela deposita feromônios nas bordas (segmentos de caminho) do caminho de solução que está construindo, fortalecendo a atração dessas bordas para futuras formigas. Com o tempo, as boas soluções tendem a ser construídas com mais frequência do que as ruins, e as soluções ruins acabam sendo ignoradas por toda a colônia.

O exemplo mais famoso de um algoritmo ACO é provavelmente o *Ant System*, proposto por Dorigo, mas muitas variantes foram propostas desde então. Algumas das variantes mais populares incluem o *Elitist Ant System* (EAS), *Max-Min Ant System* (MMAS), *Rank-Based Ant System* (RBAS), *MINMAX Ant System* (MMAS) e o *Ant Colony System* (ACS).

Os algoritmos ACO são classificados como algoritmos de Inteligência de Enxame, pois são inspirados no comportamento de colônias de formigas reais. ACO é uma abordagem particularmente atraente para resolver problemas de otimização porque não requer informações derivadas e pode ser facilmente paralelizada. Em geral, qualquer problema que possa ser reduzido a encontrar o caminho mais curto através de um grafo pode ser resolvido usando um algoritmo ACO.

3.3 *Kernighan-Lin based heuristic for Virtual Machine Placement* (KLVMP)

O algoritmo KLVMP (RODRIGUES, 2019) foi proposto com a ideia de minimizar o número de máquinas físicas ativas hospedando máquinas virtuais e minimizar o custo de comunicação entre conjuntos de VMs. Essa abordagem foi proposta, pois se observou que havia distribuição desigual no tráfego de dados entre *hosts* de um determinado *datacenter*. Assim, procura otimizar esse aspecto trazendo estabilidade a longo prazo e corte de gastos advindos da má utilização de recursos de rede. Sua denominação dá-se por conta de usar o algoritmo de Kernighan-Lin para lidar, principalmente, com os custos de comunicação entre os servidores.

Por apresentar mais de um objetivo, esse algoritmo torna-se um problema de otimização combinatória de múltiplos objetivos. Esses tipos de problemas são classificados como *NP-Hard*, portanto se utiliza de heurísticas para que se possa chegar a um resultado aproximado do ideal.

O funcionamento do KLVMP pode ser dividido nas seguintes etapas:

- Determinar o número de *hosts* utilizados, ordenando-os de modo a encontrar uma solução inicial que atenda ambos os requisitos;
- Utilizar o método *first-fit* para alocar cada *cluster* de VM;
- Aprimorar a solução utilizando-se da heurística baseada no algoritmo de Kernighan-Lin, reduzindo o custo de rede.

No geral, esse algoritmo mostrou-se eficaz no cumprimento de seus objetivos, apresentando resultados melhores do que outros algoritmos propostos para cumprir as mesmas metas.

3.4 *Particle Swarm Optimization* (PSO)

No contexto de alocação de máquinas virtuais, o PSO (REYES-SIERRA; COELLO et al., 2006) é uma abordagem interessante que pode ser usada para otimizar uma ampla variedade de métricas de desempenho. Isso é feito tratando cada VM como uma partícula em um enxame e, em seguida, usando o algoritmo PSO para otimizar iterativamente o posicionamento dessas partículas.

Um dos benefícios de usar o PSO para esse problema é que ele pode lidar com um grande número de variáveis e restrições. Isso o torna adequado para problemas em que há um grande

número de colocações em potencial para cada VM. Outro benefício do PSO é que ele pode encontrar boas soluções em um período de tempo relativamente curto. Isso pode ser importante ao alocar máquinas virtuais, pois o tempo necessário para provisionar e configurar novas VMs pode ser significativo.

A primeira etapa ao usar o PSO para alocar máquinas virtuais é escolher uma métrica de desempenho para otimizar. Isso pode ser algo como minimizar o tempo necessário para provisionar novas VMs ou minimizar a utilização de recursos. Depois que uma métrica for escolhida, a próxima etapa é definir as restrições no posicionamento da VM. Por exemplo, se houver determinados tipos de VMs que não podem ser colocados no mesmo servidor físico, isso precisaria ser representado como uma restrição.

Uma vez que a função objetivo e as restrições tenham sido definidas, o próximo passo é inicializar o enxame. Isso envolve a criação de um conjunto de partículas, cada uma representando um posicionamento potencial para uma VM. As posições iniciais dessas partículas devem ser escolhidas aleatoriamente dentro do espaço viável.

Uma vez que o enxame foi inicializado, o *loop* principal do algoritmo PSO começa. Isso envolve atualizar iterativamente a posição e a velocidade de cada partícula com base em sua posição atual, a posição de sua melhor solução pessoal e a posição da melhor solução global encontrada até agora por qualquer partícula no enxame. Após realizar esse processo por um número especificado de iterações, ou quando algum outro critério de parada for atendido, o algoritmo retornará a melhor solução encontrada por qualquer partícula no enxame como resultado final.

3.5 *Simulated Annealing* (SA)

A técnica do *simulated annealing* (LAARHOVEN; AARTS, 1987) é usada com franquênia em situações onde o espaço de pesquisa é grande demais para ser analisado com abordagens que utilizam busca exaustiva. Essa metaheurística utiliza técnicas probabilistas para tentar chegar a resultados próximos a um ótimo global do problema.

SA baseia-se no processo físico de recozimento, que se refere ao aquecimento de um material e depois em seu lento resfriamento. O objetivo é reduzir o número de defeitos no material. No contexto de SA, essa metáfora é usada para se referir ao processo de tentar encontrar o ótimo global de uma função fazendo pequenas alterações (aquecimento) e depois aceitando ou rejeitando (resfriamento).

O algoritmo começa com um palpite da solução, que pode estar em qualquer lugar no espaço de busca. A *temperatura* determina a probabilidade de o algoritmo aceitar soluções mais pobres à medida que avança. Quando a temperatura é alta, novas soluções que são piores que a solução atual são mais prováveis de serem aceitas. À medida que a temperatura diminui, torna-se menos provável que novas soluções inferiores em relação à solução atual sejam aceitas.

O principal parâmetro que precisa ser definido para que o SA funcione é o cronograma de resfriamento, que determina a rapidez com que a temperatura deve diminuir. Há muitas maneiras diferentes de definir esse cronograma; uma escolha popular é conhecida como resfriamento geométrico, onde a temperatura em cada iteração é igual a algumas constantes vezes a temperatura anterior.

Uma vantagem do SA sobre outras técnicas de otimização global é que ele pode escapar dos mínimos locais. Ou seja, se a solução atual não for o ótimo global, mas estiver próxima de um ótimo local, o SA ainda poderá encontrar o ótimo global fazendo pequenas mudanças que não sejam descendentes em termos do ótimo local. Isso torna o SA especialmente poderoso para funções que possuem muitos mínimos locais.

Uma desvantagem do SA é que ele pode ficar preso em áreas de baixa energia potencial (regiões próximas a um ótimo local), o que pode fazer com que o algoritmo seja executado

lentamente ou não convirja. Outra desvantagem é que o SA requer um ajuste cuidadoso de parâmetros como condições iniciais e cronograma de resfriamento para apresentar um bom desempenho; se esses parâmetros não estiverem configurados corretamente, o SA pode não encontrar uma boa solução ou pode demorar muito para fazê-lo.

3.6 *Tabu Search* (TS)

O algoritmo *Tabu Search* (GENDREAU, 2003) é uma meta-heurística usada para encontrar soluções quase ótimas para problemas de otimização. No contexto de alocação de máquinas virtuais, o TS pode ser usado para determinar a melhor maneira de alocar um conjunto de máquinas virtuais a um conjunto de máquinas físicas.

O algoritmo TS funciona melhorando iterativamente uma solução. A cada iteração, o algoritmo faz uma pequena alteração na solução atual e verifica se a nova solução é melhor que a anterior. Se a nova solução for melhor, ela se torna a nova solução atual; se não for, a alteração é desfeita e outra pequena alteração é feita. Este processo é repetido até que nenhuma melhoria adicional possa ser feita ou um critério de parada pré-especificado seja atendido.

Uma vantagem do TS sobre outras meta-heurísticas é que ele pode escapar de valores ótimos locais, ou seja, soluções sub-ótimas que não são o ótimo global. Isso ocorre porque o TS permite que pequenas alterações sejam feitas na solução atual, mesmo que essas alterações resultem em uma solução temporariamente inferior. Enquanto a tendência geral for de melhoria, o TS acabará por encontrar o ótimo global.

Outra vantagem do TS é sua flexibilidade: por exigir apenas uma pequena alteração a cada iteração, ele pode ser facilmente adaptado para trabalhar com qualquer tipo de problema. Isso torna o TS particularmente adequado para problemas difíceis de definir, como aqueles que envolvem variáveis contínuas ou muitas restrições.

Ao aplicar a alocação de TS para VM, há algumas coisas a serem lembradas. Primeiro, como o TS depende de pequenas mudanças, pode ser uma convergência lenta em problemas muito grandes. Em segundo lugar, o TS pode não encontrar a solução ótima absoluta – mas geralmente encontrará uma solução próxima da ideal. Finalmente, devido à sua flexibilidade, o TS pode ser aplicado a problemas com diferentes tipos de objetivos; por exemplo, ele pode ser usado para minimizar o intervalo de criação (o tempo necessário para concluir todas as tarefas) ou maximizar a utilização de recursos (a porcentagem de tempo durante o qual os recursos estão sendo usados).

3.7 *Greedy Algorithm*

O algoritmo guloso (MASDARI; NABAVI; AHMADI, 2016) é uma abordagem simples e direta para resolver o problema de alocação de máquinas virtuais. A ideia por trás do algoritmo é tomar decisões locais que provavelmente levarão a uma solução globalmente ótima.

Para aplicar o algoritmo guloso ao problema de alocação de máquinas virtuais, primeiro precisamos definir uma métrica para medir a qualidade. Diversas métricas diferentes foram propostas na literatura, incluindo o número de servidores físicos, a quantidade de recursos não utilizados, o consumo de energia e o *makespan*. O número de servidores físicos pode ser utilizado como exemplo.

Existem várias abordagens para esse tipo de heurística, algumas delas são:

- *Best-Fit* (BF)
- *First-Fit* (FF)
- *Next-Fit* (NF)

- *Worst-Fit* (WF)

O algoritmo guloso tem a garantia de encontrar uma solução viável para o problema de posicionamento da máquina virtual. No entanto, não é garantido encontrar uma solução ótima. De fato, o algoritmo guloso pode resultar em uma solução abaixo do ideal se as máquinas virtuais não forem classificadas em ordem decrescente de requisitos de CPU.

3.8 Considerações finais sobre o estado da arte

Existe uma ampla grama de algoritmos que buscam solucionar o problema da alocação de máquinas virtuais em sistemas de computação em nuvem. No geral, como o escopo do problema pode envolver conjuntos de soluções muito grandes, ou até mesmo infinitos, grande parte desses algoritmos utilizam métodos heurísticos ou meta-heurísticos para chegar a soluções aceitáveis.

A abordagem a ser escolhida depende do problema a ser resolvido, assim como também depende dos objetivos a serem alcançados pela implementação. Muitos visam a diminuição no consumo de energia elétrica, outros, a diminuição nos custos de comunicação entre diversos *datacenters*. Os objetivos são diversos e, muitas vezes, múltiplos.

Para a resolução desse problema podemos escolher dentre algoritmos gulosos, algoritmos de abordagem genética, algoritmos bioinspirados, aplicações de modelos estocásticos e estatísticos, entre outros. O estado da arte é amplo e diverso. Porém, não existe ainda um algoritmo que se destaca em todos os quesitos(MANN, 2015). Dessa forma, cada instância de implementação deve ser profundamente estudada para que se possa chegar a soluções viáveis.

4 Comparação de algoritmos

4.1 Visão geral de objetivos

Dada a heterogeneidade do ambiente de alocação de VMs em sistemas em nuvem, surgem, por consequência, variados métodos para a resolução desse problema(MANN, 2015). Cada solução apresentada tem suas próprias peculiaridades e busca atender a diferentes requisitos com implementações díspares de algoritmos objetivando maximizar o uso dos recursos disponíveis.

De acordo com(ATTAOUI; SABIR, 2018), podemos destacar e definir as seguintes classes de objetivos:

- ***Energy Consumption Minimization***: muitas abordagens de alocação de máquinas virtuais se concentram em minimizar o consumo de energia. Apesar de terem o mesmo objetivo, nem todas as abordagens são similares. Algumas implementações tentam minimizar a alimentação de corrente contínua, outras reduzem o número de máquinas físicas ligadas e outras se concentram em minimizar o consumo de energia da rede, etc.

- ***Cost Optimization***: problemas envolvendo a otimização de custos estão relacionados à grande exigência de serviços de computação em nuvem nos *datacenters*, assim como também estão relacionados a como os centros de dados estão geograficamente posicionados. Vários elementos, como o custo de rede, custo de eletricidade, custo de manutenção, custo de infraestrutura, entre outros, constituem parte do problema.

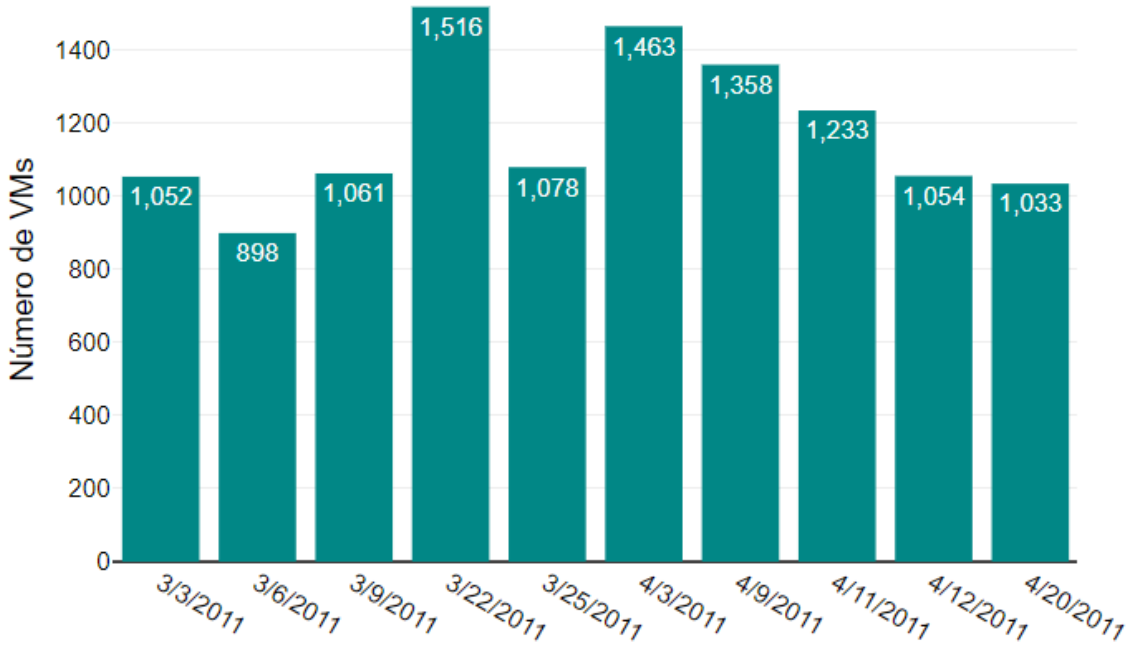
- ***Network Traffic Maximization***: o problema da maximização do tráfego de rede pode ser otimizado por vários fatores, como menor tempo de transferência de dados, otimização da latência média de tráfego, maximização de tráfego de rede e desempenho da rede. Selecionar as máquinas físicas mais adequadas para serem *hosts* de máquinas virtuais específicas de modo a atender esses requisitos é parte crucial da solução desse problema.

- ***Resource Utilization***: para maximizar a utilização de recursos, máquinas virtuais devem ser consolidadas em máquinas físicas de acordo com seus requisitos de cargas de trabalho. Para isso, são utilizados algoritmos de consolidação de VMs de modo a se evitar que existam máquinas ociosas nos *datacenters*, tendo essas que serem desligadas, e garantindo que as máquinas ativas utilizem seus recursos disponíveis da melhor maneira possível para que suas cargas de trabalho estejam sempre próximas do ponto de pico. A consolidação de máquinas virtuais de maneira eficiente implica na melhor utilização de recursos.

- ***QoS Maximization***: provedores de serviços de nuvem devem garantir *QoS* expressos no SLA acordados com seus clientes. Para isso, são aplicados métodos que visam garantir a mais alta qualidade de serviço considerando a maximização de desempenhos dos *datacenters*, alta disponibilidade do serviço e minimização da quantidade de migrações de VMs.

A proposta desse estudo é a comparação de algoritmos para alocação de máquinas virtuais em sistemas de computação em nuvem. Para isso, foram selecionados quatro algoritmos para serem implementados e testados no simulador de ambientes de computação em nuvem *CloudSim*. Para que se pudesse colher dados relevantes, foram selecionados métodos pertencentes a mesma classe de algoritmos (algoritmos gulosos). Todos os algoritmos testados serão analisados quanto ao seu consumo geral de energia e quantidade de migrações realizadas até a consolidação das VMs. Essas medições podem ser obtidas pelo pacote de energia já embutido no próprio *CloudSim*. Esse pacote fornece um modelo para medir o consumo de energia dos *hosts*, dependendo da utilização de componentes crítico do sistema, como CPU, por exemplo (BELOGLAZOV; BUYYA, 2012).

Figura 5: Número de VMs utilizadas por data



Fonte: produzido pelo autor

4.2 Ambiente de simulações

Para realizar os experimentos e coleta de dados foi empregado o simulador *CloudSim* (versão 4.0), ferramenta desenvolvida na Universidade de Melbourne, Austrália. O uso desse simulador é justificado pelo fato de que esse, o *CloudSim*, permite a modelagem de *datacenters*, agendamento de políticas de alocação e consolidação de máquinas virtuais em máquinas físicas, agendamento de recursos de *hosts*, políticas de escalonamento de tarefas, modelagem dos custos, entre outros(WICKREMASINGHE; CALHEIROS; BUYYA, 2010). Configurando-se ideal para o escopo desse trabalho ao possibilitar a implementação e análise de resultados, provenientes da comparação de algoritmos alocação de VMs, em um ambiente controlado onde se pode reproduzir as simulações propostas.

4.2.1 *PlanetLab*

Para que se pudesse obter resultados relevantes foram utilizadas cargas de trabalho executadas nos *datacenters*, geradas a partir de rastros das nuvens reais do *PlanetLab*. Essas cargas de trabalho foram coletadas entre os períodos de março a abril de 2011, subdividindo-se em 10 entradas representando 10 dias distintos, contendo informações de milhares de VMs, cujo número exato e distribuição pode ser acompanhado na figura 5. Esse pacote de dados vem como parte da própria *toolkit* do simulador e pode ser encontrado em *https://github.com/beloglazov/planetlab-workload-traces*.

4.2.2 Configurações gerais

As simulações realizadas nesse experimento foram geradas em ambientes de características heterogêneas. Sendo assim, foram utilizados oito tipos de VMs, cujas especificações podem ser acompanhadas pela tabela 1, e quatro tipos de *hosts* (tabela 2), sendo que o número de *hosts* de cada tipo era igual a 125, totalizando 500 *hosts*.

Todos os testes foram realizados de maneira equivalente, com política de seleção aleatória

	MIPS	RAM	Bandwidth (em Mbit/s)	Tamanho (em GB)
VM tipo 1	2500	870	1000	2.5
VM tipo 2	2000	1740	1000	2.5
VM tipo 3	1000	1740	1000	2.5
VM tipo 4	500	613	1000	2.5
VM tipo 5	2500	870	1000	2.5
VM tipo 6	2200	813	1000	2.5
VM tipo 7	1200	1340	1000	2.5
VM tipo 8	700	1540	1000	2.5

Tabela 1: Tabela com especificações dos tipos de VMs

	MIPS	RAM	Bandwidth (em Gbit/s)	Tamanho (em GB)
Host tipo 1	1860	8192	1000	250
Host tipo 2	2660	8192	1000	250
Host tipo 3	4500	8192	1000	250
Host tipo 4	3000	8192	1000	250

Tabela 2: Tabela com especificações dos tipos de *hosts*

de máquinas virtuais, em um computador *Samsung* modelo 760XBE. Com processador *Intel* i7-8565U, CPU com 8 núcleos 1.8GHz e 8192MB RAM.

4.3 Especificações dos algoritmos

4.3.1 MFPED - *Medium-fit power efficient decreasing algorithm*

Nesse algoritmo proposto por Mobes e Abebe em (MOGES; ABEBE, 2019), primeiro, define-se como regra que:

- Seja L_D o nível de utilização de recursos desejado de um *host*, dado que $L_D = (overload_{thr} + underload_{thr})/2$;
- Onde, $overload_{thr}$ e $underload_{thr}$ são limites de sobrecarga e subcarga dos níveis de utilização de recursos;
- Então, a regra de MF (do inglês, *median-fit*) é definida para favorecer um *host* cujo nível de recurso tenha uma distância mínima de L_D.

Isso significa que um dado *host* configura-se melhor do que outro *host* para a tarefa de alocação de máquinas virtuais, em situações onde a distância do nível de utilização da CPU do nível desejado for menor do que a de outro *host*. Esse comportamento é implementado pelos autores de acordo com o pseudocódigo do algoritmo 1.

4.3.2 PEBFD - *Power efficient best-fit decreasing algorithm*

O segundo algoritmo (algoritmo 2), também proposto por Mobes e Abebe em (MOGES; ABEBE, 2019), tem como entrada a lista de máquinas virtuais a serem migradas, assim como a lista de *hosts* nos quais podem ser alocadas. Inicialmente, essa lista de VMs é organizada em ordem decrescente quanto a utilização de CPU. Essas VMs são alocadas conforme o algoritmo encontra a PM que seria seu melhor *host* e possuir recursos o suficiente para hospedar a VM. Esse processo é feito iterativamente utilizando a regra do melhor ajuste (*best-fit*) até que se

Algorithm 1 MFPED

Input: hostList, vmList
Output: vmPlacement

```
1: L_D ← 0.6; // O nível desejado é definido como 0.6
2: Faz sort da vmList na ordem decrescente de utilização de CPU;
3: for vm em vmList do
4:     minDiff ← MAX; // MAX sendo o número máximo
5:     allocatedHost ← NULL;
6:     for host em hostList do
7:         if host está ativo e tem recursos suficientes para vm then
8:             diff ← |getUtilization(host) − L_D|;
9:             if diff < minDiff then
10:                allocatedHost ← host;
11:                minDiff ← diff;
12:            else
13:                if diff == minDif then
14:                    // PE(.) é a eficiência energética de um host
15:                    if PE(host) > PE(allocatedHost) then
16:                        allocatedHost ← host;
17:                    end if
18:                end if
19:            end if
20:        end if
21:    end for
22:    if allocatedHost ≠ NULL then
23:        adiciona (allocatedHost, vm) em vmPlacement;
24:    end if
25: end for
```

Resultado: vmPlacement

encontre o *host* que garantirá a melhor eficiência energética. Em situações onde a eficiência energética dos *hosts* empata, o *host* que estiver mais próximo de pontos de pico (objetivando maximizar a utilização de recursos) passa a hospedar a VM. Caso um *host* ativo não atinja esses objetivos, *hosts* inativos são pesquisados para cumprir essa tarefa.

4.3.3 *Power aware best-fit decreasing* (PABFD) com política de alocação *Inter quartile range* (IQR)

Para resolver o problema de alocação de VMs objetivando a minimização do consumo de energia, os autores propuseram a implementação do algoritmo *Best-fit decreasing* modificado (BELOGLAZOV; BUYYA, 2012) para que esse, o algoritmo, pudesse escolher os *hosts* de maneira a serem mais eficientes. Esse método (algoritmo 3) ordena as máquinas virtuais em ordem decrescente quanto a sua utilização de recursos de CPU, e aloca VMs de maneira a fornecer o menor aumento no consumo de energia em sua alocação.

Beloglazov e Buyya propuseram utilização do método IQR para ser usado em conjunto com o BFD, que define um limite de utilização superior adaptável. O intervalo interquartil (IQR) pode ser definido como à diferença entre o terceiro e o primeiro quartil, ou seja, $IQR = Q_3 - Q_1$. Isso significa que essa abordagem tem um *breakdown point* de 25%. Assim, sendo vantajoso em relação a abordagens que utilizam o intervalo total, por exemplo. Os autores definem o limite de utilização superior como $T_u = 1 - s * IQR$, sendo que s, o parâmetro de segurança, pode ser definido como o quão agressivamente serão feitas as consolidações dentro de um *host*. Quanto mais baixo o valor, mais agressivamente o algoritmo tentará otimizar a utilização de recursos da PM, porém, elevando o risco de violações de SLA. O contrário vale para quanto maior for o valor do parâmetro de segurança.

Em suma, a implementação desse método (algoritmo 4) verifica a utilização de CPU do *host* para saber se esse, o *host*, está sobrecarregado com base no intervalo interquartil de utilização. *Hosts* sobrecarregados comumente levam a violações do acordo de nível de serviço. Assim, o parâmetro de segurança deve ser calibrado de modo a reduzir essas violações, enquanto visa maximizar a utilização de recursos do *host*. Para isso, o algoritmo adiciona uma entrada de histórico ao *host* para poder fazer comparações futuras e evitar sobrecarga.

4.3.4 *First-Fit* (FF) com política de alocação *median absolute deviation* (MAD)

First-Fit (algoritmo 5) é uma técnica simples, mas que pode ser implementada para resolver eficientemente a tarefa de alocação de máquinas virtuais ao minimizar o tempo gasto no processo de busca e não permitir *over-allocation* nos *hosts* que encontra. Como o nome sugere, o *First-Fit* busca pela primeira máquina que apresentar recursos suficientes para a alocação da VM atual. Caso esse *host* não atenda seus requisitos, o *First-Fit* tenta a alocação no próximo servidor. Esse processo se repete até que seja encontrado um *host* que atenda as demandas da máquina virtual solicitada.

Essa abordagem pode ser combinada com a política de alocação *median absolute deviation* (BELOGLAZOV; BUYYA, 2012) (algoritmo 6) de modo a tornar o processo mais robusto. MAD é uma abordagem de limiar adaptativo e, portanto, define valores que ajustam o limite superior de utilização da CPU dos servidores. Assim, evitando situações que poderiam acarretar violações de SLA em casos onde a utilização de CPU chegaria a 100%. Essas situações acontecem frequentemente em ambientes dinâmicos, onde a carga de trabalho é imprevisível e aplicações diversas compartilham recursos físicos.

A abordagem MAD é um método estatístico especialmente interessante por ser mais resiliente a *outliers* em relação a outras abordagens, como desvio padrão. No MAD, a magnitude das distâncias de um pequeno número de *outliers* é irrelevante.

Matematicamente, MAD pode ser definido como:

Algorithm 2 PEBFD

Input: activeHostList, inactiveHostList, vmList
Output: vmPlacement

```
1: Faz sort da vmList na ordem decrescente de utilização de CPU;
2: for vm em vmList do
3:     bestPowerEfficiency ← MIN;
4:     allocatedHost ← NULL;
5:     for host em activeHostList do
6:         if host tem recursos para alocar VM then
7:             powerEfficiency ← getTotalCPU(host)/getMaxPower(host);
8:             if powerEfficiency > bestPowerEfficiency then
9:                 allocatedHost ← host;
10:                bestPowerEfficiency ← powerEfficiency;
11:            end if
12:        else
13:            if powerEfficiency == bestPowerEfficiency then
14:                if getAvailableCPU(host) < getAvailableCPU(allocatedHost) then
15:                    allocatedHost ← host;
16:                end if
17:            end if
18:        end if
19:    end for
20:    if allocatedHost ≠ NULL then
21:        adiciona (allocatedHost, vm) em vmPlacement;
22:    else
23:        //atribuir allocatedHost de hosts inativos
24:        bestPowerEfficiency ← MIN;
25:        allocatedHost ← NULL;
26:        for host em inactiveHostList do
27:            if host tem recursos para alocar VM then
28:                powerEfficiency ← getTotalCPU(host)/getMaxPower(host);
29:                if powerEfficiency > bestPowerEfficiency then
30:                    allocatedHost ← host;
31:                    bestPowerEfficiency ← powerEfficiency;
32:                else
33:                    if powerEfficiency == bestPowerEfficiency then
34:                        if getAvailableCPU(host) < getAvailableCPU(allocatedHost) then
35:                            allocatedHost ← host;
36:                        end if
37:                    end if
38:                end if
39:            end if
40:        end for
41:        if allocatedHost ≠ NULL then
42:            adiciona (allocatedHost, vm) em vmPlacement;
43:        end if
44:    end if
45: end for
```

Resultado: vmPlacement

Algorithm 3 PABFD

Input: hostList, vmList
Output: vmPlacement

```
1: Faz sort da vmList na ordem decrescente de utilização de CPU;
2: for vm em vmList do
3:     minPower ← MAX
4:     allocatedHost ← NULL
5:     for host em hostList do
6:         if host possui recursos o suficiente para vm then
7:             power ← estimatePower(host,vm)
8:             if power < minPower then
9:                 allocatedHost ← host
10:                minPower←power
11:            end if
12:        end if
13:    end for
14:    if allocatedHost neq NULL then
15:        allocation.add(vm, allocatedHost)
16:    end if
17: end for
```

Resultado: vmPlacement

Algorithm 4 IQR

Input: hostList, vmList, safetyParameter, utilizationThreshold
Output: overUtilizedHost

```
1: V ← vmList;
2: P ← hostList;
3: for p em P do
4:     upperThreshold ← 1 - safetyParameter * getHostUtilizationIqr(host);
5:     adiciona entrada de histórico para o host;
6:     totalRequestedMips ← 0;
7:     for v em V do
8:         totalRequestedMips ← totalRequestedMips + vm.getCurrentRequestedTotalMips();
9:     end for
10:    utilization ← totalRequestedMips / host.getTotalMips();
11:    if utilization > upperThreshold then
12:        Retorna True;//host sobrecarregado
13:    end if
14: end for
15:
16: function getHostUtilizationIqr(dados)
17:    Ordena o array dados;
18:    q1 = round(0.25 * (dados.length + 1)) - 1;
19:    q3 = round(0.75 * (dados.length + 1)) - 1;
20:    Retorna dados[q3] - dados[q1];
21: end function
```

Resultado: host sobrecarregado

$$MAD = mediana_i(|X_i - mediana_j(X_j)|)$$

E o valor do limite superior de utilização pode ser definido por:

$$T_u = 1 - s * MAD$$

Onde s define o quão agressivamente o sistema consolida VMs.

Algorithm 5 FF

Input: hostList, vmList
Output: vmPlacement

```
1: V ← vmList;
2: P ← hostList;
3: for v em V do
4:     for p em P do
5:         if p pode hospedar v then
6:             p ← v;
7:         end if
8:     end for
9: end for
```

Resultado: vmPlacement

4.4 Resultados

Conforme apresentado, foram realizados experimentos com a utilização do simulador *CloudSim* para obtenção e coleta de dados. Todos os quatro modelos foram testados dentro das mesmas condições já especificadas anteriormente. A seguir, serão apresentados os resultados individuais dos testes de cada um dos algoritmos apresentados.

4.4.1 MFPED

Os principais dados coletados dos testes provenientes da implementação do algoritmo de alocação de máquinas virtuais, MFPED, foram: o consumo de energia e a quantidade de migrações que esse método realizou para consolidar as VMs. No gráfico da figura 6, podemos acompanhar seu consumo energético diário no decorrer dos 10 dias, que correspondem ao espaço amostral proveniente da base de dados das nuvens do *PlanetLab*.

Além disso, também podemos acompanhar a quantidade de migrações de máquinas virtuais realizadas para completar o processo de consolidação de VMs em *hosts*, como detalha o gráfico da figura 7.

4.4.2 PEBFD

Mais uma vez, os principais dados coletados dos testes provenientes da implementação do algoritmo de alocação de máquinas virtuais, PEBFD, foram, também: o consumo de energia e a quantidade de migrações que esse método realizou para consolidar as VMs. Podemos acompanhar no gráfico da figura 8, seu consumo energético diário no decorrer dos 10 dias, que correspondem ao espaço amostral proveniente da base de dados das nuvens do *PlanetLab*.

Ainda, também podemos acompanhar a quantidade de migrações de máquinas virtuais realizadas para completar o processo de consolidação de VMs em *hosts*, como detalha o gráfico da figura 9.

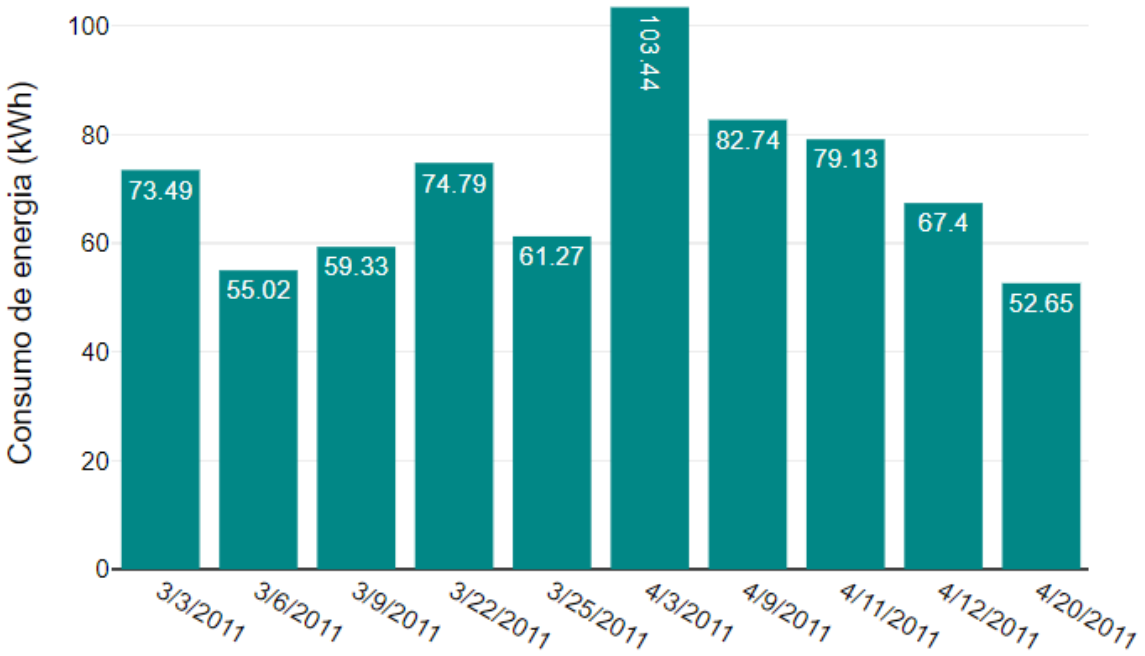
Algorithm 6 MAD

Input: hostList, vmList, safetyParameter, utilizationThreshold
Output: overUtilizedHost

```
1: V ← vmList;
2: P ← hostList;
3: for p em P do
4:     upperThreshold ← 1 - safetyParameter * getHostUtilizationMad(host);
5:     adiciona entrada de histórico para o host;
6:     totalRequestedMips ← 0;
7:     for v em V do
8:         totalRequestedMips ← totalRequestedMips + vm.getCurrentRequestedTotalMips();
9:     end for
10:    utilization ← totalRequestedMips / host.getTotalMips();
11:    if utilization > upperThreshold then
12:        Retorna True;//host sobrecarregado
13:    end if
14: end for
15:
16: function getHostUtilizationMad(dados)
17:    mad ← 0;
18:    if tamanho dados > 0 then
19:        median ← mediana de dados;
20:        deviationSum ← array do tamanho dados;
21:        for todos os elementos do array dados do
22:            deviationSum ← módulo de (mediana - dados[i]);
23:        end for
24:        mad ← mediana(deviationSum);
25:    end if
26:    Retorna mad;
27: end function
```

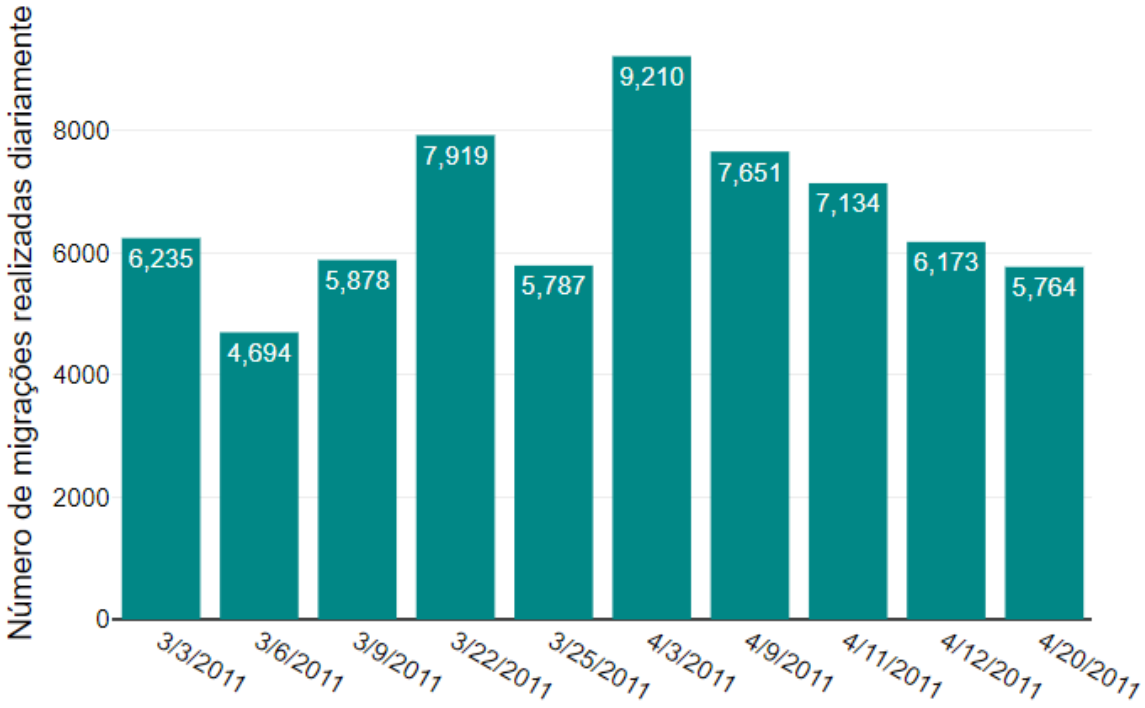
Resultado: host sobrecarregado

Figura 6: Consumo de energia diário do algoritmo MFPED dentro do período amostral



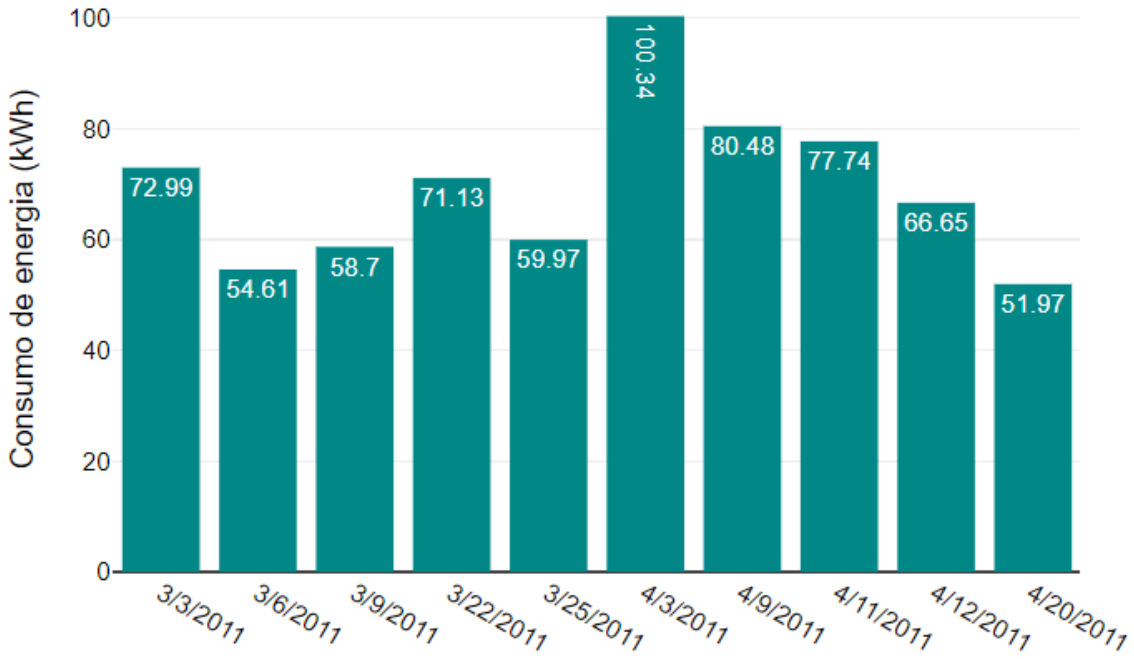
Fonte: produzido pelo autor

Figura 7: Quantidade de migrações diárias do algoritmo MFPED dentro do período amostral



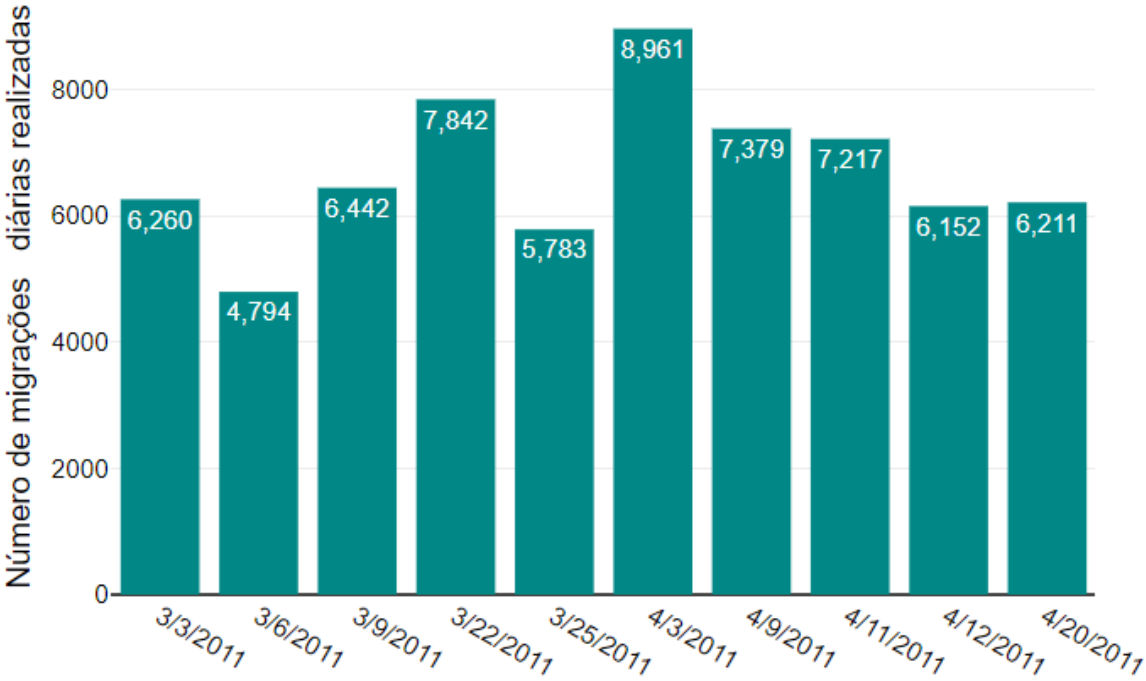
Fonte: produzido pelo autor

Figura 8: Consumo de energia diário do algoritmo PEBFD dentro do período amostral



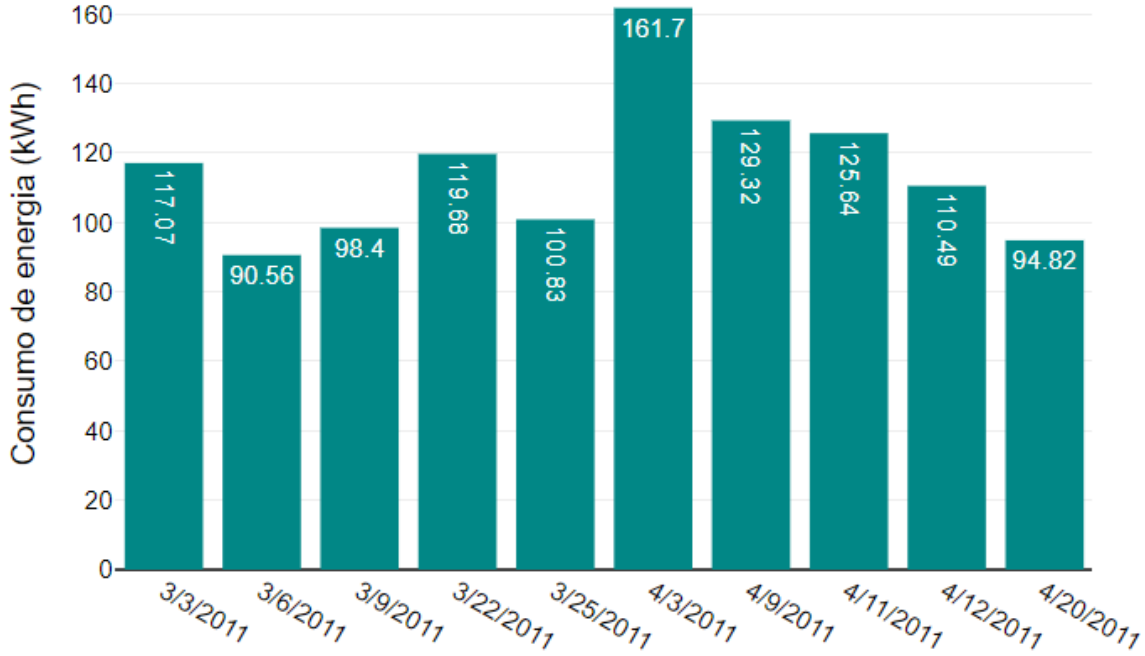
Fonte: produzido pelo autor

Figura 9: Quantidade de migrações diárias do algoritmo PEBFD dentro do período amostral



Fonte: produzido pelo autor

Figura 10: Consumo de energia diário do algoritmo PABFD-IQR dentro do período amostral



Fonte: produzido pelo autor

4.4.3 PABFD-IQR

Para o PABFD-IQR, os principais dados coletados dos testes provenientes da implementação desse algoritmo, foram, mais uma vez: o consumo de energia e a quantidade de migrações que esse método realizou para consolidar as VMs. Podemos acompanhar no gráfico da figura 10, seu consumo energético diário no decorrer dos 10 dias, que correspondem ao espaço amostral proveniente da base de dados das nuvens do *PlanetLab*.

Ainda, também podemos acompanhar a quantidade de migrações de máquinas virtuais realizadas para completar o processo de consolidação de VMs em *hosts*, como detalha o gráfico da figura 11.

4.4.4 FF-MAD

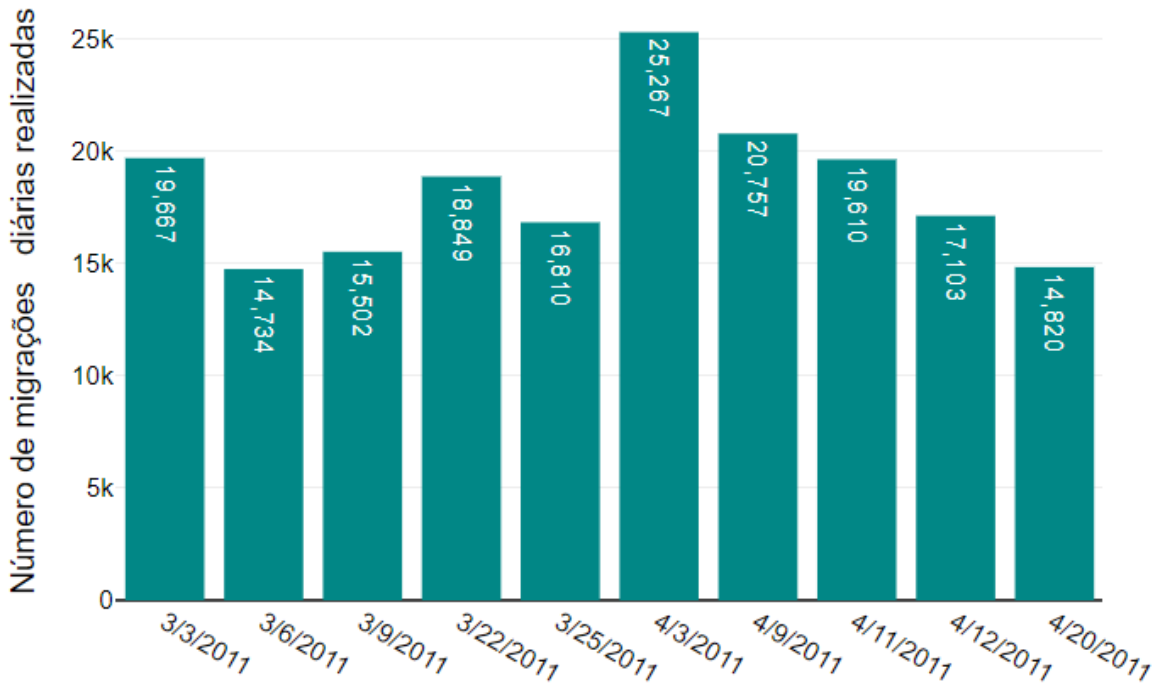
Finalmente, para a abordagem de alocação de máquinas virtuais FF-MAD, como realizado anteriormente, foram coletados: o consumo de energia e a quantidade de migrações que esse método realizou para consolidar as VMs. Podemos acompanhar os resultados obtidos quanto ao consumo diário de energia apresentado por esse algoritmo no decorrer dos dias que correspondem ao espaço amostral proveniente da base de dados das nuvens do *PlanetLab* no gráfico da figura 12.

Além disso, também podemos acompanhar a quantidade de migrações de máquinas virtuais realizadas para completar o processo de consolidação de VMs em *hosts*, como detalha o gráfico da figura 13.

4.4.5 Considerações finais

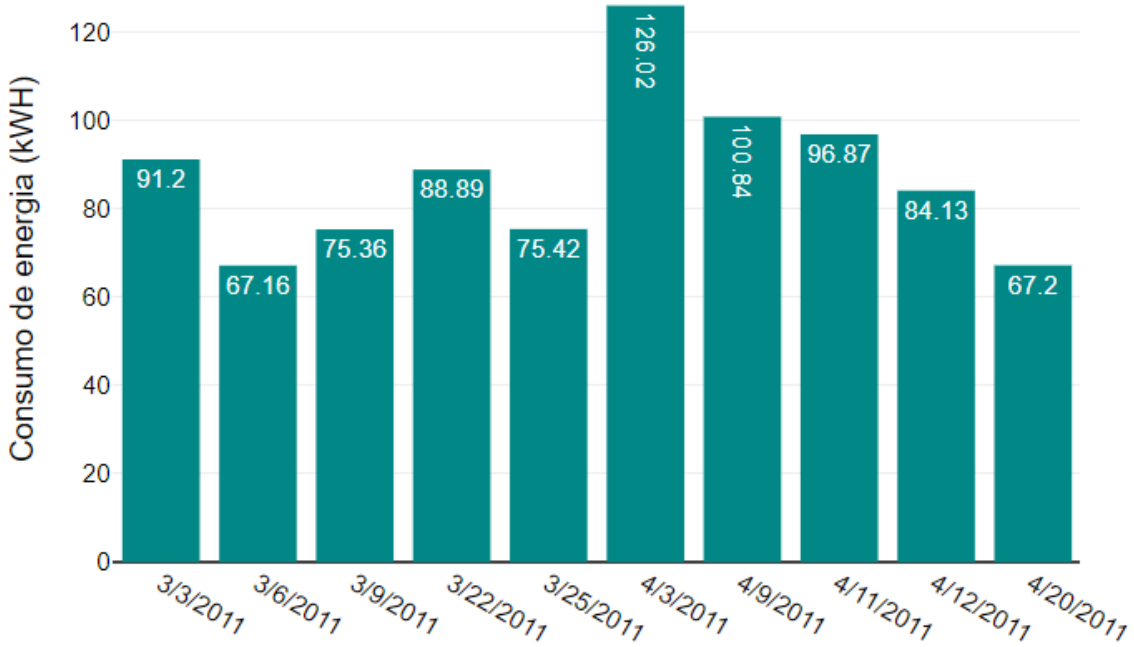
Dos resultados obtidos a partir das simulações realizadas no *CloudSim*, podemos concluir que, dentre os algoritmos apresentados, o menos eficiente em termos de consumo de energia foi o algoritmo PABFD-IQR. Em uma comparação lado a lado com os outros três métodos apresentados, como pode ser observado no gráfico da figura 14 e no gráfico *box plot* 15, fica

Figura 11: Quantidade de migrações diárias do algoritmo PABFD-IQR dentro do período amostral



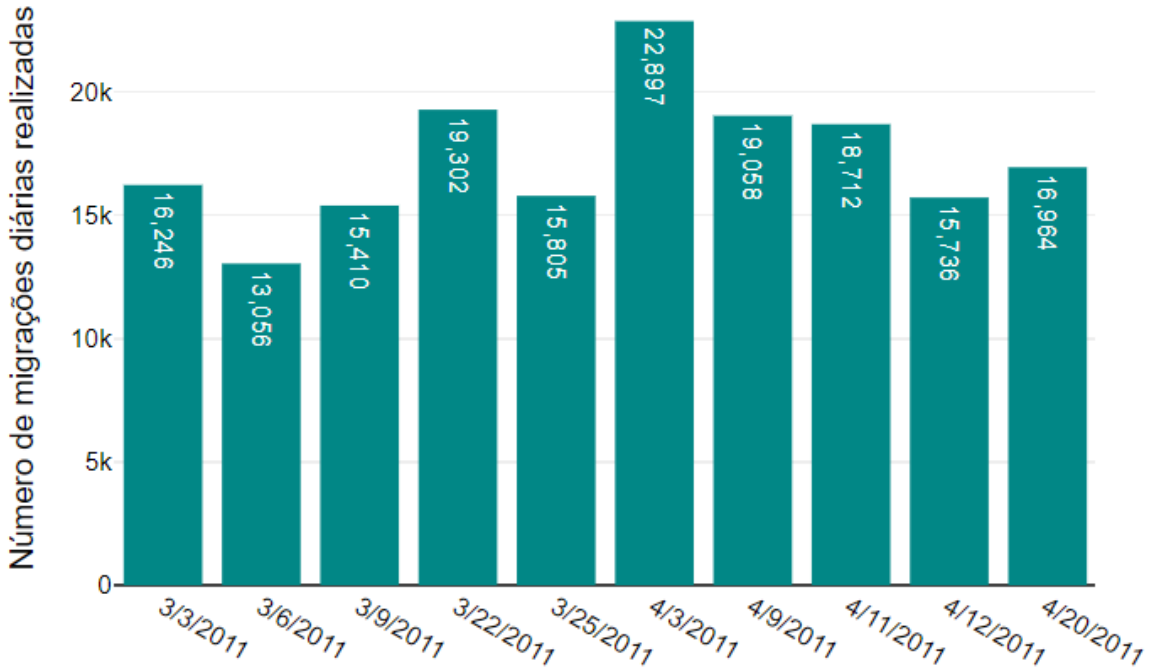
Fonte: produzido pelo autor

Figura 12: Consumo de energia diário do algoritmo FF-MAD dentro do período amostral



Fonte: produzido pelo autor

Figura 13: Quantidade de migrações diárias do algoritmo FF-MAD dentro do período amostral



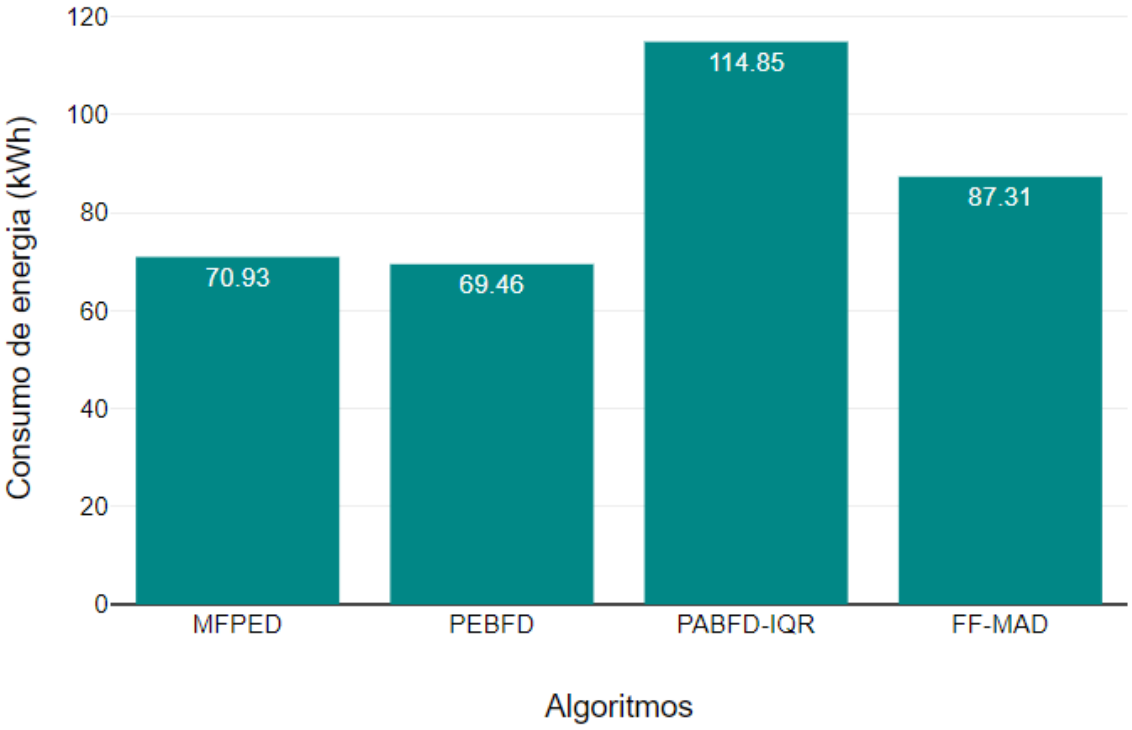
Fonte: produzido pelo autor

evidente que a eficiência energética do PABFD-IQR foi inferior, consumindo cerca de 65% mais energia, em relação ao algoritmo que apresentou o melhor caso, o PEBFD. Esse comportamento é evidenciado, mais uma vez, pelo gráfico apresentado na figura 16, onde se pode fazer uma inferência proveniente da comparação ponto a ponto do consumo de energia diário dos quatro algoritmos. Nesse gráfico, enquanto os algoritmos PEBFD e MFPED apresentam resultados quase idênticos (evidenciado por suas linhas no gráfico se sobrepondo), o PABFD-IQR apresenta resultados que evidenciam sua menor eficiência energética em todos os pontos do gráfico. Já o algoritmo FF-MAD, enquanto se apresenta superior em termos de minimização de consumo de energia em relação ao PABFB-IQR, fica atrás de ambos, PEBFD e MFPED, nesse quesito.

Em relação ao número de migrações de VMs realizadas no experimento durante o período amostral, podemos concluir que, mais uma vez, o algoritmo com menor eficiência foi o PABFD-IQR, como fica evidente da análise do gráfico das figuras 17 e 18. Nesse quesito, o algoritmo que apresentou o melhor caso foi o MFPED, seguido de perto pelo algoritmo PEBFD. Ao compararmos o PABFD-IQR ao MFPED, concluímos que, o PABFD-IQR realizou, em média, cerca de 275% mais migrações para consolidar máquinas virtuais em servidores do que o MFPED, constituindo uma diferença significativa. O FF-MAD, mais uma vez, consolida-se na terceira posição em termos de eficiência. No entanto, agora, apresenta desempenho muito similar ao do desempenho apresentado pelo algoritmo menos eficiente, o PABFD-IQR, tendo sua *performance* diferindo em, apenas, 5% do mesmo. Esses comportamentos podem ser acompanhados, também, pelo gráfico da figura 19. Desse gráfico fica evidente o quão próxima é a eficiência, em termos de número de migrações de VMs, do algoritmo MFPED e do algoritmo PEBFD. Também podemos observar um comportamento muito mais próximo de ambos os algoritmos PABFD-IQR e FF-MAD nesse gráfico, em relação ao que foi observado no gráfico da figura 16 quanto ao consumo de energia, onde o FF-MAD teve desempenho consideravelmente melhor.

Enfim, podemos concluir que, apesar de que fica claro qual algoritmo teve a pior taxa de minimização de consumo elétrico e minimização de migrações de VMs, fica difícil definir, entre o PEBFD e o MFPED, qual dos algoritmos teve o melhor desempenho geral. Em termos de

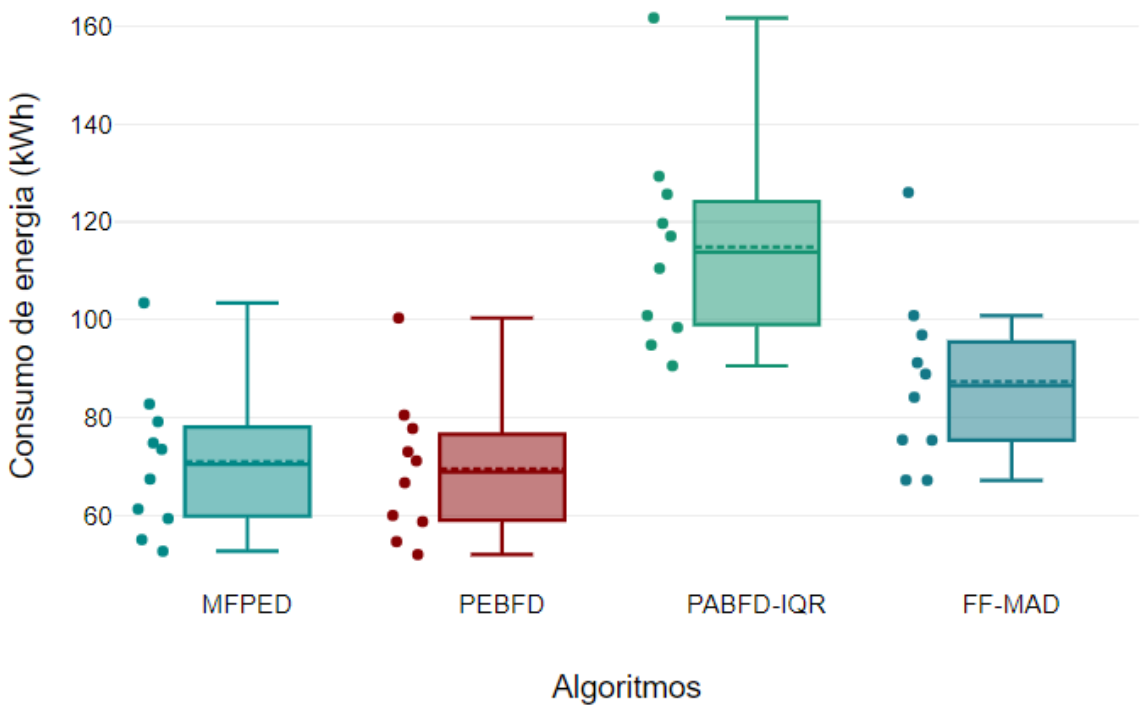
Figura 14: Consumo médio de energia dos algoritmos dentro do período amostral



Fonte: produzido pelo autor

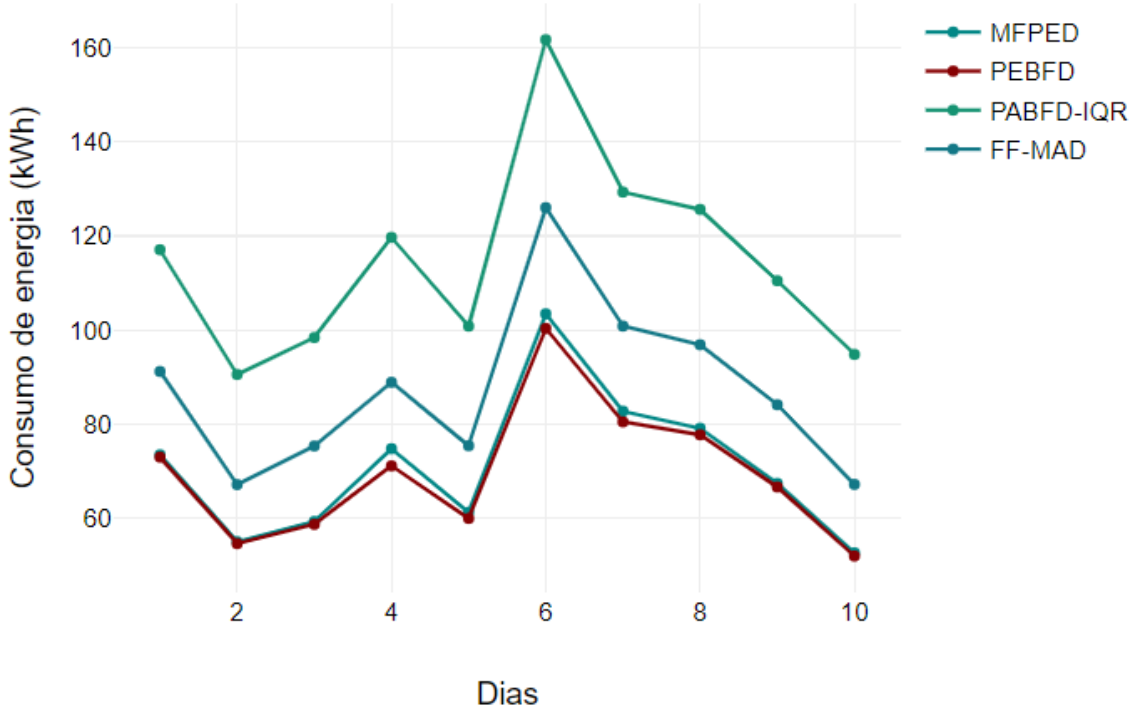
minimização de consumo de energia, o PEBFD apresenta a menor média de consumo dentro do período amostral representado, no entanto, essa diferença em relação ao MFPED configura-se, apenas, cerca de 1%. Em contrapartida, o algoritmo MFPED apresenta o menor número médio de migrações de máquinas virtuais dentre os quatro métodos testados. Porém, essa diferença contabiliza, mais uma vez, apenas cerca de 1% quando comparado ao segundo algoritmo com menor taxa de migrações realizadas, o PEBFD. Diferença essa, desprezível. Por fim, podemos concluir que, dos resultados obtidos no experimento, o FF-MAD consolida-se fortemente na terceira posição quanto a minimização de consumo de energia e quantidade de migrações realizadas para consolidar VMs.

Figura 15: Boxplot do consumo de energia dos algoritmos dentro do período amostral



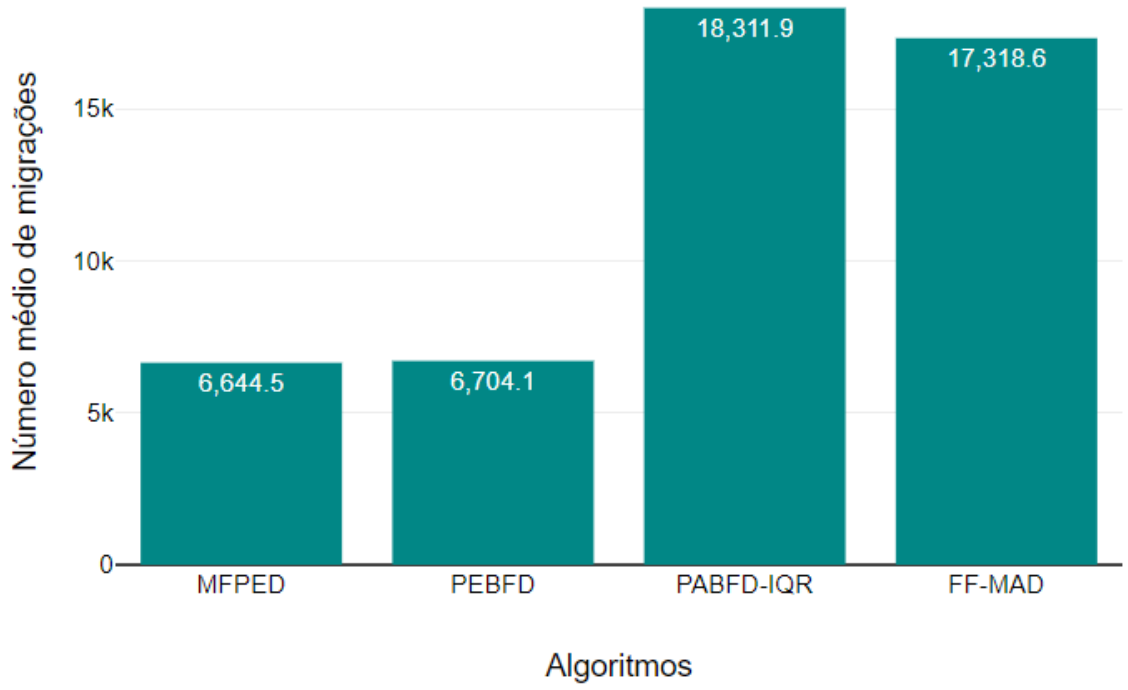
Fonte: produzido pelo autor

Figura 16: Consumo de energia diário dos algoritmos dentro do período amostral



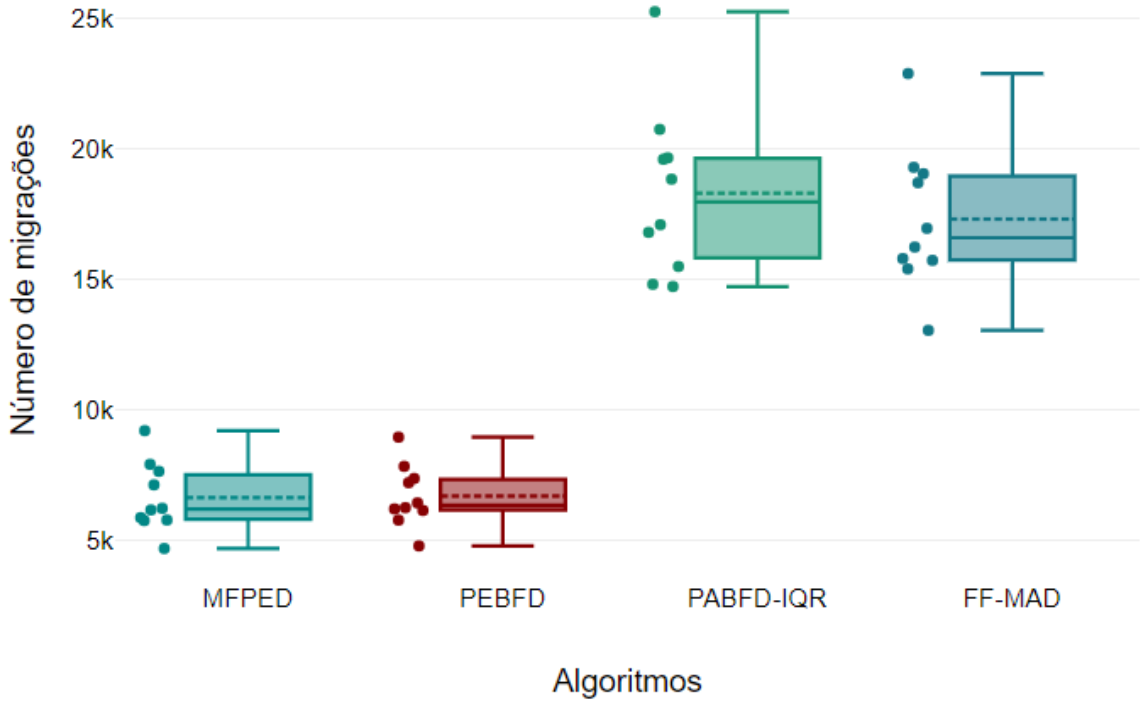
Fonte: produzido pelo autor

Figura 17: Quantidade média de migrações realizadas pelos algoritmos dentro do período amostral



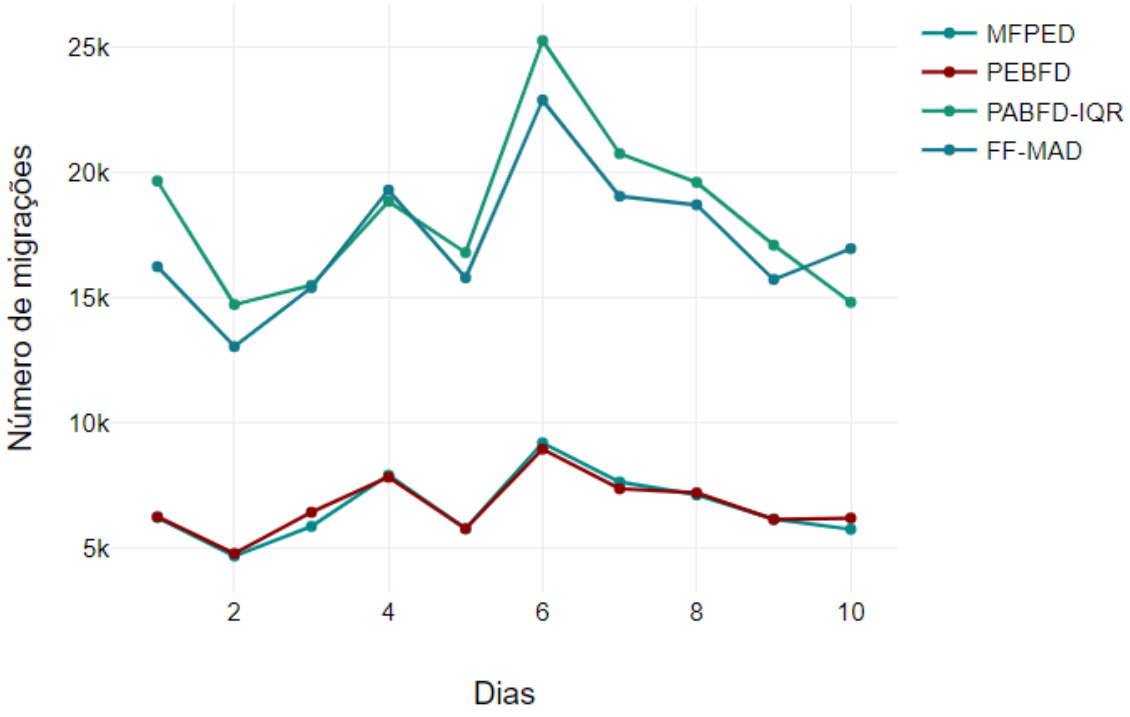
Fonte: produzido pelo autor

Figura 18: Boxplot do número de migrações de VMs realizadas pelos algoritmos dentro do período amostral



Fonte: produzido pelo autor

Figura 19: Número de migrações diárias realizadas pelos algoritmos dentro do período amostral



Fonte: produzido pelo autor

5 Conclusões

Sistemas de computação em nuvem estão se tornando cada vez mais populares por conta de suas vantagens, como, por exemplo, acesso sob demanda, conveniência e flexibilidade. Esse modelo tecnológico permite acesso a recursos compartilhados de computação por meio de uma conexão de rede a custos atraentes. Contudo, para poder oferecer esse tipo de serviço, tais sistemas devem, por meio da virtualização, ser capazes de alocar recursos para usuários em ambientes distribuídos.

O problema da alocação de máquinas virtuais em sistemas de computação em nuvem é de grande importância para que se possa construir sistemas eficientes e econômicos, objetivando a minimização de custos em simultâneo com a disponibilidade de serviços de alta qualidade para usuários. Por se tratar de um problema de otimização combinatória, soluções exatas não costumam ser viáveis para lidar com o ambiente dinâmico e heterogênico dos sistemas em nuvem. Dessa forma, uma variedade de algoritmos foram propostos na literatura para tentar sanar esse problema.

Algoritmos que buscam solucionar o problema da alocação de VMs em sistemas de computação em nuvem podem ser classificados em diversas classes, dependendo de sua implementação e objetivos. Esses algoritmos precisam ser rápidos, eficientes e escaláveis, para poderem lidar com a alta demanda de recursos desse tipo de ambiente. Muitas vezes, também, é necessário que esses algoritmos consigam realizar migrações em tempo de execução. Essa é uma das principais operações de virtualização, contribuindo para que o sistema torne-se flexível e consiga cumprir com os requisitos de sistemas e acordos de nível de serviço firmados com os usuários.

Esse trabalho visou apresentar os principais conceitos relacionados a sistemas distribuídos e sistemas de computação em nuvem. Bem como apresentou algoritmos propostos no estado da arte para que se fosse possível solucionar a questão da alocação de VMs em máquinas físicas. E, também, tratar da questão da migração de máquinas virtuais.

Dessa maneira, visando encontrar métodos que solucionem a questão da alocação eficientemente, foram propostos experimentos para que se pudesse testar e comparar métodos que tivessem como objetivo primário a minimização do consumo de energia e maximização da eficiência de migrações de VMs. Para isso, foi utilizado o *software* de simulação, *CloudSim*, que permite a modelagem das principais variáveis encontradas nos ambientes de nuvem, assim como a coleta de dados precisos. Almejando resultados condizentes e relevantes, as simulações contaram com a utilização de dados e configurações reais de *datacenters* provenientes das nuvens do *PlanetLab*.

Como resultado desses experimentos constatou-se que, dos quatro algoritmos testados, o algoritmo de menor desempenho foi o algoritmo *PABFD-IQR*, seguido pelo algoritmo FF-MAD. Os outros dois, o MFPED e o PEBFD, obtiveram resultados muito similares e satisfatórios.

5.1 Trabalhos futuros

O objetivo desse trabalho foi implementar e comparar técnicas de alocação de máquinas virtuais em sistemas de computação em nuvem. Especificamente, o foco dessa pesquisa esteve na minimização do consumo de energia e minimização da quantidade de migrações de VMs. Dito isso, existem diversas outras métricas que podem ser consideradas e comparadas como, por exemplo, a quantidade de *hosts* simultaneamente em uso, custos de comunicação, complexidade de tempo para consolidação de VMs, entre outros. Sendo assim, como trabalho futuro, os objetivos e restrições desse projeto podem ser alterados de modo a se mensurar e comparar tais aspectos e verificar a eficácia dos algoritmos implementados sob uma diferente perspectiva.

Além disso, outra linha de trabalho que pode ser seguida envolve a implementação das demais abordagens apresentadas no estado da arte. Implementações futuras podem fazer comparações lado a lado das demais classes de algoritmos e tirar conclusões quanto a eficácia de cada uma delas, levando em consideração objetivos e restrições variados para encontrar algoritmos mais eficientes que consigam resolver esse problema.

Referências

ATTAOUI, W.; SABIR, E. Multi-criteria virtual machine placement in cloud computing environments: a literature review. *arXiv preprint arXiv:1802.05113*, 2018.

BADGER, M. L. et al. *Cloud computing synopsis and recommendations*. [S.l.]: National Institute of Standards & Technology, 2012.

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.

BOSS, G. et al. Cloud computing. technical report. *IBM high performance on demand solutions*, p. 10–08, 2007.

BUYYA, R.; BROBERG, J.; GOSCINSKI, A. M. *Cloud computing: Principles and paradigms*. [S.l.]: John Wiley & Sons, 2010.

CALHEIROS, R. N. et al. Cloudsim: a toolkit for modeling and simulation of cloud computing environments and evaluation of resource provisioning algorithms. *Software: Practice and experience*, Wiley Online Library, v. 41, n. 1, p. 23–50, 2011.

CLARK, C. et al. Live migration of virtual machines. In: *Proceedings of the 2nd conference on Symposium on Networked Systems Design & Implementation-Volume 2*. [S.l.: s.n.], 2005. p. 273–286.

COULOURIS, G. F.; DOLLIMORE, J.; KINDBERG, T. *Distributed systems: concepts and design*. [S.l.]: pearson education, 2005. 1 p.

DASHTI, S. E.; RAHMANI, A. M. Dynamic vms placement for energy efficiency by pso in cloud computing. *Journal of Experimental & Theoretical Artificial Intelligence*, Taylor & Francis, v. 28, n. 1-2, p. 97–112, 2016.

DESAI, A. et al. Hypervisor: A survey on concepts and taxonomy. *International Journal of Innovative Technology and Exploring Engineering*, Citeseer, v. 2, n. 3, p. 222–225, 2013.

DORIGO, M.; BIRATTARI, M.; STUTZLE, T. Ant colony optimization. *IEEE computational intelligence magazine*, IEEE, v. 1, n. 4, p. 28–39, 2006.

GAO, Y. et al. A multi-objective ant colony system algorithm for virtual machine placement in cloud computing. *Journal of computer and system sciences*, Elsevier, v. 79, n. 8, p. 1230–1242, 2013.

GENDREAU, M. An introduction to tabu search. In: *Handbook of metaheuristics*. [S.l.]: Springer, 2003. p. 37–54.

GOUDARZI, H.; GHASEMAZAR, M.; PEDRAM, M. Sla-based optimization of power and migration cost in cloud computing. In: *2012 12th IEEE/ACM International Symposium on Cluster, Cloud and Grid Computing (ccgrid 2012)*. [S.l.: s.n.], 2012. p. 172–179.

Virtualization for security. In: HOOPES, J. (Ed.). Boston: Syngress, 2009. p. 1–43. ISBN 978-1-59749-305-5. Disponível em: <https://www.sciencedirect.com/science/article/pii/B9781597493055000013>.

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

LYNCH, M. The cloud wars: $100+ billion at stake. *Merrill Lynch*, 2008.

LYNCH, N. A. *Distributed algorithms*. [S.l.]: Elsevier, 1996.

MANN, Z. Á. Allocation of virtual machines in cloud data centers—a survey of problem models and optimization algorithms. *Acm Computing Surveys (CSUR)*, ACM New York, NY, USA, v. 48, n. 1, p. 1–34, 2015.

MANN, Z. A.; SZABÓ, M. Which is the best algorithm for virtual machine placement optimization? *Concurrency and Computation: Practice and Experience*, Wiley Online Library, v. 29, n. 10, p. e4083, 2017.

MASDARI, M.; NABAVI, S. S.; AHMADI, V. An overview of virtual machine placement schemes in cloud computing. *Journal of Network and Computer Applications*, Elsevier, v. 66, p. 106–127, 2016.

MEISNER, D.; GOLD, B. T.; WENISCH, T. F. Powernap: eliminating server idle power. *ACM SIGARCH Computer Architecture News*, ACM New York, NY, USA, v. 37, n. 1, p. 205–216, 2009.

MELL, P.; GRANCE, T. et al. The nist definition of cloud computing. Computer Security Division, Information Technology Laboratory, National . . . , 2011.

MITCHELL, M. *An introduction to genetic algorithms*. [S.l.]: MIT press, 1998.

MOGES, F. F.; ABEBE, S. L. Energy-aware vm placement algorithms for the openstack neat consolidation framework. *Journal of Cloud Computing*, Springer, v. 8, n. 1, p. 1–14, 2019.

PEREZ-BOTERO, D.; SZEFER, J.; LEE, R. B. Characterizing hypervisor vulnerabilities in cloud computing servers. In: *Proceedings of the 2013 international workshop on Security in cloud computing*. [S.l.: s.n.], 2013. p. 3–10.

RASHID, A.; CHATURVEDI, A. Cloud computing characteristics and services: a brief review. *International Journal of Computer Sciences and Engineering*, v. 7, n. 2, p. 421–426, 2019.

REYES-SIERRA, M.; COELLO, C. C. et al. Multi-objective particle swarm optimizers: A survey of the state-of-the-art. *International journal of computational intelligence research*, v. 2, n. 3, p. 287–308, 2006.

RODRIGUES, J. A. M. Otimização de alocação de máquinas virtuais em datacenter heterogêneo de sistema de computação em nuvem. Universidade Estadual Paulista (Unesp), 2019.

SAJID, M.; RAZA, Z. Cloud computing: Issues & challenges. In: *International Conference on Cloud, Big Data and Trust*. [S.l.: s.n.], 2013. v. 20, n. 13, p. 13–15.

SPEITKAMP, B.; BICHLER, M. A mathematical programming approach for server consolidation problems in virtualized data centers. *IEEE Transactions on services computing*, IEEE, v. 3, n. 4, p. 266–278, 2010.

SRINIVAS, J.; REDDY, K. V. S.; QYSER, A. M. Cloud computing basics. *International journal of advanced research in computer and communication engineering*, v. 1, n. 5, p. 343–347, 2012.

STEEN, M. V.; TANENBAUM, A. Distributed systems principles and paradigms. *Network*, v. 2, 2002.

SUNYAEV, A. Cloud computing. In: _____. *Internet Computing: Principles of Distributed Systems and Emerging Internet-Based Technologies*. Cham: Springer International Publishing, 2020. p. 195–236. ISBN 978-3-030-34957-8. Disponível em: <https://doi.org/10.1007/978-3-030-34957-8_7>.

TALBI, E.-G. *Metaheuristics: from design to implementation*. [S.l.]: John Wiley & Sons, 2009.

TALEBIAN, H. et al. Optimizing virtual machine placement in iaas data centers: taxonomy, review and open issues. *Cluster Computing*, Springer, v. 23, n. 2, p. 837–878, 2020.

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

WICKREMASINGHE, B.; CALHEIROS, R. N.; BUYYA, R. Cloudanalyst: A cloudsim-based visual modeller for analysing cloud computing environments and applications. In: IEEE. *2010 24th IEEE international conference on advanced information networking and applications*. [S.l.], 2010. p. 446–452.

YANAGISAWA, H.; OSOGAMI, T.; RAYMOND, R. Dependable virtual machine allocation. In: IEEE. *2013 Proceedings IEEE INFOCOM*. [S.l.], 2013. p. 629–637.

ZHANG, Q.; CHENG, L.; BOUTABA, R. Cloud computing: state-of-the-art and research challenges. *Journal of internet services and applications*, SpringerOpen, v. 1, n. 1, 2010.

ZISSIS, D.; LEKKAS, D. Addressing cloud computing security issues. *Future Generation computer systems*, Elsevier, v. 28, n. 3, p. 583–592, 2012.