UNIVERSIDADE FEDERAL DE SÃO CARLOS

CENTRO DE CIÊNCIAS EXATAS E DE TECNOLOGIA

PROGRAMA DE PÓS-GRADUAÇÃO EM CIÊNCIA DA COMPUTAÇÃO

ESCALABILIDADE DE APLICAÇÕES BAG-OF-TASKS EM PLATAFORMAS DISTRIBUÍDAS HETEROGÊNEAS

JAIME FREIRE DE SOUZA

ORIENTADOR: PROF. DR. HERMES SENGER

São Carlos – SP

Abril/2019

UNIVERSIDADE FEDERAL DE SÃO CARLOS

CENTRO DE CIÊNCIAS EXATAS E DE TECNOLOGIA

PROGRAMA DE PÓS-GRADUAÇÃO EM CIÊNCIA DA COMPUTAÇÃO

ESCALABILIDADE DE APLICAÇÕES BAG-OF-TASKS EM PLATAFORMAS DISTRIBUÍDAS HETEROGÊNEAS

JAIME FREIRE DE SOUZA

Dissertação apresentada ao Programa de Pós-Graduação em Ciência da Computação da Universidade Federal de São Carlos, como parte dos requisitos para a obtenção do título de Mestre em Ciência da Computação, área de concentração: Sistemas de Computação
Orientador: Prof. Dr. Hermes Senger

São Carlos – SP

Abril/2019

UNIVERSIDADE FEDERAL DE SÃO CARLOS

Centro de Ciências Exatas e de Tecnologia
Programa de Pós-Graduação em Ciência da Computação

Folha de Aprovação

Assinaturas dos membros da comissão examinadora que avaliou e aprovou a Defesa de Dissertação de Mestrado do candidato Jaime Freire de Souza, realizada em 08/04/2019:

Paulo Matias

Prof. Dr. Paulo Matias
UFSCar

Prof. Dr. Hermes Senger
UFSCar

Prof. Dr. Cesar Augusto Cavalheiro Marcondes
ITA

Certifico que a defesa realizou-se com a participação à distância do(s) membro(s) Hermes Senger, Cesar Augusto Cavalheiro Marcondes e, depois das arguições e deliberações realizadas, o(s) participante(s) à distância está(ao) de acordo com o conteúdo do parecer da banca examinadora redigido neste relatório de defesa.

AGRADECIMENTOS

Serei eternamente grato a todos que contribuíram para que eu pudesse concluir essa jornada, a saber:

A Deus pelo dom da vida e por me abençoar e ajudar em cada etapa até aqui.

Aos meus pais Onofre e Zizália (*in memoriam*) que sempre me apoiaram nos meus estudos e são a base de tudo. E a toda a minha família.

Ao meu orientador, Prof. Dr. Hermes Senger, pelas importantes contribuições, paciência e ensinamentos.

Aos colegas de laboratório e de departamento pelo apoio prestado além da pesquisa, pela amizade, e pelas boas conversas.

Aos professores e funcionários do Departamento de Computação e do Programa de Pós-Graduação em Ciência da Computação da UFSCar, e a todas as pessoas, que direta ou indiretamente, contribuíram para o sucesso deste trabalho.

A Cloud@UFSCar pelo fornecimento de recursos computacionais para a execução dos experimentos.

E por fim, à CAPES pelo apoio financeiro. O presente trabalho foi realizado com apoio da Coordenação de Aperfeiçoamento de Pessoal de Nível Superior – Brasil (CAPES) – Código de Financiamento 001.

”A persistência é o menor caminho do êxito”

Charles Chaplin

RESUMO

Aplicações *Bag-of-Tasks* (BoT) são aplicações paralelas compostas de tarefas independentes (ou seja, embaraçosamente paralelas), que não se comunicam entre si, podem depender de um ou mais arquivos de entrada e podem ser executadas em qualquer ordem. As aplicações BoT são muito frequentes em diversas áreas e comumente executadas em grandes sistemas de computação distribuída, como nas grades computacionais ou na nuvem. Este trabalho estuda a escalabilidade de aplicações BoT executando em grandes sistemas de computação distribuída heterogêneos organizados como uma plataforma mestre-escravo. Os resultados mostram que plataformas mestre-escravo heterogêneas podem alcançar limites de escalabilidade mais altos que as plataformas homogêneas para a execução de aplicações BoT, quando a capacidade computacional dos nós individuais da plataforma homogênea é fixa. No entanto, quando nós individuais da plataforma homogênea podem escalar verticalmente, é mostrado neste trabalho que plataformas homogêneas apresentam escalabilidade próxima do linear.

Palavras-chave: Computação distribuída, Aplicações *Bag-of-Tasks*, Escalabilidade, Sistemas heterogêneos, Computação em nuvem

ABSTRACT

Bag-of-Tasks (BoT) applications are parallel applications composed of independent (*i.e.*, embarrassingly parallel) tasks, which do not communicate with each other, may depend upon one or more input files, and can be executed in any order. BoT applications are very frequent in several scientific areas, and it is the ideal application class for execution on large distributed computing systems composed of hundreds to many thousands of computational resources. This paper focusses on the scalability of BoT applications running on large heterogeneous distributed computing systems organized as a master-slave platform. The results demonstrate that heterogeneous master-slave platforms can achieve higher scalability than homogeneous platforms for the execution of BoT applications, when the computational power of individual nodes in the homogeneous platform is fixed. However, when individual nodes of the homogeneous platform can scale-up, experiments show that master-slave platforms can achieve near linear speedups.

Keywords: Distributed Computing, Bag-of-Tasks Applications, Scalability, Heterogeneous systems, Cloud computing

LISTA DE FIGURAS

LISTA DE TABELAS

GLOSSÁRIO

API – *Application Programming Interface*

ASF – *Apache Software Foundation*

AWS – *Amazon Web Services*

BoT – *Bag-of-Tasks*

CCR – *Communication to Computation Ratio*

CPU – *Central Processing Unit*

FLOPS – *Floating-point Operations Per Second*

FPGA – *Field Programmable Gate Array*

GPU – *Graphics Processing Unit*

GWA – *Grid Workloads Archive*

HDD – *Hard Disk Drive*

HOT – *Heat Orchestration Template*

HPC – *High Performance Computing*

IFA – *Input File Affinity*

IP – *Internet Protocol*

IaaS – *Infrastructure as a Service*

IoT – *Internet of Things*

KVM – *Kernel-based Virtual Machine*

LXC – *Linux Containers*

MIPS – *Million Instruction Per Second*

NASA – *National Aeronautics and Space Administration*

NIST – *National Institute of Standards and Technology*

NWS – *Network Wheather Service*

P2P – *Peer-to-Peer*

PaaS – *Platform as a Service*

REST – *Representational State Transfer*

RPC – *Remote Procedure Call*

S4U – *Simgrid For You*

SSD – *Solid-state Drive*

SaaS – *Software as a Service*

TCP – *Transmission Control Protocol*

TI – *Tecnologia da Informação*

UML – *Unified Modeling Language*

VM – *Virtual Machine*

WSRF – *Web Services Resource Framework*

XBT – *Extensible Box of Tools*

XML – *Extensible Markup Language*

vCPU – *Virtual CPU*

SUMÁRIO

Capítulo 1

INTRODUÇÃO

1.1 Contexto

O presente trabalho apresenta um estudo sobre a execução de aplicações *Bag-of-tasks* (BoT) em plataformas computacionais distribuídas compostas por recursos heterogêneos. Aplicações BoT representam uma parcela significativa das cargas de trabalho científicas executadas corriqueiramente sobre plataformas computacionais distribuídas, incluindo os portais científicos (*science-gateways*), *clusters*, grades computacionais e nuvem (SILVA; GLATARD, 2012; IOSUP et al., 2008). Aplicações BoT são compostas por tarefas sequenciais e independentes, que não se comunicam entre si, tal como observado em aplicações embaraçosamente paralelas. A entrada para cada tarefa é composta de um ou mais arquivos, e um mesmo arquivo pode ser a entrada para mais de uma tarefa. Cada tarefa gera seu próprio conjunto de arquivos de saída, que podem ser compostos de um ou mais arquivos.

Exemplos de aplicações BoT incluem simulações de Monte Carlo, buscas massivas (como quebra de chave), aplicações de manipulação de imagens e algoritmos de mineração de dados. Elas são frequentes em áreas como astronomia, física de alta energia, mineração de dados (SENGER et al., 2007), bioinformática (CASTRO et al., 2017), e muitas outras. Daí a importância de estudar a escalabilidade de aplicações BoT sob o ponto de vista prático. Sob o ponto de vista teórico, BoT são aplicações paralelas que não possuem dependências. Isto as torna particularmente interessantes, pois se poderia conjecturar que, sob circunstancias específicas, os limites máximos de escalabilidade das aplicações BoT poderiam ser limites absolutos de uma dada arquitetura, para quaisquer aplicações paralelas.

A escalabilidade é uma propriedade relacionada a uma combinação algoritmo-máquina, em vez de ser uma propriedade exclusiva da arquitetura da máquina ou do algoritmo (SUN; RO-

VER, 1994). A análise de escalabilidade permite a identificação dos gargalos de desempenho do sistema, um melhor entendimento do efeito desses gargalos sobre a escalabilidade do sistema e também oferece subsídios para mitigar esses gargalos e promover a melhoria da escalabilidade. Os ganhos podem ser consideráveis, como por exemplo atingir *speedups* da ordem de 30 vezes para aplicações de mineração de dados implementadas como BoT executadas em um *cluster* de PCs (SENGER et al., 2007) ou acima de 50 para aplicação de alinhamento de sequências (bioinformática) executando na nuvem (CASTRO et al., 2017).

1.2 Motivação e Objetivos

As aplicações *Bag-of-Tasks* são muito comuns em sistemas de larga escala. Por isso, são bastante utilizadas pelas comunidades científica e industrial. Há vários estudos sobre a escalabilidade de aplicações BoT em plataformas de computação distribuída, como por exemplo em (SILVA; SENGER, 2009, 2011, 2015; SENGER; SILVA, 2012). Nesses estudos, as arquiteturas eram todas homogêneas, ou seja, eram compostas de recursos homogêneos. Contudo, verifica-se que a grande parte das aplicações de alto desempenho são atualmente executadas em sistemas de computação heterogêneos. Nos centros de computação de alto desempenho, é comum a presença de várias gerações de máquinas sendo utilizadas para a execução das cargas de trabalho corriqueiras.

Plataformas distribuídas, incluindo a nuvem, federações de nuvem e grades computacionais disponibilizam recursos (em geral virtualizados) sob a forma de instâncias com variadas capacidades em termos de processamento, memória e armazenamento. Por exemplo, o inventário do Grid5000 relata ao menos 25 modelos diferentes de processadores e nove tipos de interconexão[1]. A nuvem tornou-se uma plataforma interessante para a execução de aplicações de HPC com custo total de propriedade reduzido, permitindo alocar rapidamente grandes quantidades de recursos e pagando pelo uso. Na computação em nuvem, a capacidade de computação é empacotada na forma de instâncias (máquinas virtuais, contêineres, ou máquinas dedicadas) e entregue na forma de um serviço de utilidade. É bastante comum provedores de recursos de nuvem oferecerem dezenas de instâncias com as mais variadas configurações e preços. A título de exemplo, um provedor disponibiliza mais de uma centena de configurações com diferentes capacidades[2].

Este trabalho apresenta um estudo sobre a escalabilidade de aplicações BoT, quando estas são executadas em plataformas de computação distribuída que apresentam recursos computa-

[1] https://www.grid5000.fr/mediawiki/index.php/Hardware
[2] https://aws.amazon.com/ec2/instance-types/?nc1=h_ls

cionais heterogêneos. Mais precisamente, o presente estudo tem por objetivo investigar qual é o impacto do uso de nós de processamento com capacidades heterogêneas na execução de aplicações BoT, com respeito à escalabilidade. Como principais contribuições, este trabalho demonstra que plataformas mestre-escravo heterogêneas podem alcançar limites de escalabilidade melhores que sua contraparte homogênea para a execução de aplicações BoT, quando o poder computacional de nós individuais em plataformas homogêneas é fixo. Tais efeitos são estudados para aplicações com diferentes perfis.

1.3 Metodologia

As análises realizadas neste trabalho são feitas através de simulações. Experimentos podem ser realizados usando simuladores, *testbeds* ou provedores de nuvens reais. A realização de experimentos em nuvens públicas refletem a realidade, mas possuem fatores externos que não podem ser controlados. Além disso, a utilização de nuvens reais acarreta em custos financeiros para a pesquisa. As *testbeds* personalizadas oferecem mais controle na plataforma, mas exigem grandes esforços para a configuração do sistema. Já o uso de simuladores permite que os pesquisadores testem o desempenho de seus sistemas em um ambiente repetível, controlável e livre de custos (AL-DHURAIBI et al., 2018). O simulador utilizado neste trabalho é o Simgrid[3] (CASANOVA et al., 2014), que permite a simulação de sistemas distribuídos de larga escala, como a nuvem.

1.4 Organização do trabalho

O restante desta dissertação é organizado da seguinte forma. O capítulo 2 apresenta o referencial teórico. O capítulo 3 aborda os trabalhos relacionados. O capítulo 4 apresenta os experimentos e a análise da escalabilidade de aplicações *Bag-of-Tasks* executando em plataformas mestre-escravo heterogêneas. O capítulo 5 propõe um algoritmo de provisionamento de recursos que aplica as observações feitas no capítulo 4. E por fim, o capítulo 6 apresenta as conclusões.

[3]SimGrid: http://simgrid.gforge.inria.fr/

Capítulo 2

REFERENCIAL TEÓRICO

2.1 Computação em Nuvem

Como afirma Hashem et al. (2015), a computação em nuvem é uma das mudanças mais significativas no âmbito das modernas tecnologias de informação e comunicação, e nos serviços para aplicações corporativas, tornando-se uma arquitetura poderosa para realizar computações complexas e em larga escala.

2.1.1 Definição

Ao longo do seu processo de evolução, a computação foi se transformando em modelo composto de serviços que são fornecidos de forma semelhante aos serviços tradicionais, como água, eletricidade, gás e telefonia (BUYYA et al., 2009). Neste contexto, a computação em nuvem representa uma mudança de paradigma na forma como as empresas e os profissionais pensam em recursos de tecnologia da informação (TI), referindo-se ao fornecimento de aplicações e recursos como serviços através da *Internet* (ARMBRUST et al., 2010).

A computação em nuvem é definida pelo NIST[1], como um modelo para permitir o acesso ubíquo, conveniente e sob demanda a um conjunto compartilhado de recursos computacionais configuráveis (por exemplo, redes, servidores, armazenamento, aplicações e serviços). Esses recursos podem ser rapidamente provisionados e lançados com o mínimo esforço de gerenciamento ou interação com o provedor de serviços (MELL; GRANCE, 2011).

O termo nuvem se refere aos recursos de *hardware* e *software* que compõem os *data centers* que proveem os serviços de computação (ARMBRUST et al., 2010). Portanto, a nuvem é

[1]NIST - Instituto Nacional de Padrões e Tecnologia dos Estados Unidos.

um tipo de sistema distribuído composto por uma coleção de computadores interconectados e virtualizados, que são apresentados como um ou mais recursos computacionais unificados (HASHEM et al., 2015).

O provedor de serviços de nuvem é uma empresa ou organização que dispõe de uma infraestrutura de nuvem e oferece os serviços de computação conforme as necessidades de cada consumidor. Essas empresas costumam cobrar por esses serviços com base no modelo de pagamento por uso. Desta forma, é possível pagar pelo uso de recursos de computação por um determinado período, conforme necessário (por exemplo, processadores por hora e armazenamento por dia), bem como liberar os recursos quando não houver mais demanda (ARMBRUST et al., 2010).

Uma característica importante no provisionamento e liberação de recursos na nuvem é a elasticidade. Sem a necessidade de intervenção humana por parte dos provedores de serviços, o consumidor pode alocar recursos a qualquer tempo e em qualquer quantidade, de forma que seja possível aumentar ou diminuir de acordo com a demanda. Nesse aspecto, os recursos parecem ser ilimitados (MELL; GRANCE, 2011).

2.1.2 Modelos de Serviço

Como visto anteriormente, a computação em nuvem emprega um modelo de negócios orientado a serviços, ou seja, os recursos de hardware e software são fornecidos como serviços sob demanda (ZHANG; CHENG; BOUTABA, 2010). Esses serviços de computação em nuvem podem ser divididos em três grandes categorias:

- **Sofware como Serviço (SaaS – *Software as a Service*):** se refere ao fornecimento de aplicações de *software* pela Internet. As aplicações providas aos consumidores, executam em uma infraestrutura de nuvem, podendo ser acessadas pelos dispositivos clientes através de diferentes interfaces, tais como nevegadores web ou interfaces de programação (MELL; GRANCE, 2011). Exemplos de provedores de SaaS incluem Google Drive[2], Dropbox[3] e Adobe Creative Cloud[4].

- **Plataforma como Serviço (PaaS – *Platform as a Service*):** esse nível de abstração dos sistemas em nuvem fornece uma plataforma para desenvolvimento, testes, execução e manutenção de aplicações de *software* (VAQUERO et al., 2008). Isso facilita a criação

[2]Google Drive: https://www.google.com/drive/
[3]Dropbox: https://www.dropbox.com
[4]Adobe Creative Cloud: http://www.adobe.com/br/creativecloud

de aplicativos, pois elimina a preocupação com a configuração e gerenciamento da infraestrutura subjacente (rede, servidores, armazenamento, etc), uma vez que o consumidor não gerencia essa infraestrutura. Por sua vez, os desenvolvedores podem controlar as aplicações instaladas e gerenciar as configurações do ambiente de hospedagem e execução das aplicações (MELL; GRANCE, 2011). Exemplos de provedores de PaaS incluem Google App Engine[5] e Microsoft Azure[6].

- **Infraestrutura como Serviço (IaaS – *Infrastructure as a Service*):** consite no fornecimento de recursos de *hardware* como serviços (HASHEM et al., 2015). É a categoria mais básica de serviços de computação em nuvem, permitindo o provisionamento, sob demanda, de processamento, armazenamento, redes e outros recursos computacionais necessários para a instalação e execução de *softwares*. Como afirma Vaquero et al. (2008), através da virtualização, é possível dividir, atribuir e redimensionar dinamicamente os recursos. Exemplos de provedores de IaaS incluem Amazon EC2[7] e IBM Cloud[8].

2.1.3 Arquitetura em Camadas

Zhang, Cheng e Boutaba (2010) apresenta uma arquitetura baseada em camadas (Figura 2.1) para o ambiente de computação em nuvem. Cada camada da arquitetura pode ser implementada como um serviço para a camada superior, e esta por sua vez, é percebida como um consumidor dos recursos da camada inferior.

A primeira camada é a camada de *hardware*, que consiste nos recursos físicos da nuvem, incluindo servidores, discos de armazenamento e equipamentos de rede. A camada seguinte é a camada de infraestrutura, que faz uso das tecnologias de virtualização para particionar os recursos físicos e criar um conjunto de recursos de computação e armazenamento. O modelo de serviço IaaS atua no nível dessas duas camadas. A terceira camada, chamada de camada de plataforma, permite a implantação do modelo de serviço PaaS e consiste nos sistemas operacionais e *frameworks* para o desenvolvimento de aplicações, provendo uma API[9] para a implementação de regras de negócios e armazenamento de dados. A última camada na hierarquia é a camada de aplicação, que consiste nas aplicações da nuvem, sendo o ambiente de implantação do modelo de serviço SaaS.

[5]Google App Engine: https://cloud.google.com/appengine/
[6]Microsoft Azure: https://azure.microsoft.com
[7]Amazon EC2: https://aws.amazon.com/pt/ec2/
[8]IBM Cloud: https://www.ibm.com/cloud/
[9]API - Interface de Programação de Aplicações.

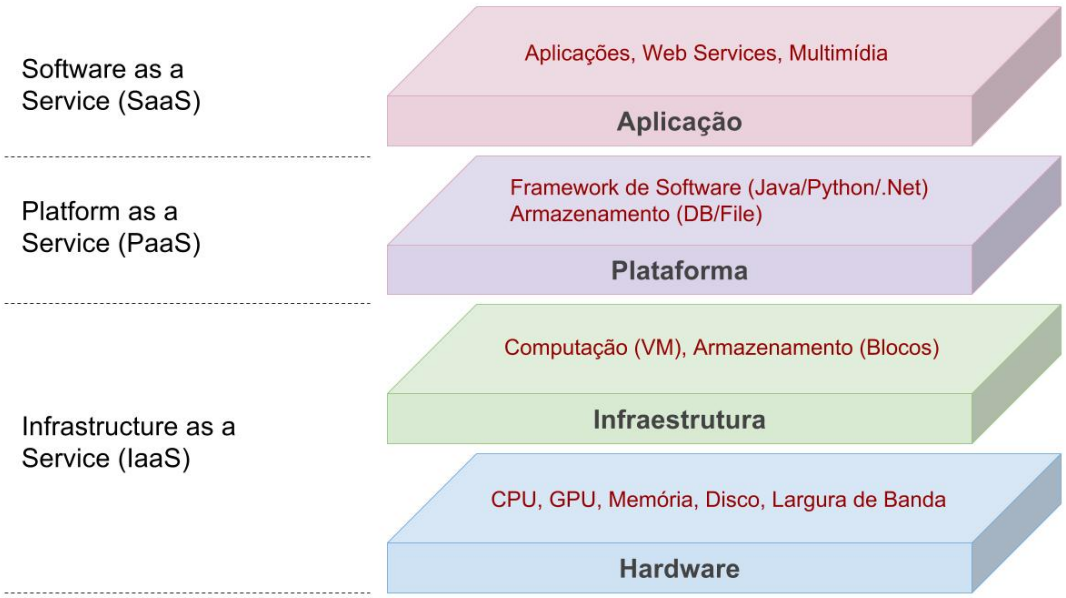


Figura 2.1: Arquitetura da computação em nuvem (adaptado de Zhang, Cheng e Boutaba (2010))

2.1.4 Modelos de Implantação

De acordo com o modelo de implantação, a nuvem pode ser classificada nas seguintes categorias:

- **Nuvem privada:** neste modelo, a infraestrutura de nuvem é de uso exclusivo de uma única organização, portanto, os recursos computacionais não são compartilhados com terceiros (CHOU, 2015). A estrutura de *data center* pode ser gerenciada pela própria organização ou por provedores externos, através de uma rede privada.

- **Nuvem pública:** é o tipo mais comum de computação em nuvem. Neste modelo, a empresa proprietária da nuvem provê os serviços para o público em geral, baseados em regras predefinidas, políticas e modelos de pagamento por uso (JULA; SUNDARARAJAN; OTHMAN, 2014).

- **Nuvem comunitária:** é utilizada e gerenciada por um grupo de organizações que possuem interesses comuns (GUPTA; SEETHARAMAN; RAJ, 2013). Essas organizações estabelecem uma comunidade e compartilham recursos de computação em nuvem para uso por parte dos seus membros.

- **Nuvem híbrida:** é um modelo de implantação de nuvem que combina infraestruturas de duas ou mais nuvens distintas (privada, pública ou comunitária), que se mantêm como entidades únicas, porém ligadas por tecnologias que permitem o compartilhamento de dados e aplicações entre elas (MELL; GRANCE, 2011).

2.2 Softwares para Provisionamento de IaaS

A plataforma da nuvem é composta por um conjunto de recursos computacionais tais como servidores, sistemas de armazenamento, dispositivos de rede, entre outros recursos, que são alocados em infraestruturas de *data centers*. O gerenciamento desses recursos é feito por uma camada de software, um *middleware* ou plataforma de gerenciamento de nuvem, que possui a finalidade de esconder a complexidade do sistema, permitir um mecanismo de comunicação transparente entre os componentes da nuvem e provisionar os recursos de IaaS (AMRANI et al., 2012).

Entre as funcionalidades e características dos *middlewares* de nuvem, pode-se destacar: a replicação de dados, escalonamento de tarefas, tolerância a falhas, balanceamento de cargas, suporte a interoperabilidade, segurança, gerenciamento de imagens de disco, monitoramento de recursos, gerenciamento de virtualização, gerenciamento de usuários e painel de controle (AMRANI et al., 2012). Exemplos de plataformas que gerenciam a infraestrutura da nuvem incluem *OpenStack*[10], *Apache CloudStack*[11], *OpenNebula*[12], *Nimbus*[13] e *Eucalyptus*[14].

2.2.1 OpenStack

OpenStack é um *software* de código aberto que permite a criação de serviços de IaaS. O projeto OpenStack teve início em 2010 a partir de uma parceria entre a Rackspace e a NASA. Em 2016, o projeto passou a ser administrado pela *OpenStack Foundation*[15]. O OpenStack é conhecido como o sistema operacional da nuvem, por conta da sua capacidade de gerenciar um grande conjunto de recursos computacionais, de armazenamento e de rede em um *data center*.

O OpenStack é uma solução recente, em comparação com outros gerenciadores de IaaS disponíveis, e está em constante desenvolvimento, contando com a participação ativa de empresas e pessoas de todo o mundo. É escrito em Python e suporta a maioria das soluções de virtualização do mercado, tais como: ESX, Hyper-V, KVM, LXC, QEMU, UML, Xen e XenServer (MISHRA, 2017).

O projeto OpenStack é uma plataforma de computação em nuvem para todos os tipos de nuvens, e tem como objetivo ser uma solução escalável, simples de implementar e rica em re-

[10] OpenStack: https://www.openstack.org/
[11] Apache CloudStack: https://cloudstack.apache.org/
[12] OpenNebula: https://opennebula.org/
[13] Nimbus: http://www.nimbusproject.org/
[14] Eucalyptus: https://docs.eucalyptus.com/
[15] OpenStack Foundation: https://www.openstack.org/foundation/

cursos. Isso é feito por meio de um conjunto de serviços inter-relacionados. Esses serviços interagem entre si por meio de APIs públicas, sendo possível instalar alguns ou todos os serviços, de acordo à necessidade. Os serviços que compõem a arquitetura do OpenStack (Figura 2.2) são descritos a seguir (OPENSTACK, 2018):

- ***Horizon (Dashboard):*** fornece uma interface web para interação com os serviços subjacentes do OpenStack, possibilitando que o cliente da nuvem gerencie seus recursos.
- ***Nova (Compute service):*** gerencia o ciclo de vida das instâncias de computação. É responsável pelo provisionamento de máquinas virtuais sob demanda.
- ***Neutron (Networking service):*** provê conectividade de rede como serviço para outros componentes do OpenStack. Fornece uma API para configuração de redes virtuais.
- ***Swift (Object Storage service):*** possibilita o armazenamento e recuperação de objetos de dados. Possui tolerância a falhas, pois grava objetos e arquivos em várias unidades, garantindo replicação dos dados.
- ***Cinder (Block Storage service):*** fornece armazenamento de bloco persistente para instâncias em execução.
- ***Keystone (Identity service):*** provê autenticação e autorização para os serviços do OpenStack.
- ***Glance (Image service):*** armazena e recupera imagens de disco que são utilizadas no provisionamento de máquinas virtuais.
- ***Ceilometer (Telemetry service):*** realiza o monitoramento da nuvem para fins de faturamento, *benchmarking*, escalabilidade e avaliações estatísticas.
- ***Heat (Orchestration service):*** realiza a orquestração de aplicações compostas em nuvem usando o formato nativo HOT ou o formato *Cloud Formation* da AWS.
- ***Trove (Database service):*** provê a funcionalidade de banco de dados como serviço na nuvem.
- ***Sahara (Data Processing service):*** fornece recursos para provisionar e dimensionar *clusters* do Hadoop no OpenStack.

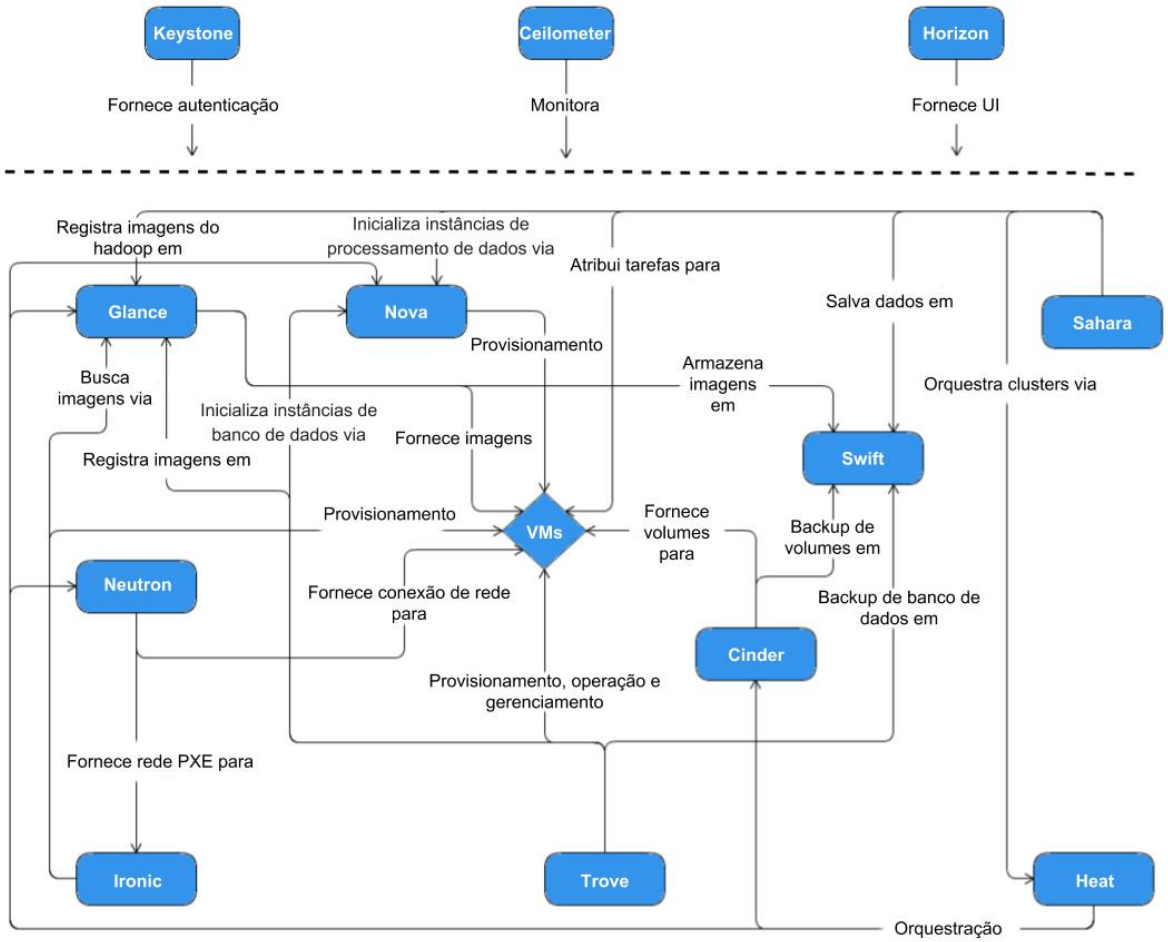


Figura 2.2: Arquitetura do OpenStack (adaptado de OpenStack (2018))

2.2.2 Apache CloudStack

Apache CloudStack é um *software* de código aberto usado na implementação e gerenciamento de plataformas de IaaS, permitindo a criação de nuvens públicas, privadas e híbridas. O CloudStack gerencia recursos computacionais, de armazenamento e de rede, e suporta a maioria dos *hypervisors* populares, tais como: VMware, KVM, Citrix XenServer, Xen Cloud Platform (XCP), Oracle VM server e Microsoft Hyper-V (KUMAR et al., 2014).

O projeto CloudStack teve início em 2008, sendo originalmente desenvolvido pela empresa Cloud.com, que foi adquirida pela Citrix em julho de 2011. Em abril de 2012, a Citrix liberou o código fonte do CloudStack sob a licença Apache 2.0. Atualmente, o Apache CloudStack é um projeto de alto nível da *Apache Software Foundation* (ASF) e fornece uma plataforma de orquestração de nuvem aberta e flexível, provendo nuvens confiáveis e escalonáveis (CLOUDSTACK, 2018).

De maneira geral, a arquitetura de implementação do CloudStack é composta pelo servidor

de gerenciamento e pelos recursos a serem gerenciados. O servidor de gerenciamento orquestra e aloca os recursos no ambiente da nuvem. Ele é executado em um *container* do Apache Tomcat, que pode ser alocado em uma máquina dedicada ou virtual. Além disso, sua execução requer um banco de dados MySQL para persistência (CLOUDSTACK, 2018). O servidor de gerenciamento é responsável por:

- prover uma interface web para interação com o administrador e com o cliente.
- prover APIs para a operação e uso da nuvem.
- gerenciar a alocação de VMs para um *host* específico.
- gerenciar a atribuição de endereços de IP públicos e privados.
- alocar armazenamento durante o processo de instanciação da VM.
- gerenciar *snapshots* e imagens de disco.
- prover um único ponto de configuração para a nuvem.

Os recursos dentro da nuvem do CloudStack (Figura 2.3) são organizados e gerenciados da seguinte forma (CLOUDSTACK, 2018):

- ***Host:*** é a menor unidade organizacional dentro do ambiente do CloudStack. Trata-se de um computador com um *hypervisor* instalado e que provê recursos para a execução de máquinas virtuais.
- **Armazenamento Primário:** está associado a um *cluster* e armazena discos virtuais para todas as VMs que estão executando nos *hosts* desse *cluster*.
- **Armazenamento Secundário:** armazena *templates* de disco, imagens ISO e *snapshots* disponíveis em uma *zone*.
- ***Cluster:*** consiste no agrupamento de *hosts* e servidores de armazenamento primário.
- ***Pod:*** representa um *rack* ou fileira de *racks*, consistindo de um ou mais *clusters*.
- ***Zone:*** é equivalente a um único *data center*, consistindo de um ou mais *pods* e armazenamento secundário.
- ***Region:*** é um conjunto de *zones* geograficamente próximas e que são gerenciadas por um ou mais servidores de gerenciamento.

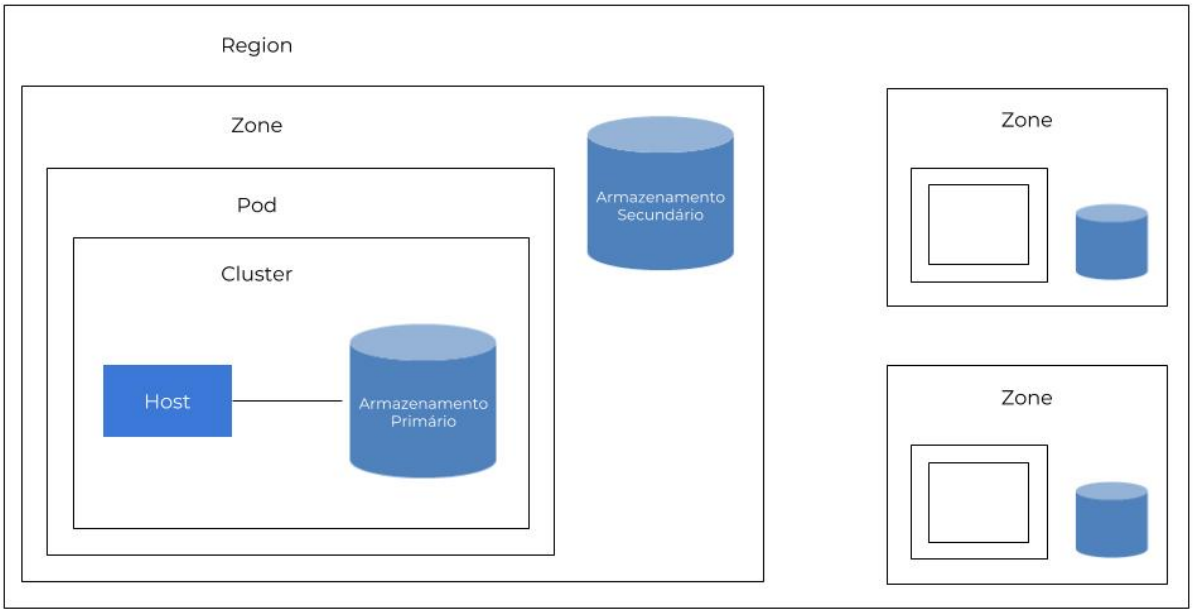


Figura 2.3: Organização dos recursos no Apache CloudStack (adaptado de CloudStack (2018))

2.2.3 OpenNebula

O OpenNebula é uma plataforma de gerenciamento que possibilita a criação de serviços de IaaS em nuvens privadas, públicas e híbridas. O OpenNebula teve início como um projeto de pesquisa em 2005, tendo sua primeira versão lançada ao público em 2008. Desde então, o *software* evoluiu com o lançamento de novas versões e opera como um projeto de código aberto. O OpenNebula foi projetado para ser modular, de forma a permitir a integração com diferentes ambientes e soluções de virtualização, oferecendo suporte completo para os *hypervisors* KVM e VMware, além de suportar também outras tecnologias de virtualização como Xen e LXC (OPENNEBULA, 2018).

No OpenNebula, a infraestrutura física da nuvem adota uma arquitetura tradicional de *cluster*. Nesse modelo, um nó do agrupamento de máquinas, denominado *front-end*, executa os processos principais do OpenNebula e controla os demais nós, que são responsáveis pela execução das máquinas virtuais (CORDEIRO et al., 2010).

A arquitetura do OpenNebula é dividida em quatro camadas: ferramentas, interfaces, núcleo e *drivers*, como mostra a Figura 2.4. A camada de ferramentas contém os módulos que proveem funcionalidades para interação com administradores e clientes da nuvem, tais como interface de linha de comando e interface gráfica. Essa camada também implementa um módulo de escalonamento, que é responsável pela alocação das máquinas virtuais. A camada de interface fornece uma API para integração com outras ferramentas que podem ser criadas, e oferece as funcionalidades do núcleo do OpenNebula por meio de ligações para Ruby, Java, XML e

Chamada de Procedimento Remoto (RPC).

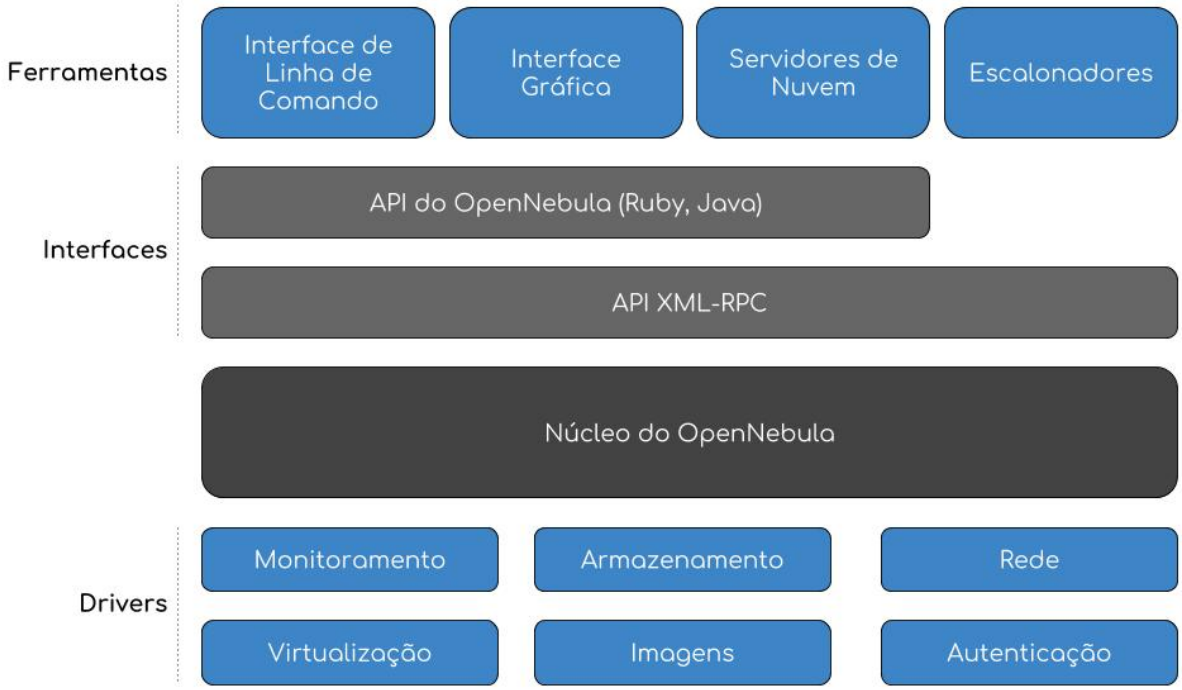


Figura 2.4: Arquitetura do OpenNebula (adaptado de OpenNebula (2018))

O núcleo do OpenNebula é responsável por processar as requisições dos clientes e gerenciar os recursos da nuvem. Essa camada cuida dos procedimentos para gerenciamento de imagens de disco, provimento de ambientes de rede e gerenciamento do ciclo de vida das máquinas virtuais. A camada de *drivers* consiste de módulos que promovem a interação com tecnologias específicas, como protocolos de transferência de arquivos, *hypervisors* de virtualização, dispositivos de rede, entre outros. Esses módulos executam em processos distintos e suportam diferentes plataformas subjacentes (CORDEIRO et al., 2010).

2.2.4 Nimbus

Nimbus é um conjunto de ferramentas de código aberto, focado no fornecimento de serviços de IaaS para a comunidade científica. Permite que provedores de recursos criem nuvens privadas, públicas ou comunitárias. Sua missão é desenvolver infraestruturas de nuvem com ênfase em necessidades e projetos científicos, mas muitos casos de uso não científicos também são suportados. O projeto Nimbus foi criado por uma colaboração internacional, envolvendo desenvolvedores e instituições. Sua plataforma permite a utilização dos *hypervisors* Xen e KVM (NIMBUS, 2018).

A Figura 2.5 apresenta os principais componentes do Nimbus. O *Workspace Service* é um

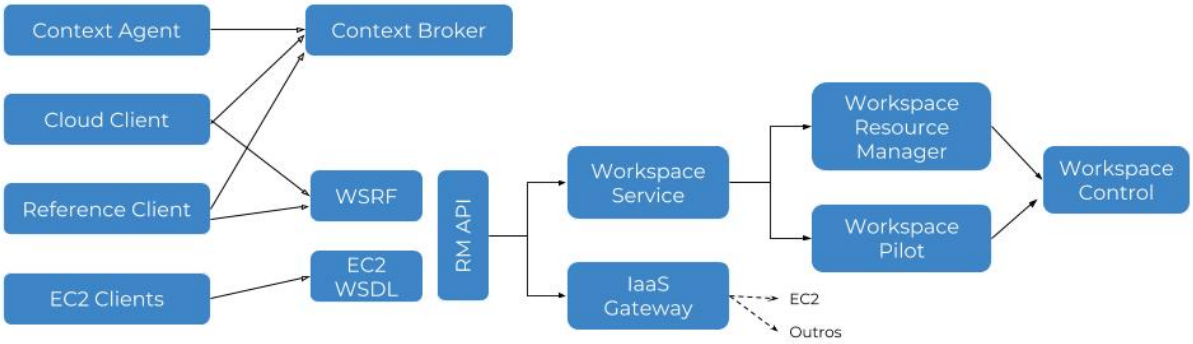


Figura 2.5: Componentes do Nimbus (adaptado de Nimbus (2018))

gerenciador de máquinas virtuais que pode ser invocado por diferentes protocolos de *front-end* remotos, como protocolos baseados em *Web Services Resource Framework (WSRF)* e *Amazon Elastic Compute Cloud (EC2)*. O *Cumulus* é uma implementação da API REST da *Amazon S3*, sendo utilizado como um repositório de imagens do Nimbus. O *Cloud Client* tem o objetivo de auxiliar os usuários da nuvem com o lançamento de instâncias. O *Reference Client* oferece o conjunto de recursos do protocolo WSRF como um cliente de linha de comandos. O *Workspace Pilot* permite a integração de VMs com recursos já configurados para gerenciar tarefas. O *Workspace Resource Manager* é usado no gerenciamento de um conjunto de recursos físicos. O *Workspace Control* implementa o gerenciamento de VMs e tarefas específicas de rede em cada *hypervisor*. O *Context Broker* permite aos clientes coordenar lançamentos de *clusters* virtuais de forma automática. O *Context Agent* é executado nas VMs e interage com o *Context Broker* na inicialização da VM (ISMAEEL et al., 2015) (NIMBUS, 2018).

2.2.5 Eucalyptus

O Eucalyptus (*Elastic Utility Computing Architecture for Linking Your Programs To Useful Systems*) é um *software* de código aberto, desenvolvido na Universidade da Califórnia, para criação e gerenciamento de nuvens privadas e públicas. O Eucalyptus tem como objetivo implementar uma nuvem compatível com a *Amazon Web Services (AWS)*, fornecendo uma plataforma de computação compatível com a *Amazon EC2* e uma plataforma de armazenamento compatível com a *Amazon S3*. O sistema possui suporte para execução de VMs dos *hypervisors* Xen, KVM e VMware (ISMAEEL et al., 2015).

O Eucalyptus é baseado em uma abordagem de projeto modular e hierárquica, onde o papel de cada componente do sistema é claramente definido (KHAN; REHMAN; ANWAR, 2011). Sua arquitetura é composta por cinco componentes principais, que estão agrupados em diferentes camadas, como mostra a Figura 2.6. Esses componentes são descritos a seguir:

- ***Cloud Controller (CLC):*** é o controlador principal, responsável por gerenciar a plataforma de nuvem como um todo. Além disso, é o *front-end* de todos os usuários e administradores na nuvem. Suas funções incluem escalonamento, alocação de recursos e contabilidade.
- ***Walrus Storage Controller (WS3):*** é um sistema de armazenamento de arquivos, similar ao *Amazon S3*. Fornece armazenamento persistente de imagens e *snapshots* para todas as instâncias.
- ***Cluster Controller (CC):*** é responsável por gerenciar toda a rede de instâncias virtuais. Além disso, age como uma ponte de comunição entre os *Node Controllers* e o *Cloud Controller*.
- ***Storage Controller (SC):*** fornece armazenamento em nível de bloco para as instâncias dentro de um *cluster*.
- ***Node Controller (NC):*** é o componente básico do Eucalyptus, responsável por controlar as instâncias das máquinas virtuais nos nós. Ele gerencia o ciclo de vida das instâncias em cada nó, através de interações com o sistema operacional, *hypervisor* e *Cluster Controller*.

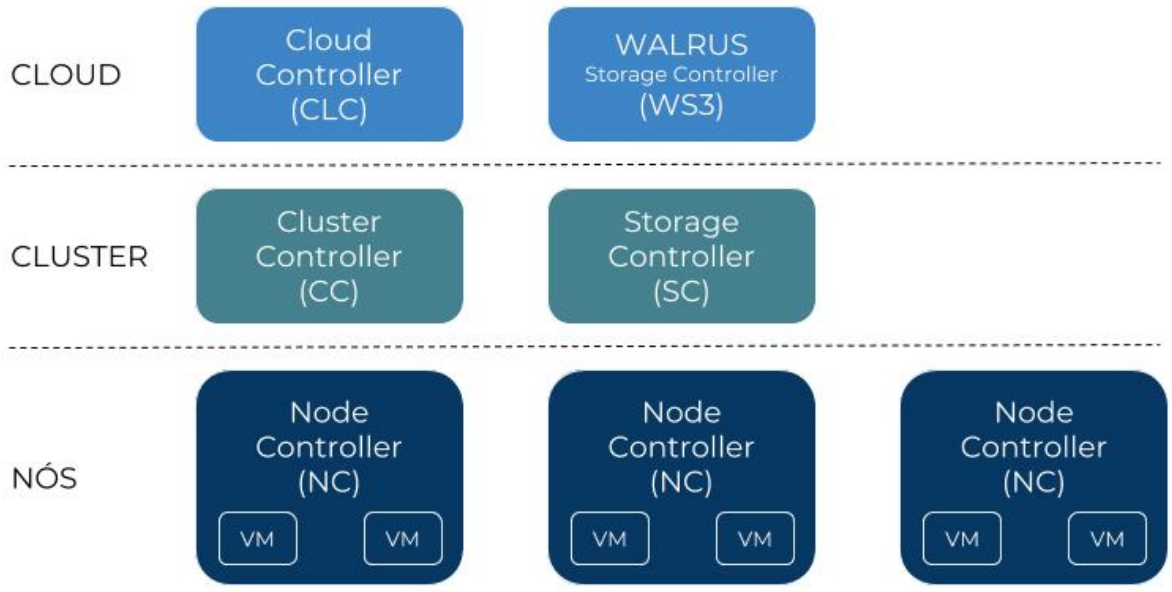


Figura 2.6: Componentes do Eucalyptus (adaptado de Ismaeel et al. (2015))

2.3 Provedores de Nuvem

Como abordado anteriormente, os provedores de nuvem são empresas que possuem infraestruturas de *data centers* e oferecem serviços baseados em nuvem. Esse provedores permitem

que os clientes utilizem os recursos virtuais (computação, memória, armazenamento, redes, etc) que estão executando em máquinas físicas. Nesse modelo, os usuários têm o benefício de usar a infraestrutura, plataforma e *softwares* disponibilizados pelo provedores de nuvem a um custo viável e sob demanda (OKI et al., 2017).

Atualmente existem diversas empresas que proveem serviços de computação em nuvem. Provedores como *Amazon Web Services*[16], *Google Cloud Platform*[17] e *Microsoft Azure*[18] são alguns exemplos de empresas consolidadas no segmento. Cada provedor oferece distintos serviços de IaaS, PaaS e SaaS, além de apresentar diferentes planos de pagamento que podem ser contratados de acordo às necessidades das aplicações dos clientes.

2.3.1 Amazon Web Services

A Amazon Web Services (AWS) começou a oferecer serviços de infraestrutura de TI para empresas em 2006. Atualmente a AWS oferece um amplo conjunto de produtos e serviços em nuvem, como computação, armazenamento, bancos de dados, análise, redes, dispositivos móveis, ferramentas para desenvolvedores, ferramentas de gerenciamento, IoT, segurança e aplicações empresariais. Esses serviços são entregues sob demanda e com definição de preço conforme o uso.

Os serviços de computação da AWS são disponibilizados por meio do *Amazon Elastic Compute Cloud (Amazon EC2)*. Através de uma interface web, o Amazon EC2 permite que o usuário da nuvem obtenha e inicialize instâncias virtuais em minutos, possibilitando também escalar os recursos para aumentar ou diminuir a capacidade à medida que os requisitos de computação são alterados. Além disso, é possível utilizar o *Amazon EC2 Auto Scaling* para manter a disponibilidade da aplicação, pois esse recurso assegura que a aplicação tenha a quantidade necessária de capacidade computacional. O *Amazon EC2 Auto Scaling* aumenta a tolerância a falhas ao realizar a detecção e substituição de instâncias com problemas, e também realiza a adição e remoção de instâncias para respectivamente, manter a performance quando há um aumento da demanda e reduzir custos quando a demanda for menor. Ainda em termos de serviços de computação, a AWS oferece o *Amazon Elastic Container Service (Amazon ECS)*. Trata-se de um serviço para orquestração de *containers* Docker, que permite a criação e inicialização de *containers* de forma altamente escalável e em questão de segundos.

O Amazon EC2 disponibiliza um amplo conjunto de tipos de instâncias, com diferentes

[16] Amazon Web Services: https://aws.amazon.com/
[17] Google Cloud Platform: https://cloud.google.com/
[18] Microsoft Azure: https://azure.microsoft.com

combinações de CPU, memória e armazenamento. Isso permite a adequação das instâncias para casos de uso específicos. De acordo com os requisitos das aplicações a serem executadas, as instâncias do Amazon EC2 podem ser agrupadas em:

- **Uso geral:** é composto pelas instâncias T2 que oferecem um nível básico de performance de CPU e pelas instâncias M5 e M4, que apresentam um equilíbrio de recursos de computação, memória e rede.
- **Otimizadas para computação:** nesse agrupamento as instâncias dos tipos C5 e C4 são otimizadas para aplicações com uso intensivo de computação.
- **Otimizadas para memória:** compõem esse agrupamento as instâncias dos tipos X1e, X1 e R4. Essas instâncias são otimizadas para aplicações que fazem uso intensivo de memória.
- **Computação acelerada:** é composto por instâncias dos tipos P3 e P2, que são destinadas às aplicações que realizam processamento em GPU. Além disso são disponibilizadas as instâncias G3 que são otimizadas para aplicações com alto consumo de gráficos e as instâncias F1 que oferecem aceleração de *hardware* personalizável com FPGAs.
- **Otimizadas para armazenamento:** são instâncias que oferecem maiores espaços de armazenamento em disco e com alta taxa de transferência. As instâncias H1 oferecem até 16 TB de armazenamento local baseado em HDD, as instâncias I3 oferecem armazenamento baseado em SSD e otimizado para baixa latência, e por fim, as instâncias D2 oferecem até 48 TB de armazenamento local baseado em HDD.

A AWS oferece três formas de pagamento por instâncias do Amazon EC2: instâncias sob demanda, instâncias reservadas e instâncias *spot*. Além disso, disponibiliza também o pagamento por *hosts* dedicados. No plano sob demanda, o cliente paga apenas pelas instâncias que utilizar, permitindo o pagamento da capacidade computacional por hora ou por segundo, e não requer compromissos de longo prazo. No plano por reserva, as instâncias podem ser compradas por um período de tempo de um ou três anos. As instâncias reservadas oferecem descontos de até 75% em relação aos preços das instâncias sob demanda e são recomendadas para aplicações com estado constante. Já as instâncias *spot* são recomendadas para aplicações que têm períodos de início e de término flexíveis. Neste modelo, os preços são ajustados de acordo com as tendências de oferta e demanda da capacidade computacional das instâncias *spot* e o cliente paga o preço em vigor no período de execução das instâncias. E os *hosts* dedicados são servidores físicos que podem ser comprados sob demanda ou por reserva.

2.3.2 Google Cloud Platform

O Google Cloud Platform é um conjunto de serviços de computação em nuvem oferecido pelo Google. A plataforma teve início em 2008 com o lançamento do *Google App Engine*, oferecendo PaaS para a hospedagem de aplicações. Desde então, novos serviços foram adicionados à plataforma. Atualmente são oferecidos serviços de computação, armazenamento e banco de dados, rede, IoT, aprendizado de máquina, ferramentas de gerenciamento, ferramentas para desenvolvedores, entre outros.

O serviço de computação do Google Cloud Platform é conhecido como *Google Compute Engine*. Esse serviço oferece duas categorias de tipos de máquinas virtuais: tipos predefinidos e personalizados. As VMs predefinidas possuem configurações fixas para cada necessidade. Existem instâncias com recursos otimizados para uso padrão, alta utilização de CPU e uso intensivo de memória. Já as VMs personalizadas permitem a customização das quantidades de memória e vCPUs das instâncias, de modo a definir quantidades ideais de recursos para as cargas de trabalho. Todos os tipos de máquinas virtuais são cobrados de acordo ao uso por segundo. A plataforma também disponibiliza o *Google Kubernetes Engine*, que é um ambiente para implantação e gerenciamento de aplicativos em *containers*.

O Google Compute Engine oferece descontos por uso prolongado e uso contínuo. O uso prolongado é caracterizado pela utilização de uma instância por mais de 25% de um mês. Neste caso, ocorre um desconto automático para cada segundo a mais usado nessa instância, podendo atingir até 30% de desconto para instâncias executadas por um mês completo. Já o uso contínuo exige um contrato de compromisso. Nesse modelo, é possível comprar um número específico de recursos com até 57% de desconto sobre os preços totais. O cliente, portanto, assume um compromisso de uso por um prazo determinado e é cobrado por mês, independente da ocorrência de uso.

2.3.3 Microsoft Azure

O Microsoft Azure é um conjunto de serviços em nuvem que começou a operar em 2010. A plataforma conta com diversos produtos como computação, armazenamento, bancos de dados, redes, recursos de inteligência artificial e aprendizado de máquina, ferramentas de gerenciamento, ferramentas para desenvolvedores, análises, entre outros serviços.

Da mesma forma que outros provedores de nuvem, o Microsoft Azure disponibiliza capacidade de computação com escala sob demanda, no qual o cliente paga apenas pelos recursos que utilizar. O Azure permite o provisionamento rápido de infraestruturas para todas as cargas

de trabalho e em máquinas virtuais Windows e Linux. Além disso, o usuário tem a opção de utilizar instâncias de *containers* para a execução das aplicações, através do *Azure Container Instances (ACI)*. É possível também, usar o *Azure Kubernetes Service (AKS)*, um serviço de orquestração de *containers* do Kubernetes, que simplifica a implantação e o gerenciamento da infraestrutura de *containers*.

O Microsoft Azure também dispõe de serviços para a aplicação de dimensionamento automático no conjunto de máquinas virtuais. Dessa forma, a plataforma possibilita a quantidade certa de recursos em execução para suprir a demanda da carga de trabalho da aplicação. O dimensionamento automático permite adicionar recursos para lidar com aumentos da carga e também reduzir custos removendo os recursos ociosos. Neste modelo, o usuário define um número mínimo e máximo de instâncias a serem executadas e o sistema realiza o dimensionamento com base em um conjunto de regras. O dimensionamento automático ocorre apenas horizontalmente, ou seja, realiza o aumento ou diminuição do número de instâncias. Mas o Azure permite também o redimensionamento vertical, no qual o número de VMs é mantido, e ocorre o aumento ou diminuição do poder computacional de cada VM. É um método mais limitado, pois depende da disponibilidade de um *hardware* mais potente e também exige a reinicialização da máquina virtual.

As instâncias do Microsoft Azure são otimizadas e classificadas de acordo com os requisitos das aplicações. O provedor oferece os seguintes tipos de instâncias: propósito geral, otimizadas para computação, otimizadas para memória, otimizadas para armazenamento, otimizadas para GPU e otimizadas para computação de alto desempenho. O Microsoft Azure permite as opções de pagamento conforme o uso das máquinas virtuais por segundo, ou por reserva de instâncias. Na primeira opção não há compromisso de longo prazo ou pagamentos adiantados. Na opção por reserva ocorre uma compra adiantada de uma ou mais VMs por um período de um ou três anos. Nesse plano há, portanto, um compromisso feito antecipadamente, porém, ele concede um desconto de até 72% em comparação com o plano de pagamento conforme o uso.

2.4 Nuvem para Computação Científica

A computação é uma ferramenta essencial para uma gama de estudos científicos. Diversas áreas como física, química, genética, economia, engenharia, entre tantas outras, fazem uso da computação para simulação de eventos e processamento de grandes massas de dados. Nesse contexto a computação de alto desempenho (HPC) se tornou uma parte integral de muitas áreas de pesquisa. Entretanto, infraestruturas tradicionais de HPC, tais como *clusters* e supercom-

putadores demandam alto investimento financeiro, sendo inviável para muitos pesquisadores. O advento da computação em nuvem, porém, tornou a computação científica mais acessível, possibilitando aos pesquisadores alugar os recursos computacionais sob demanda pelo tempo que necessitar (JORISSEN et al., 2012).

Nos últimos anos, a computação em nuvem amadureceu e se consolidou como uma tecnologia que permite o provisionamento de diversos tipos de recursos computacionais tais como processamento, armazenamento e transmissão de dados. Aliado a isso, apresenta um conjunto de aplicações e ferramentas que são oferecidas sob a forma de serviço para organizações e usuários em geral. E quando comparada com as demais plataformas de alto desempenho, a computação em nuvem se mostra mais barata e escalável. Desta forma, se tornou uma alternativa interessante para a execução de aplicações científicas.

2.5 Simuladores

Simulações são amplamente utilizadas para a modelagem de processos do mundo real em diversas áreas como física, indústria, construção e ciência da computação. A simulação possibilita o estudo de diferentes características dos sistemas como viabilidade, comportamento e desempenho. Além disso, o uso de simuladores elimina a necessidade de criação de sistemas reais. Desta forma, com o mínimo de esforço é possível reduzir os custos e o tempo de geração dos resultados (SULISTIO; YEO; BUYYA, 2018).

A simulação oferece um ambiente controlável e repetível para a execução dos experimentos, e com cenários quase ilimitados. Isso permite que qualquer pesquisador possa reproduzir os resultados de experimentos realizados (GRACE; PRIYA; SURYA, 2012). Diante disso, muitas ferramentas foram desenvolvidas para a simulação de sistemas paralelos e distribuídos. Essas ferramentas realizam a modelagem de operações reais e comportamentos de diversos tipos de sistemas computacionais, bem como, simulam a comunicação e os eventos que ocorrem entre eles (SULISTIO; YEO; BUYYA, 2018).

Os simuladores são amplamente usados na simulação de plataformas de nuvem. A execução de experimentos nesse ambiente escalável e livre de custos possibilita a experimentação de novas soluções e o ajuste de gargalos de desempenho antes de implementações em nuvens reais (AL-DHURAIBI et al., 2018). Alguns simuladores de nuvem são:

- **CloudSim:** é um *framework* para a modelagem e simulação de infraestruturas e serviços de computação em nuvem. Suporta a simulação do comportamento e do sistema dos

componentes da nuvem como *data centers*, máquinas virtuais, *containers* e políticas de provisionamento de recursos, além de nuvens federadas (CALHEIROS et al., 2011).

- **GreenCloud:** é um ambiente de simulação para *data centers* de computação em nuvem. É projetado para capturar detalhes da energia consumida pelos componentes como servidores, *switches* e *links* de rede. Além disso, é usado no desenvolvimento de soluções de monitoramento, alocação de recursos, escalonamento de cargas de trabalho e otimização dos protocolos de comunicação e infraestruturas de rede (KLIAZOVICH et al., 2010).

- **iCanCloud:** é uma plataforma para a modelagem e simulação de sistemas de computação em nuvem. Esse simulador tem como objetivos conduzir grandes experimentos, reproduzir os tipos de instâncias providos por uma dada infraestrutura de nuvem e disponibilizar uma interface gráfica para a configuração e realização de simulações, que podem consistir de uma única VM até grandes sistemas com milhares de máquinas (NúñEZ et al., 2012).

- **SimGrid:** é um *framework* para a simulação de sistemas distribuídos de larga escala como *grids*, HPC, P2P e nuvem. Esse simulador é versátil e flexível, e pode ser usado para avaliar heurísticas e protótipos de aplicações (CASANOVA et al., 2014).

2.6 SimGrid

Esta seção apresenta uma descrição mais detalhada do Simgrid, que é o simulador utilizado neste trabalho. Como já mencionado, o Simgrid é um *framework* para a simulação de sistemas distribuídos de larga escala, que pode ser usado na avaliação de performance de protótipos de aplicações e pesquisas científicas. O Simgrid é um *software* livre e teve seu desenvolvimento iniciado de 1999. Atualmente, o Simgrid é amplamente utilizado no mundo todo e conta com diversos usuários de diferentes comunidades de computação distribuída, fundamentando os experimentos de numerosos trabalhos científicos[19]. Uma das principais razões para o sucesso do Simgrid é que ele reúne vários modelos adaptados para projetar as características de sistemas distribuídos. Por isso, é versátil e flexível, permitindo que os usuários personalizem o nível desejado de heterogeneidade, interferência e volatilidade. Além disso, tem boa escalabilidade e acurácia (CASANOVA et al., 2014). Por conta das características mencionadas acima, aliado ao fato de satisfazer as especificidades dos experimentos propostos neste trabalho, o Simgrid foi escolhido com a ferramenta de simulação utilizada.

Apesar do nome, o Simgrid não é uma ferramenta dedicada para a computação em grade.

[19] https://simgrid.org/Usages.html

O simulador evoluiu bastante ao longo do tempo, se tornando um *framework* genérico para a simulação de diferentes tipos de plataformas distribuídas, tais como *clusters*, nuvem, HPC, computação voluntária, grades computacionais e sistemas P2P (CASANOVA et al., 2014). Essas plataformas apresentam ideias e aspectos comuns que precisam ser modelados. Desta forma, o Simgrid visa propor um *framework* geral para a avaliação de sistemas de larga escala. A simulação no Simgrid é executada definindo as configurações da plataforma e as aplicações. Para isso, são utilizadas duas abstrações: recursos e ações. Os recursos são os parâmetros de *hardware* que descrevem a plataforma, como *links* de rede, roteadores e unidades de processamento. As ações são partes de *softwares* que consomem os recursos e podem ser de computação (representa uma carga de trabalho a ser executada em um nó) e comunicação (representa um fluxo de dados de uma origem para um destino) (VELHO, 2011). A seguir são abordados alguns parâmetros necessários para execução de simulações no Simgrid.

2.6.1 Descrição da Plataforma

A plataforma no Simgrid consiste na definição dos recursos de rede e de computação. Esses recursos descrevem a infraestrutura de comunicação e o poder computacional providos pelo sistema distribuído. Para representar esse conjunto de recursos, o Simgrid utiliza um arquivo XML. Tal arquivo, chamado de arquivo de plataforma, é carregado no início da simulação para informar a configuração dos recursos ao *kernel* do simulador.

```
<?xml version='1.0'?>
<!DOCTYPE platform SYSTEM "simgrid.dtd">
<platform version="4.1">
  <zone id="AS0" routing="Full">

    <host id="H1" speed="10Gf"/>
    <host id="H2" speed="20Gf"/>

    <router id="router1"/>
    <router id="router2"/>

    <link id="link1" bandwidth="10Mbps" latency="10ms"/>
    <link id="link2" bandwidth="1Gbps" latency="20ms"/>

    <route src="H1" dst="H2">
      <link_ctn id="link1"/>
    </route>

    <route src="router1" dst="router2">
```

```
      <link_ctn id="link1"/>
      <link_ctn id="link2"/>
    </route>

  </zone>
</platform>
```

Código 2.1: Arquivo platform.xml (XML que descreve uma plataforma no Simgrid).

Essencialmente, três tipos de recursos são representados através de *tags* no XML: *hosts*, *links* e roteadores (CASANOVA et al., 2014). A *tag* ***host*** descreve uma máquina física a ser simulada. Cada *host* possui um nome que serve como um identificador para a referência deste *host* durante a simulação. Além disso, o *host* também possui um poder computacional associado, que é uma medida expressada em Flops (operações de ponto flutuante por segundo). A *tag* ***link*** define um *link* de rede. Cada *link* também possui um identificador para propósitos de roteamento e apresenta medidas de largura de banda e latência. Por fim, a *tag* ***router*** declara um roteador na rede. Os roteadores são usados para unir dois ou mais *links* e basicamente são *hosts* que não realizam o processamento de tarefas. O arquivo de plataforma também carrega informações sobre rotas, ou seja, traça os caminhos (sequência de *links*) utilizados para a transmissão de dados entre dois *hosts*. O Código 2.1 mostra um exemplo de arquivo de plataforma no Simgrid, com uma topologia simples composta por dois *hosts* (H1 e H2), dois roteadores (router1 e router2) e dois *links* (link1 e link2). A comunicação entre os dois *hosts* é feita por uma rota através do primeiro *link*. Já a comunicação entre os dois roteadores ocorre por uma rota que passa pelos dois *links*. O exemplo apresenta uma plataforma heterogênea, pois os *hosts* possuem diferentes capacidades computacionais e também os *links* se diferem em latência e largura de banda.

2.6.2 Aplicação

A aplicação distribuída representa uma parte ativa em qualquer sistema de computação em larga escala, sendo responsável pela criação de ações de comunicação e computação na plataforma durante o processo de execução do sistema. Uma aplicação no Simgrid realiza a produção e simulação de várias ações de transmissão de dados ou execução de cargas de trabalho nos recursos simulados na plataforma (VELHO, 2011). O Simgrid funciona como uma biblioteca. As aplicações no Simgrid podem ser expressas usando diferentes interfaces de programação, tais como:

- Interface S4U (em C++);

- Simulação de programas MPI com SMPI (em C, C++ ou Fortran);
- Interfaces legadas: MSG para algoritmos distribuídos (em C e Java) e SimDAG para algoritmos centralizados (em C).

É possível notar que o Simgrid é bastante modular e oferece diversas formas de uso. No contexto deste trabalho, é utilizada a interface S4U (*Simgrid for you*), que de acordo com a documentação do Simgrid, é a interface preferida para descrever algoritmos abstratos nos domínios de configurações de nuvem, P2P, HPC, IoT e similares. As simulações do Simgrid são compostas por vários atores, que executam funções fornecidas pelo usuário. Do modo a expressar ações de computação, comunicação e outras atividades, os atores fazem uso da interface S4U. Cada ator está localizado em um *host* simulado e pode se comunicar com outros atores através de mensagens enviados para a uma *mailbox*, que serve como um ponto de encontro entre atores comunicantes. Além disso, os atores podem interagir através de mecanismos clássicos de sincronização, como barreira, semáforo, exclusão mútua e variáveis de condição. O Simgrid prevê o tempo gasto por cada atividade e orquestra os atores de acordo.

```
#include <simgrid/s4u.hpp>

// classe master (processo mestre)
class Master {

    ...

    void operator()(){

        // obtem a mailbox do escravo (worker)
        simgrid::s4u::MailboxPtr mailbox = simgrid::s4u::Mailbox::
            by_name("worker-1");

        // envia uma tarefa para o worker
        mailbox->put(new double(compute_size), communicate_size);

    }
};

// classe worker (processo escravo)
class Worker {

    ...
```

```
    void operator()(){

        // mailbox de chegada de mensagens
        simgrid::s4u::MailboxPtr mailbox = simgrid::s4u::Mailbox::
            by_name("worker-1");

        // espera ate chegar uma tarefa
        double* task  = static_cast<double*>(mailbox->get());
        double compute_size = *task;
        delete task;

        // processa a tarefa
        simgrid::s4u::this_actor::execute(compute_size);

    }

};

// funcao principal
int main(int argc, char* argv[]){
    simgrid::s4u::Engine e(&argc, argv);

    // registra as classes que representam os atores
    e.register_actor<Master>("master");
    e.register_actor<Worker>("worker");

    // carrega o arquivo de plataforma
    e.load_platform(argv[1]);

    // carrega o arquivo de implantacao da simulacao
    e.load_deployment(argv[2]);

    // executa a simulacao
    e.run();

    XBT_INFO("Tempo de simulacao: %g", e.get_clock());

    return 0;
}
```

Código 2.2: Arquivo application.cpp (Exemplo de aplicação no Simgrid).

O Código 2.2 ilustra um exemplo de código em C++ para a descrição de uma aplicação

utilizando a API S4U. Trata-se de uma aplicação mestre-escravo, em que um ator mestre envia uma tarefa para um ator escravo, que executa essa tarefa. O carregamento dos arquivos com a descrição da plataforma e mapeamento entre processos (atores) e *hosts* ocorre na função *main*.

2.6.3 Implantação

O arquivo de implantação (ou *deployment*, na língua inglesa), é um arquivo XML que descreve o mapeamento entre processos e *hosts* no início da simulação. Além disso, ajuda a especificar os parâmetros necessários para a instanciação de cada processo.

```
<?xml version='1.0'?>
<!DOCTYPE platform SYSTEM "simgrid/simgrid.dtd">
<platform version="4.1">
  <!-- processo mestre (com alguns argumentos) -->
  <actor host="H1" function="master">
    <!-- tamanho da tarefa -->
    <argument value="50000000"/>
    <!-- quantidades de dados a serem transmitidos -->
    <argument value="1000000"/>
  </actor>

  <!-- processo escravo -->
  <actor host="H2" function="worker">
    <argument value="worker-1"/> <!-- identificador do escravo -->
  </actor>

  <!-- processo escravo -->
  <actor host="H2" function="worker">
    <argument value="worker-2"/> <!-- identificador do escravo -->
  </actor>
</platform>
```

Código 2.3: Arquivo deployment.xml (XML de implantação no Simgrid).

O Código 2.3 apresenta um exemplo de arquivo XML de implantação que relaciona os *hosts* da plataforma no Código 2.1 com os atores (processos) da aplicação no Código 2.2. O arquivo também define os valores dos argumentos que cada processo receberá. É importante destacar que o uso do arquivo de implantação é opcional, pois a API S4U possui funções para a criação de processos nos *hosts* da plataforma.

2.6.4 Arquitetura do Simgrid

O Simgrid apresenta uma arquitetura organizada em diferentes camadas (Figura 2.7). A camada de mais alto nível fornece vários ambientes de programação construídos em cima de um *kernel* de simulação único. É usada para definir modelos de aplicações, onde os usuários podem utilizar as diferentes interfaces (MSG, SMPI, SimDAG, S4U) para simular as ações sobre os recursos da plataforma (CASANOVA et al., 2014). A Camada do *kernel* de simulação é uma camada de baixo nível que serve como base para a camada de ambientes de programação. Essa camada fornece as principais funcionalidades para simular uma plataforma virtual através do módulo chamado SURF. O SURF é responsável pela predição do tempo necessário para a finalização das ações de comunicação e computação da aplicação. O Simgrid também possui uma interface, chamada SIMIX que funciona como um sistema operacional leve e orquestra *threads* na simulação de uma forma determinística. O SIMIX fornece chamadas básicas de sistema que podem ser usadas pelas diversas interfaces de programação disponíveis no Simgrid (VELHO, 2011). Além disso, o Simgrid também apresenta uma camada base para o *kit* de ferramentas, chamada XBT (*Extensible Box of Tools*). Esse conjunto de ferramentas é uma biblioteca portátil que fornece recursos como registro de *logs*, suporte a exceções (C e C++) e estruturas de dados (em C).

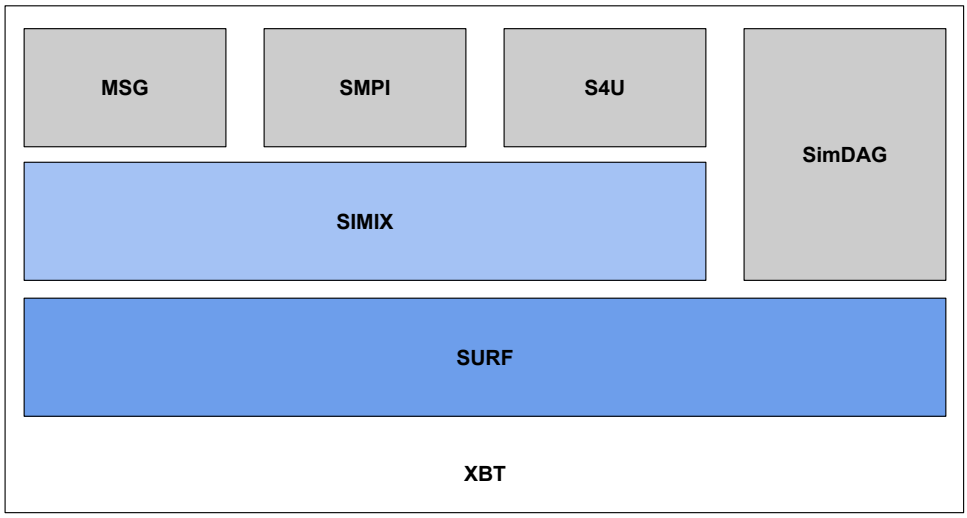


Figura 2.7: Arquitetura do Simgrid (adaptado de Velho (2011)).

2.6.5 Processo de Simulação

A simulação no Simgrid se inicia com o carregamento do arquivo de plataforma que mapeia os recursos definidos pelo usuário para o SURF. Na sequência são criados os processos (através do carregamento do arquivo de implantação). Nesse momento diversas *threads* (uma

para cada processo) são criadas no SIMIX. Essas *threads* são responsáveis por executar o código definido pelo usuário. Então na medida em que a simulação acorre, o SIMIX orquestra as *threads* sequencialmente, de modo a prover um comportamento repetível. A API utilizada (MSG, SMPI, SimDAG, S4U) fornece uma série de primitivas de simulação que bloqueiam o código do usuário até algum evento ocorrer. Quando todos processos estivem aguardando por uma primitiva de simulação para completar, o SIMIX consulta o SURF para saber qual a ação a ser executada. Sempre que uma ação é criada no SURF, ela tem associada uma quantidade de trabalho a fazer e uma quantidade de trabalho feita. Dentro do SURF, o compartilhamento de ações que concorrem pelo uso dos mesmos recursos é resolvido por um modelo que estima o tempo de finalização das ações de comunicação e computação. Após isso, o SURF retorna para o SIMIX uma lista das ações que devem finalizar primeiro e atualiza a quantidade de trabalho realizada por cada ação. Esse processo se repete até todas as *threads* finalizarem (VELHO, 2011).

2.7 Aplicações *Bag-of-Tasks*

O modelo de aplicação estudado neste trabalho, são as aplicações *Bag-of-Tasks* (BoT). De modo geral, uma aplicação BoT A é composta de k tarefas independentes em que a carga de trabalho associada a cada tarefa pode variar entre as tarefas. Por serem independentes, as tarefas não se comunicam entre si e podem ser executadas em qualquer ordem. Para ser processada, cada tarefa depende de um ou mais arquivos de entrada, que pode ser compartilhado entre as tarefas. Ao final do processamento, cada tarefa gera seu próprio conjunto de arquivos de saída, que pode ser composto por um ou mais arquivos.

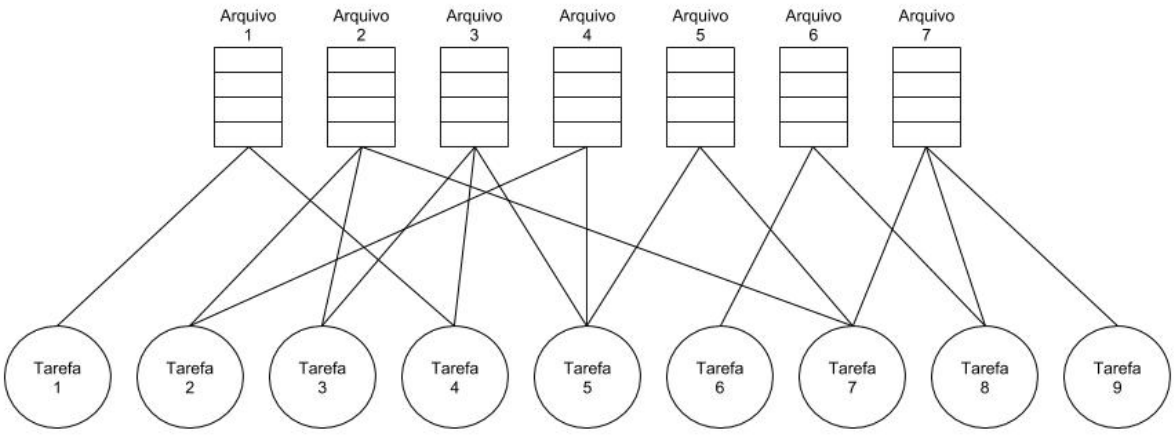


Figura 2.8: Representação de uma aplicação BoT com 9 tarefas e 7 arquivos de entrada (adaptado de Silva e Senger (2009))

A Figura 2.8 representa uma aplicação BoT como um grafo bipartido, em que o primeiro conjunto de nós representa os arquivos de entrada, o segundo conjunto representa as tarefas, e as arestas representam as dependências entre as tarefas e os arquivos (GIERSCH; ROBERT; VIVIEN, 2006). As aplicações BoT são frequentes em áreas como astronomia, física, bioinformática, entre muitas outras. Exemplos de aplicações BoT incluem simulações de Monte Carlo, algoritmos de busca (como quebra de chaves), manipulação de imagens, e algoritmos de mineração de dados (SILVA; SENGER, 2009).

2.8 Escalabilidade de Sistemas

A escalabilidade pode ser definida como a habilidade de um sistema para aumentar o *speedup* à medida que o número de processadores aumenta (GRAMA; GUPTA; KUMAR, 1993). Uma outra definição diz que uma combinação entre algoritmo e máquina é escalável se a velocidade média alcançada pelo algoritmo na máquina em questão, permanecer constante com o aumento do número de processadores, desde que o tamanho do problema possa ser aumentado com o tamanho do sistema (SUN; ROVER, 1994).

Kumar e Rao (1987) propuseram o conceito de função de isoeficiência para caracterizar a escalabilidade de um algoritmo em uma determinada arquitetura. A abordagem consiste em ajustar a eficiência até um certo valor desejado e calcular o quanto a carga de trabalho deve ser aumentada para manter a eficiência inalterada na medida que o número de máquinas aumenta. A função de isoeficiência $F(P)$ relaciona o tamanho da máquina P com a quantidade de trabalho W necessária para manter a eficiência. A quantidade de trabalho W é definida como a soma da quantidade de computação de todas as tarefas que compõem a aplicação. A equação 2.1 ilustra a definição de eficiência em computação paralela.

$$E = \frac{S}{P} \tag{2.1}$$

Na equação 2.1 a eficiência é definida como o *speedup* S dividido pelo número de processadores P. O *speedup* por sua vez, é definido pela equação 2.2, onde T_{seq} é o tempo necessário para a execução de todas as tarefas em um único processador, e T_P é o tempo necessário para a execução de todas as tarefas em P processadores.

$$S = \frac{T_{seq}}{T_P} \tag{2.2}$$

De acordo com as equações 2.1 e 2.2, a definição de eficiência pode ser sintetizada na

equação 2.3.

$$E = \frac{\frac{T_{seq}}{T_P}}{P} \tag{2.3}$$

As equações acima são válidas para plataformas homogêneas. Entretanto, a literatura apresenta uma proposta de eficiência heterogênea, que estende a análise de isoeficiência para máquinas heterogêneas (PASTOR; BOSQUE, 2001) (BOSQUE; PEREZ, 2004) (BOSQUE et al., 2011). A eficiência heterogênea é definida na equação 2.4.

$$E = \frac{W}{T_R \cdot C_T} \tag{2.4}$$

Na equação 2.4, W é a quantidade total de trabalho, T_R é o tempo de execução da aplicação e C_T é o poder computacional total da plataforma. O poder computacional da plataforma é definido (equação 2.5), como o somatório do poder computacional C_i de cada máquina i que compõe a plataforma. O poder computacional pode ser expresso em termos de alguma métrica de interesse, como por exemplo, a capacidade de realização de operações de ponto flutuante por segundo (FLOP/s).

$$C_T = \sum_{1}^{P} C_i \tag{2.5}$$

A escalabilidade pode ser obtida de forma horizontal (*scale-out*, na língua inglesa), quando se adiciona nós ao sistema. Esse é o método mais comumente utilizado quando se utilizam recursos na nuvem (AL-DHURAIBI et al., 2018). A escalabilidade também pode ser implementada de forma vertical (*scale-up*, na língua inglesa), aumentando a quantidade de recursos (CPU, memória, velocidade de I/O, armazenamento) de um nó do sistema. A escalabilidade vertical também pode ser facilmente obtida por usuários de infraestrutura como serviço (IaaS), que podem escolher as configurações (e o preço) de suas instâncias.

2.9 *Input File Affinity* (IFA)

Silva e Senger (2009) introduziram o conceito de *Input File Affinity* (IFA) para medir o grau de afinidade de um conjunto de tarefas de acordo com seus arquivos de entrada. De acordo com os autores, dado um conjunto G composto por K tarefas, $G = \{T_1, T_2, ..., T_K\}$, e um conjunto F de Y arquivos de entrada necessários para a execução das tarefas do grupo G, $F = \{f_1, f_2, f_3, ..., f_Y\}$, o IFA é definido como:

$$IFA(G) = \frac{\sum_{i=1}^{Y}(N_i - 1)|f_i|}{\sum_{i=1}^{Y} N_i|f_i|} \tag{2.6}$$

Na equação 2.6, $|f_i|$ é tamanho do arquivo f_i em *bytes* e $N_i(1 \leq N_i \leq K)$ é o número de tarefas do grupo G que dependem do arquivo f_i. O termo $(N_i - 1)$ no numerador ocorre, porque um arquivo é enviado apenas uma vez para a máquina escrava que irá processar as tarefas que dependem desse arquivo. Vale notar que $0 \leq IFA < 1$. O IFA igual a zero indica que não existe nenhum arquivo compartilhado entre as tarefas. Portanto, quanto maior o valor do IFA, maior será a quantidade de *bytes* de dados de arquivos que são compartilhados.

Capítulo 3

TRABALHOS RELACIONADOS

3.1 Escalabilidade em Plataformas Distribuídas

Aplicações BoT compostas por tarefas independentes com compartilhamento de arquivos foram alvo de estudo em muitos trabalhos anteriores (BEAUMONT et al., 2008; BERMAN et al., 2003; CASANOVA et al., 2000; CIRNE et al., 2003). Plataformas comumente utilizadas para executar aplicações BoT geralmente implementam o modelo mestre-escravo, que possui algumas limitações fundamentais: a comunicação do nó mestre e o acesso ao repositório de arquivos centralizado podem se transformar em gargalos para a execução das aplicações, e portanto, limitando a escalabilidade. Casanova et al. (2000) propuseram heurísticas que consideram o compartilhamento de arquivos, de modo que os arquivos de entrada previamente transferidos para processadores não precisam ser retransferidos. Essa melhoria reduz o gargalo no nó mestre. Giersch, Robert e Vivien (2006) provaram limites teóricos para a complexidade computacional associada a algumas instâncias do problema de escalonamento e propuseram várias novas heurísticas capazes de produzir escalonamentos que abordam o desempenho alcançado pelas heurísticas propostas por Casanova et al. (2000), porém mantendo a complexidade computacional uma ordem de grandeza menor.

A métrica de isovelocidade (*isospeed*, na língua inglesa) foi proposta por Sun e Rover em (SUN; ROVER, 1994), para estudar a escalabilidade de sistemas de computação paralela. Na isovelocidade, a velocidade média da aplicação é fixada e estima-se a escalabilidade em termos do aumento na quantidade de trabalho necessário para manter a velocidade enquanto número de processadores é aumentado. Sun e Rover (1994) estudam também a extensão da métrica de isovelocidade para sistemas computacionais heterogêneos. Os autores demonstram que a isovelocidade funciona bem em ambientes homogêneos e heterogêneos. No entanto, o estudo não se concentra nem em uma plataforma específica nem em uma classe de aplicações específica

(por exemplo, aplicações BoT).

Kumar e Rao (1987) propuseram a função de isoeficiência (*isoefficiency*, na língua inglesa) para caracterizar a escalabilidade de um algoritmo em uma determinada arquitetura. Essa função ajusta a eficiência até um certo valor desejado e calcula qual deve ser o aumento da quantidade de carga de trabalho para manter a eficiência inalterada na medida que o número de máquinas aumenta. Pastor e Bosque (2001), Bosque e Perez (2004) e Bosque et al. (2011) propuseram uma função de eficiência para sistemas computacionais heterogêneos. Seu trabalho amplia a noção de modelo de escalabilidade de isoeficiência homogêneo para sistemas de computação heterogêneos.

Anteriormente, foi estudada a escalabilidade de aplicações BoT com compartilhamento de arquivos em áreas como mineração de dados (SENGER et al., 2007) e alinhamento de sequências (bioinformática) (CASTRO et al., 2017). Em (SILVA; SENGER, 2009), foi desenvolvido um estudo mais teórico que demonstra que o limite inferior da função de isoeficiência de aplicações BoT executando em plataformas mestre-escravo é $\Omega(P^2)$. Ou seja, para compensar o aumento dos custos de comunicação decorrentes do aumento do tamanho da plataforma e manter a eficiência em um patamar fixo, é preciso aumentar a carga de trabalho proporcionalmente a P^2, onde P é o número de nós. Nesse mesmo trabalho também foi proposto um algoritmo de escalonamento (chamado *Dynamic Clustering* - DC) que agrupa tarefas que compartilham arquivos e melhora a escalabilidade das aplicações. Buscando plataformas mais escaláveis, em (SILVA; SENGER, 2011, 2015) foi realizado um estudo sobre plataformas hierárquicas, como extensões do mestre-escravo puro. Nesse trabalho, foi demonstrado que o limite de escalabilidade para a execução de aplicações BoT é $\Omega(P \log P)$ (ou seja, uma ordem de grandeza mais escalável do que o mestre-escravo puro) para vários modelos de comunicação incluindo *broadcast* e enlaces TCP). Em (SENGER; SILVA, 2012), foi avaliado o impacto da contenção causada pela transmissão de arquivos de saída na escalabilidade de aplicações BoT. Todos os estudos apresentados em (SENGER et al., 2007; SILVA; SENGER, 2009, 2011; SENGER; SILVA, 2012; SILVA; SENGER, 2015; CASTRO et al., 2017) trataram da escalabilidade de plataformas homogêneas. Contudo, grande parte das aplicações BoT é executada em plataformas heterogêneas. Assim, o presente trabalho apresenta um estudo da escalabilidade de aplicações BoT considerando plataformas mestre-escravo compostas de nós heterogêneos. A computação heterogênea tem sido estudada por mais de duas décadas. Porém, poucos estudos avaliam os efeitos dessa heterogeneidade sobre a escalabilidade das aplicações.

Yero e Henriques (2007) apresentam uma análise de escalabilidade de aplicações mestre-escravo em *clusters* heterogêneos. Inicialmente, os autores desconsideraram os efeitos da

contenção de comunicação no modelo de execução, mas quando os autores explicitamente consideraram a contenção, eles concluíram que o sistema não é escalável quando a contenção é proporcional ao número de processadores. No presente estudo, é demonstrado que o par composto de plataformas mestre-escravo heterogêneas e aplicações BoT pode ser escalável, mesmo quando a contenção é proporcional ao número de processadores, desde que a quantidade de trabalho possa ser aumentada de acordo.

Rosenberg e Chiang (2010, 2011), fizeram uma análise dos efeitos da heterogeneidade em plataformas distribuídas. As análises e resultados derivados deste trabalho são muito interessantes, com destaque para as seguintes conclusões: (i) se alguém pode substituir apenas um computador em um *cluster* por um mais rápido, é provável que (quase) sempre seja mais vantajoso substituir o mais rápido; (ii) se os computadores em dois *clusters* tiverem a mesma velocidade média, então o *cluster* com a maior variação na velocidade é (quase) sempre o mais rápido; (iii) heterogeneidade pode realmente dar poder a um *cluster*. Contudo, os autores não focam em uma arquitetura ou aplicação específicos.

O presente trabalho apresenta um estudo sobre a escalabilidade de plataformas heterogêneas com foco em sistemas onde o poder de processamento dos nós de computação pode variar. Até onde se sabe, não há estudo que avalie a escalabilidade para a execução de aplicações BoT com compartilhamento de arquivos em plataformas mestre-escravo altamente heterogêneas como as analisadas neste trabalho.

3.2 Provisionamento de Recursos

Este trabalho também apresenta um algoritmo que explora a heterogeneidade da plataforma para selecionar um conjunto de instâncias a serem provisionados para executar uma determinada aplicação. A literatura apresenta muitos trabalhos que propõem soluções para o problema do provisionamento de recursos na nuvem. A seguir são descritos alguns desses trabalhos que enfatizam, principalmente, o problema da seleção de um conjunto de instâncias para provisionamento.

Ashwini, Divya e Sanjay (2013) propõe um *framework* para agrupar as melhores máquinas virtuais para a execução de aplicações de HPC. Esse trabalho também parte do princípio de que em uma nuvem existem diversos recursos disponíveis como computação, armazenamento e recursos de comunicação. E esses recursos são heterogêneos por natureza. E portanto, a escolha do melhor conjunto de recursos para a execução das aplicações é um problema desafiador. O *framework* proposto considera o poder computacional e a largura de banda disponível entre as

VMs para selecionar o melhor conjunto de instâncias para a execução de aplicações de HPC. O *framework* é composto por dois módulos:

- No primeiro módulo é feito o agrupamento das VMs baseado na capacidade de CPU. Eles utilizam o algoritmo de agrupamento de dados *K-means* para agrupar as máquinas virtuais similares. Nesse contexto, a similaridade é definida pelo tempo de execução de uma aplicação de referência nas VMs.

- No segundo módulo ocorre a escolha das melhores VMs do *cluster* selecionado no primeiro módulo, baseado na largura de banda. Na implementação deste módulo é utilizado o NWS (*Network Wheather Service*) para realizar o monitoramento da rede. Dessa forma, é medida a largura de banda disponível em um *link* entre duas VMs. No final, são escolhidas as *N* máquinas mais eficientes no melhor *cluster* e que satisfazem o critério de largura de banda.

Como resultado, é observado que as máquinas virtuais selecionadas pelo método proposto são aproximadamente as mesmas de resultados utilizando técnicas de força bruta. Além disso, o *framework* resulta em um eficiente conjunto de VMs para a execução de aplicações de HPC. Entretanto, a proposta não explora muito bem a heterogeneidade da nuvem, pois agrupa as VMs de acordo com a capacidade de CPU.

Silva, Veiga e Ferreira (2008) apresenta uma heurística para a otimização do número de máquinas que devem ser alocadas para o processamento de tarefas de aplicações *Bag-of-Tasks*, de modo a alcançar o máximo *speedup* com orçamento limitado. O trabalho afirma que sem o conhecimento prévio do tempo de processamento de cada tarefa, é difícil determinar a quantidade de máquinas a serem criadas. A criação de muitas VMs aumenta o *speedup*, mas também eleva o custo. Já a criação de poucas instâncias implica em baixo custo, mas deixa o *speedup* abaixo das expectativas. No trabalho, o tempo de execução de cada tarefa é variável e não é conhecido antecipadamente. Para definir a quantidade de máquinas, é necessário conhecer o tempo de execução das tarefas. Nesse processo, inicialmente, um *host* é alocado e inicia a execução de uma tarefa aleatória. Após concluir a tarefa, o tempo de execução é obtido e usado para calcular o tempo médio de processamento de uma tarefa. Esse processo continua com a alocação de mais *hosts* e execução de mais tarefas. Ao final de cada tarefa, o tempo médio de execução é calculado e usado na predição do número de *hosts* necessários para concluir o trabalho com o mínimo de custo. Os resultados mostram que a heurística determina o número de máquinas necessárias para garantir que o tempo de cobrança seja próximo ao valor desejado. Além disso, apresenta *speedup* próximo ao número de máquinas alocadas. Con-

tudo, a heurística não considera a heterogeneidade da plataforma, uma vez que a nuvem possui máquinas com diferentes capacidades computacionais e preços.

Hwang e Kim (2012) propõe duas abordagens para o provisionamento de recursos para a execução de aplicações *MapReduce* com restrições de prazo. O objetivo é minimizar o custo das máquinas virtuais para a execução das aplicações. A primeira proposta é baseada nas políticas de preços das instâncias. A política de preço de uma VM é definida pela performance da VM em MIPS (*Million Instruction Per Second*) e pelo custo do uso em segundos. Nessa proposta, as políticas de preços das VMs disponíveis são ordenadas em uma ordem específica, de acordo com a performance ou custo, e alocadas para tarefas de *map* e *reduce*. Esse processo acorre até o tempo limite ser alcançado ou não haver mais políticas de preço. A segunda abordagem é baseada em tarefas com consciência de tempo limite. Nesse caso, o tempo limite para a conclusão é divido em dois sublimites, um sublimite para as tarefas *map* e o outro para as tarefas *reduce*. Em seguida, as tarefas são alocadas para as VMs na medida em que o tempo de execução delas seja menor que o seu sublimite. As avaliações foram feitas através de simulações, usando o CloudSim[1], e as propostas foram analisadas em diversos aspectos. É possível notar que a primeira abordagem prioriza as máquinas virtuais mais baratas, enquanto que a segunda abordagem prioriza as máquinas virtuais mais potentes.

Gutierrez-Garcia e Sim (2012) propõe um algoritmo genético para estimar um conjunto subótimo de recursos computacionais para a execução de aplicações *Bag-of-Tasks* com restrições de orçamento e de prazo. O algoritmo assume que a carga de trabalho da aplicação (em termos de MIPS) é conhecida e com base nos tipos, disponibilidade e preços das instâncias da nuvem, estima a quantidade de recursos a serem alocados e o tempo (em horas) de alocação de cada recurso. O cromossomo do algoritmo genético é definido por um grupo de pares de genes. Cada par de gene é composto por: (a) o número de recursos a serem alocados para um dado tipo de instância, (b) o número de horas para a alocação desses recursos. Portanto, o cromossomo possui o tamanho de $2n$ genes, onde n representa a quantidade de tipos de recursos computacionais disponíveis no ambiente da nuvem. A função de *fitness* é determinada pelos seguintes fatores:

- Quantidade de instruções que podem ser computadas com os recursos;
- Custo de alocação dos recursos;
- Número máximo de horas alocadas para cada tipo de recurso;
- Número máximo de recursos que podem ser alocados, de acordo com a disponibilidade do provedor.

[1] CloudSim: http://www.cloudbus.org/cloudsim/

Os resultados demonstram que o algoritmo proposto pode estimar conjuntos de instâncias de forma autônoma para a execução de aplicações *Bag-of-Tasks* com restrições orçamentárias e de prazos. Esses conjuntos são compostos por recursos mais econômicos (mas mais lentos) na presença de prazos maiores e mais poderosos (mas mais caros) na presença de grandes orçamentos. A abordagem não considera o tempo para transmitir os arquivos de entrada das tarefas, e desta forma, despreza o *overhead* de comunicação. Além disso, assume que o tamanho de cada tarefa é conhecido, o que não é muito realista na prática.

Abdi et al. (2017) apresenta um modelo de programação matemática para o problema de alocação de recursos em nuvens federadas. O trabalho dispõe de uma federação de múltiplas nuvens que proveem tipos de instâncias com diferentes preços e níveis de performance. O modelo proposto é um problema de programação linear binário que considera o tempo limite para a execução de aplicações *Bag-of-Tasks* e as restrições de recursos da federação de nuvens. O objetivo é minimizar o custo total de execução das aplicações. O modelo, o objetivo e as restrições da problema são representados usando funções e expressões matemáticas. Além disso, o modelo matemático assume que o número de tarefas em cada aplicação e o tempo de execução das tarefas (em uma VM de referência) são conhecidos. Também é considerado o *overhead* da transferência de dados. Os resultados mostram que o custo ótimo na federação de nuvens é menor do que o custo apresentado por um único provedor de nuvem. De acordo com os autores, a otimização em nuvens federadas é menos sensível aos dados de entrada da formulação do problema.

Alguns dos trabalhos apresentados acima consideram aplicações *compute-intensive*, e dessa forma desprezam o *overhead* de transferência de dados. Entretanto, muitas aplicações *Bag-of-Tasks* são *data-intensive*, e neste caso, a transferência dos arquivos de entrada para o processamento das tarefas apresenta um peso elevado no desempenho da aplicação. Além disso, a maioria dos trabalhos assume que a carga de trabalho da aplicação é conhecida antecipadamente, mas na prática é complicado definir com exatidão o tamanho de cada tarefa. Este trabalho contribui com uma solução de provisionamento para melhorar o desempenho da aplicação (e consequentemente reduzir os custos). Para isso, pretende-se explorar a heterogeneidade da plataforma de modo a reduzir o *overhead* de transferência de dados de aplicações *Bag-of-Tasks*. O algoritmo proposto é oblívio em relação ao tamanho de cada tarefa, ou seja, a priori o tempo de execução da tarefa não é conhecido.

Capítulo 4

ANÁLISE DE ESCALABILIDADE DE APLICAÇÕES BOT EM PLATAFORMAS HETEROGÊNEAS

Rosenberg e Chiang afirmam que a heterogeneidade pode realmente dar poder a um *cluster* (ROSENBERG; CHIANG, 2010) (ROSENBERG; CHIANG, 2011). Este capítulo tem o objetivo de verificar a veracidade dessa afirmação em termos de escalabilidade, para a execução de aplicações BoT em plataformas mestre-escravo. Para isso, são realizados experimentos que comparam a escalabilidade de plataformas homogêneas e heterogêneas, usando a função de iso-eficiência. Inicialmente são definidos os modelos de aplicação e plataforma, e na sequência são descritos os diferentes conjuntos de experimentos com as devidas análises.

4.1 Modelo de Aplicação

Tipicamente, uma aplicação BoT A é composta por k tarefas independentes, em que a quantidade de computação associada a cada tarefa é pré-definida e pode variar entre as tarefas. No caso de tarefas homogêneas, cada tarefa executa uma quantidade de computação w_i, sendo que w_i é o mesmo valor para todas as tarefas. Já em tarefas heterogêneas, as quantidades de computação w_i e w_j podem ser diferentes, para tarefas t_i e t_j distintas. Em qualquer caso, a quantidade de computação w_i é fixa e não pode ser modificada arbitrariamente.

Para executar, cada tarefa depende de um ou mais arquivos de entrada que podem ser compartilhados entre duas ou mais tarefas. O modelo de aplicação dos experimentos considera IFA máximo, ou seja, todas as tarefas dependem do mesmo arquivo de entrada. Assume-se que a quantidade de dados associada a cada tarefa (tamanho dos arquivos de entrada) é fixo e não pode ser alterado arbitrariamente. Tais suposições são válidas para aplicações reais, tais como mineração de dados (como discutido em (SENGER et al., 2007)), reconhecimento de

padrões, *ray tracing*, simulações de Monte Carlo, e mapeamento de cromossomos. A classe de aplicações apresentada aqui não permite divisão arbitrária da computação entre diferentes processos, ou seja, não são consideradas tarefas com cargas de trabalho divisíveis (BEAUMONT; LEGRAND; ROBERT, 2003).

4.2 Modelo de Plataforma

A plataforma mestre-escravo (ou *master-slave*, na língua inglesa) é uma abstração que representa a maioria das plataformas utilizadas para a execução de aplicações BoT (GIERSCH; ROBERT; VIVIEN, 2006) (BEAUMONT et al., 2008). Em uma plataforma mestre-escravo, os nós escravos têm a função de executar as tarefas da aplicação sob a supervisão centralizada de um nó mestre. Em plataformas mestre-escravo heterogêneas, nós (com múltiplos processadores) são organizados como uma árvore de dois níveis (Figura 4.1). O primeiro nível (topo) é composto por um nó mestre, que é responsável pelo escalonamento das tarefas entre os diversos nós escravos e coleta de resultados. No segundo nível da hierarquia, nós folha (escravos) apenas realizam computações. Uma vez que a plataforma é heterogênea, o poder computacional de cada nó escravo pode variar.

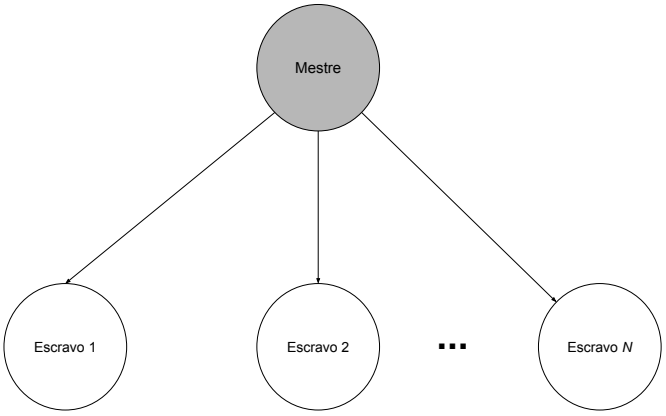


Figura 4.1: Topologia básica de uma plataforma mestre-escravo.

Também considera-se que todos os arquivos de entrada são inicialmente armazenados no nó mestre, que serve como um repositório para todos os arquivos de entrada e de saída (contendo resultados). Cada arquivo de entrada requerido por uma tarefa tem que ser transferido do nó mestre para o nó escravo responsável pela execução da tarefa apenas se o arquivo ainda não existir nesse nó. Nós escravos possuem um disco local onde arquivos temporários podem ser armazenados. A transmissão de dados de novas tarefas do nó mestre para um determinado nó escravo pode ocorrer em paralelo com a execução de uma tarefa anteriormente escalonada para este nó.

4.3 Configuração do Ambiente de Experimentação

Para maior realismo, alguns parâmetros do simulador foram calibrados com base no desempenho de uma plataforma real. Para modelar a capacidade da rede, foram feitas medições com duas máquinas virtuais executando em diferentes máquinas físicas da Cloud@UFSCar, com o intuito de medir a largura de banda e a latência da rede que interliga os nós. Para medir a largura de banda foi utilizado o Iperf[1] e para medir a latência foi utilizado o Qperf[2]. As capacidades aferidas foram largura de banda média de 3,4 Gbps e uma latência média de 100 μs. Os experimentos foram feitos utilizando o Simgrid[3] e foram executados em uma máquina virtual com 364 GB de memória e 40 vCPUs da nuvem da Universidade Federal de São Carlos (UFSCar) - Cloud@UFSCar[4].

4.4 Primeiro Experimento: Plataforma Homogênea *vs.* Heterogênea

4.4.1 Objetivo

O primeiro experimento tem por objetivo verificar se plataformas mestre-escravo heterogêneas podem apresentar ganhos na execução de aplicações BoT (em termos de escalabilidade) quando comparadas com plataformas homogêneas de mesma capacidade agregada.

4.4.2 Configuração

Foram simuladas plataformas mestre-escravo de diferentes tamanhos, e para cada tamanho, diferentes graus de heterogeneidade. Foram simuladas aplicações com tarefas homogêneas, de modo a analisar a escalabilidade em condições ideais. As aplicações homogêneas são compostas por tarefas com carga de trabalho de 3×10^{13} operações de ponto flutuante. Uma tarefa com esse tamanho leva cerca de 300 segundos para ser executada na máquina base de 100 GFLOP/s, aproximando a duração de cargas de trabalho típicas para aplicações BoT (IOSUP et al., 2008). O tamanho dos arquivos de entrada é 375 MB, sendo necessários cerca de 30 segundos para a transferência de um arquivo em um *link* de 100 Mbps. Ou seja, o tempo para executar uma tarefa na máquina de referência é cerca de dez vezes maior do que o tempo requerido para

[1] Iperf: https://iperf.fr/
[2] Qperf: https://github.com/linux-rdma/qperf
[3] Simgrid: http://simgrid.gforge.inria.fr/
[4] Cloud UFSCar: http://portalcloud.ufscar.br/

transmitir a tarefa (CCR - *Communication to Computation Ratio* = 0,1), o que representa uma aplicação com intenso uso de CPU. Estudos anteriores mostram que o CCR não altera o limite assintótico da função de isoeficiência (SILVA; SENGER, 2009).

Os nós escravos têm a função de executar tarefas de uma aplicação BoT, e na prática podem ser implementados por nós de um *cluster*, máquinas em um *data center*, instâncias dedicadas (*bare-metal*) ou virtualizadas (máquinas virtuais ou contêineres) na nuvem. O nó mestre é responsável por despachar tarefas para execução nos nós escravos. A conexão entre o nó mestre e o nós escravos é representada por um *link* de 100 Mbps, que é uma velocidade típica oferecida por muitos provedores de *Internet*. Neste caso, considera-se que o nó mestre representa uma máquina externa de um usuário com acesso à plataforma distribuída, através da *Internet*.

A potência de 100 GFLOP/s para a máquina de referência foi adotada nos experimentos tendo em mente que processadores modernos oferecem desempenhos típicos no intervalo entre 15 a 75 GFLOP/s por núcleo em ponto flutuante de 64 bits (MCINTOSH-SMITH et al., 2018). Como o objetivo é medir a escalabilidade de plataformas heterogêneas, a potência da plataforma é expressa pela sua capacidade agregada (C_T). Neste experimento, a capacidade da plataforma variou de 100 GFLOP/s até 1 PFLOP/s, que seria equivalente a uma máquina com 2 núcleos até um *cluster* com cerca de 20000 núcleos. As plataformas foram geradas da seguinte forma:

- As plataformas homogêneas são geradas com 1, 10, 100, 1000 e 10000 nós com capacidade de 100 GFLOP/s. Nas Figuras 4.2 e 4.3, a curva de isoeficiência para plataformas homogêneas é rotulada como "0%" (0% de heterogeneidade).

- As plataformas heterogêneas são geradas com diferentes níveis de heterogeneidade. Foram simuladas plataformas com 25%, 50%, 75% e 99% de heterogeneidade. O procedimento consiste em: (*i*) seja C_T o poder total da plataforma a ser gerada, e H a porcentagem de C_T destinada para a máquina heterogênea; (*ii*) uma máquina com poder $H \times C_T$ é adicionada à plataforma; e (*iii*) o poder computacional restante é distribuído com a adição de nós homogêneos (com 100 GFLOP/s cada) até que o poder agregado desses nós seja $C_T \times (1-H)$.

Com tarefas homogêneas e plataforma homogênea foi utilizado o escalonamento *round-robin*, que oferece desempenho ótimo (GIERSCH; ROBERT; VIVIEN, 2006). No caso da plataforma heterogênea existe uma segunda etapa de escalonamento dinâmico que permite que nós ociosos roubem tarefas dos demais, balanceando a carga entre os nós. A cada experimento, a isoeficiência é estimada aumentando-se a quantidade de trabalho necessária para atingir a eficiência desejada.

4.4.3 Resultados e Conclusões

Os resultados são ilustrados na Figura 4.2 e Tabela 4.1. O gráfico está apresentado em escala *log-log* para evidenciar o comportamento assintótico de cada função de isoeficiência. O eixo horizontal contém o poder computacional agregado C_T da plataforma simulada, enquanto o eixo vertical representa o valor da função de isoeficiência. Vale destacar que a isoeficiência é dada pela quantidade de trabalho necessário para manter o nível de eficiência da aplicação BoT acima do patamar desejado, no caso 90%. Ou seja, quanto menor o valor (e a taxa de crescimento) da função, mais escalável é a plataforma.

Nível de Heterogeneidade	**Poder Agregado da Plataforma**				
	100 GFLOP/s	**1 TFLOP/s**	**10 TFLOP/s**	**100 TFLOP/s**	**1 PFLOP/s**
0%	3E+13	4,80E+15	3,84E+17	3,07E+19	4,92E+21
25%	3E+13	3,84E+15	2,92E+17	2,31E+19	3,69E+21
50%	3E+13	2,88E+15	1,96E+17	1,54E+19	2,46E+21
75%	3E+13	1,44E+15	9,98E+16	7,71E+18	1,23E+21
99%	3E+13	4,80E+14	3,07E+16	6,76E+17	4,96E+19

Tabela 4.1: Valor da função de isoeficiência para execução de tarefas homogêneas em plataformas mestre-escravo, mantendo a eficiência acima de 90%.

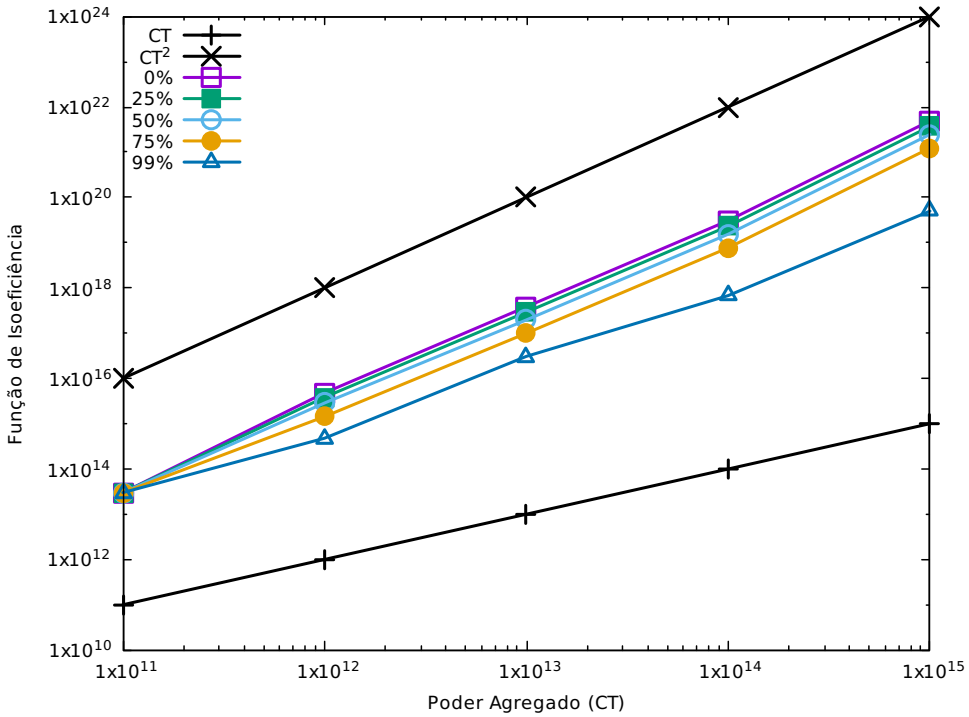


Figura 4.2: Função de isoeficiência para execução de tarefas homogêneas em plataformas mestre-escravo, mantendo a eficiência acima de 90%.

De maneira geral, verifica-se que a isoeficiência da plataforma homogênea (com rótulo de 0%) apresenta um comportamento assintótico similar (mesma taxa de crescimento) ao da curva

$C_T{}^2$. Esse resultado é consistente com (SILVA; SENGER, 2009), onde foi provado que o limite inferior da função isoeficiência de plataformas mestre-escravo homogêneas é $\Omega(P^2)$, onde P é o número de nós. A novidade, contudo, é que as curvas de isoeficiência para plataformas heterogêneas apresentam crescimento inferior à medida que a heterogeneidade é aumentada. A plataforma que possui 99% de sua capacidade alocada em nós heterogêneos apresenta função de isoeficiência com crescimento inferior, bastante próximo ao linear, representado pela curva C_T. Nesse tipo de gráfico, mesmo pequenas diferenças de comportamento assintótico representam grandes diferenças em termos de desempenho da plataforma.

Nível de Heterogeneidade	**Poder Agregado da Plataforma**				
	100 GFLOP/s	**1 TFLOP/s**	**10 TFLOP/s**	**100 TFLOP/s**	**1 PFLOP/s**
0%	3,00E+13	6,00E+14	4,80E+16	3,84E+18	6,14E+20
25%	3,00E+13	4,80E+14	3,65E+16	2,88E+18	4,61E+20
50%	3,00E+13	3,60E+14	2,45E+16	1,92E+18	3,07E+20
75%	3,00E+13	3,60E+14	1,25E+16	9,64E+17	1,54E+20
99%	3,00E+13	6,00E+13	1,92E+15	8,45E+16	6,21E+18

Tabela 4.2: Valor da função de isoeficiência para execução de tarefas homogêneas em plataformas mestre-escravo, mantendo a eficiência acima de 50%.

Um segundo conjunto de execuções deste experimento (Tabela 4.2 e Figura 4.3), foi realizado para um nível de 50% de eficiência desejada. Apesar da curva de isoeficiência ser deslocada para baixo no gráfico, os resultados mantêm o comportamento da função em cada plataforma simulada. Isto confirma descobertas anteriores que mostram que o nível de eficiência escolhido não altera o comportamento assintótico dos limites de isoeficiência (SILVA; SENGER, 2009) (SILVA; SENGER, 2011) (SENGER; SILVA, 2012).

4.5 Segundo Experimento: *Workloads* Típicos de Sistemas Reais

4.5.1 Objetivo

O objetivo deste experimento é verificar se a heterogeneidade pode melhorar a escalabilidade de aplicações com perfis mais realistas, ou seja, mais próximos das cargas de trabalho executadas cotidianamente em plataformas reais de produção ou de pesquisa.

4.5.2 Configuração

Para tal, foi utilizado um gerador de cargas sintéticas apresentado em (CARVALHO; BRASILEIRO, 2012), que reproduz as características (tempo entre submissão dos *jobs*, duração das

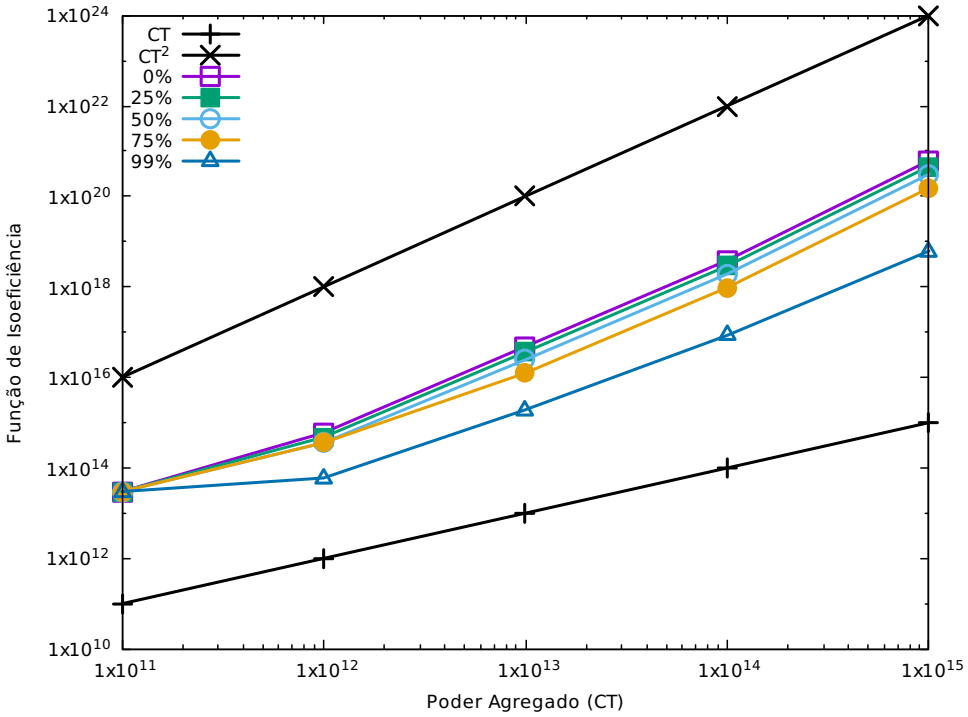


Figura 4.3: Função de isoeficiência para execução de tarefas homogêneas em plataformas mestre-escravo, mantendo a eficiência acima de 50%.

tarefas, duração dos *jobs*, distribuição estatística) de cargas reais executadas em vários centros de supercomputação, segundo rastros de execução publicados no *Grid Workloads Archive* (GWA[5]). A Tabela 4.3 mostra a relação de coleções de rastros utilizados para a construção dos modelos de geração de carga.

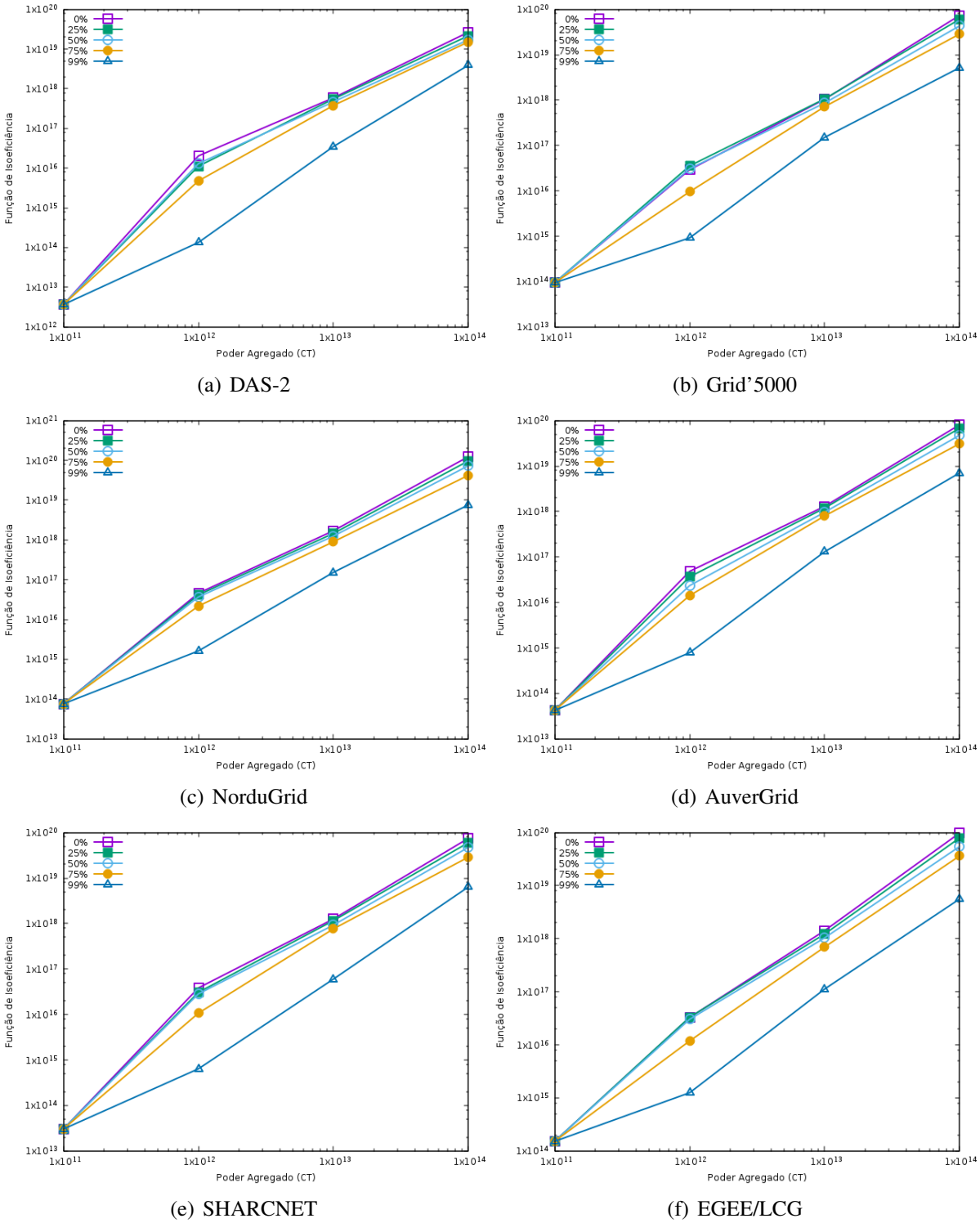
Sistema	Localização	Perfil
DAS-2	Holanda	Acadêmico
Grid'5000	França	Acadêmico
NorduGrid	Europa	Ambos
AuverGrid	França	Produção
SHARCNET	Canadá	Produção
EGEE/LCG	Europa	Produção

Tabela 4.3: Bases de dados utilizadas no gerador de cargas. Adaptado de (CARVALHO; BRASILEIRO, 2012).

O procedimento para computar a função de isoeficiência é similar ao do primeiro experimento. São geradas as plataformas com diferentes tamanhos, variando de 100 GFLOP/s até 100 TFLOP/s. Fixa-se o nível de eficiência desejado (no caso, de 0,9) e em seguida geram-se cargas (*jobs* compostos de tarefas BoT) que são executadas até que seja atingido o patamar de eficiência desejado. As cargas são geradas de acordo com as distribuições de probabilidades

[5]http://gwa.ewi.tudelft.nl/

verificadas em cada grupo de usuários para cada uma das seis plataformas listadas na Tabela 4.3. Os tamanhos dos arquivos são gerados de forma que o CCR médio seja de 0,1, tomando por base a máquina de referência.



(a) DAS-2 (b) Grid'5000

(c) NorduGrid (d) AuverGrid

(e) SHARCNET (f) EGEE/LCG

Figura 4.4: Função de isoeficiência para execução de tarefas heterogêneas em plataformas mestre-escravo, mantendo a eficiência acima de 90%.

4.5.3 Resultados e Conclusões

Os resultados são ilustrados na Figura 4.7. Como se pode observar em todos os casos avaliados, com ligeiras oscilações, as plataformas com mais heterogeneidade são mais escaláveis pois apresentam valores mais baixos para a função de isoeficiência. De fato, o experimento dá evidências de que o limite assintótico da função de isoeficiência de plataformas heterogêneas é inferior à curva P^2 (que é o limite assintótico de plataformas mestre-escravo homogêneas provado em (SILVA; SENGER, 2009)). Em outras palavras, substituir alguns nós de processamento por um único nó maior (de potência equivalente aos nós removidos) confere maior escalabilidade para a execução de aplicações BoT.

4.6 Terceiro Experimento: *Scale-out versus Scale-up*

4.6.1 Objetivo

O objetivo do próximo experimento é verificar se empregar apenas a escalabilidade vertical para adicionar recursos a um sistema mestre-escravo é melhor do que utilizar a escalabilidade horizontal. Mais que isso, deseja-se identificar, em condições ideais, se existe diferença significativa (expressa pela curva de isoeficiência) no caso em que o número de nós é mantido e aumenta-se apenas as suas capacidades (*scale-up* puro) e o caso em que aumenta-se a capacidade adicionando nós homogêneos à plataforma.

4.6.2 Configuração

Para isto, foi preciso criar um cenário que simula o comportamento de um sistema em condições ideais para se obter máxima escalabilidade. O cenário criado foi uma plataforma com um número fixo de 10 nós homogêneos e que vai gradativamente aumentando o poder de cada nó, de modo que a potência agregada da plataforma varia de 100 GFLOP/s a 10 PFLOP/s. Foram simuladas tarefas homogêneas de tamanho 3×10^{12} FLOPs em um único *job* em que todas as tarefas dependem de um mesmo arquivo de entrada, pois estas seriam condições ideais para a melhor escalabilidade (*input file affinity* máximo), conforme provado em (SILVA; SENGER, 2009). Foi utilizado o escalonamento *round-robin*, que é ótimo (GIERSCH; ROBERT; VIVIEN, 2006). Apesar de tais condições ideais serem improváveis em plataformas reais, este experimento tem por objetivo investigar uma questão de fundo teórico, mas que tem implicações práticas como será discutido adiante.

Neste experimento o valor de eficiência foi fixado em 0,25 e em 0,5 (em vez de 0,9 utilizado nos experimentos anteriores). Isto possibilitou reduzir significativamente a quantidade de tarefas simuladas no experimento para se obter a curva de isoeficiência. Conforme demonstrado em (SILVA; SENGER, 2009, 2011) e verificado também na seção 4.4, fixar o nível de eficiência em patamares inferiores não altera o comportamento assintótico da curva de isoeficiência, mas apenas desloca a curva para baixo no gráfico *log-log*.

4.6.3 Resultados e Conclusões

A Figura 4.5 mostra que a curva de isoeficiência para tarefas de 3×10^{12} FLOPs. Nota-se que as duas curvas de eficiência de 0,5 e 0,25 apresentam tendência de crescimento muito abaixo de ${C_T}^2$ (limite inferior da função para plataformas mestre-escravo homogêneas provado em (SILVA; SENGER, 2009)) e até mesmo abaixo da curva $C_T \cdot \log C_T$ (limite inferior para plataformas hierárquicas provado em (SILVA; SENGER, 2011)). Mais que isso, as duas curvas apresentam uma tendência de crescimento bastante similar ao da curva C_T, sugerindo um desempenho de escalabilidade próximo ao linear das plataformas mestre-escravo quando se aumenta o poder da plataforma somente pelo *scale-up*.

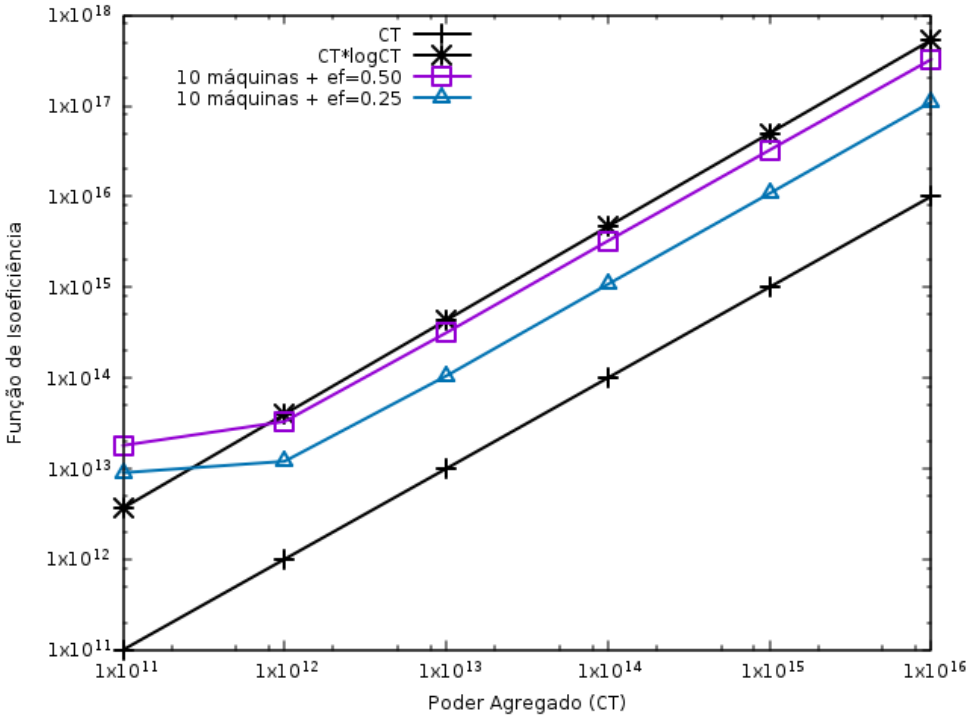


Figura 4.5: Função de isoeficiência para plataformas mestre-escravo homogênea, variando o poder dos nós.

4.7 Quarto Experimento: Plataforma de Nuvem

4.7.1 Objetivo

O objetivo deste experimento é analisar a escalabilidade de aplicações BoT executando, especificamente, em plataformas de computação em nuvem. Nos experimentos anteriores, cada nó pode crescer infinitamente (modelo teórico), fato que poderia comprometer a análise em ambientes cujos nós possuem limitações acerca da sua capacidade computacional máxima, como é o caso da nuvem. Portanto, pretende-se verificar a validade das análises de escalabilidade em um contexto de plataforma mais realista.

4.7.2 Configuração

Antes da execução dos experimentos, foi feita a calibragem do simulador para reproduzir as configurações de um sistema real. Esse processo compreende a configuração dos recursos da plataforma (capacidade computacional das máquinas, largura de banda da rede) e parâmetros da aplicação (quantidade de tarefas, tamanho médio das tarefas, tamanho dos arquivos de entrada). Esses valores foram definidos com base na plataforma e aplicação utilizados em (CASTRO et al., 2017). Trata-se de uma aplicação de bioinformática, que realiza alinhamento de sequências de genomas. Essa aplicação pode ser caracterizada como *Bag-of-Tasks*, pois cada sequência é processada de forma independente. Neste caso, cada sequência compreende uma tarefa da aplicação.

No trabalho em questão, a ferramenta foi executada nas plataformas de nuvem da Google e Microsoft Azure. Para calibragem do simulador, foram utilizados os dados baseados na execução da ferramenta na plataforma da Microsoft Azure, pois foram utilizadas instâncias de uma mesma zona de disponibilidade, enquanto que na plataforma da Google, as instâncias foram alocadas em diferentes zonas geograficamente distribuídas. Nos experimentos de Castro et al. (2017) realizados na Azure, a quantidade de máquinas virtuais variou em 1, 3, 7, 15, 31 e 63. As VMs alocadas são do tipo A3 (4 vCPUs e 7 GB de memória). A aplicação consiste em 86968 sequências que são processadas de acordo com uma base de dados de metagenoma. Desta forma, a aplicação possui 86968 tarefas que compartilham um arquivo de entrada de 805 MB (base de dados de metagenoma).

As instâncias A3 da Azure apresentam largura de banda de rede de 1 Gbps[6]. Entretanto esse é o valor limite esperado. Para medir a real taxa de transferência da rede foi uti-

[6]https://docs.microsoft.com/pt-br/azure/virtual-machines/linux/sizes-previous-gen

lizado o NTTTCP[7], recomendado pela própria Microsoft[8]. As medições foram feitas com duas máquinas virtuais (cliente e servidor) e taxa de transferência aferida foi de 909 Mbps. Para medir a capacidade computacional de uma instância A3 em termos de FLOP/s (unidade de medida utiliza pelo Simgrid) foi utilizado o HPL[9], indicando média aproximada de 4 GFlops por vCPU. Esses valores foram utilizados para calibrar a plataforma do simulador. Para determinar o tempo médio das tarefas, foi feita a divisão do tempo de processamento sequencial (real) da aplicação pela quantidade de tarefas (sequências). Após isso, o processo de calibragem ocorre de forma iterativa e consiste em ajustar os parâmetros da plataforma e da aplicação até o simulador conseguir imitar o sistema real com precisão aceitável.

Tempo de Processamento (segundos)				
Instâncias	**vCPUs**	**Tempo Simulado**	**Tempo Real**	**Variação** (%)
1	4	170837	143228,95	16,16%
3	12	56965,6	47031,62	17,44%
7	28	24458,3	24850,51	-1,60%
15	60	11502,9	11692,45	-1,65%
31	124	5749,58	6041,64	-5,08%
63	252	3198,02	3138,64	1,86%

Tabela 4.4: Precisão alcançada do tempo simulado em comparação com o tempo real de processamento da aplicação.

A Tabela 4.4 e Figura 4.6 mostram a precisão alcançada no processo de calibragem. O tempo médio para processamento de uma tarefa é de 7 segundos em uma vCPU com capacidade de 4 GFlops. Desta forma, o tamanho médio das tarefas é $2,95 \times 10^{10}$ FLOPs. A largura de banda da rede foi fixada em 900 Mbps no *link* de conexão entre o nó mestre e os nós escravos. Vale destacar que todos os nós da plataforma (mestre e escravos) são instâncias virtuais dentro de uma mesma zona de disponibilidade. Com essa configuração, o arquivo de entrada de 805 MB leva cerca de 7 segundos para ser transferido do mestre para um nó escravo. Isso representa um CCR de 1, ou seja, o tempo de comunicação é equivalente ao tempo de computação da tarefa. De maneira semelhante aos experimentos anteriores, o arquivo de entrada é compartilhado por todas as tarefas (IFA máximo), sendo transferido para nó escravo apenas uma vez. Neste caso, o número de transmissões depende da quantidade de nós escravos.

O objetivo da calibragem é obter as configurações da plataforma de computação em nuvem e parâmetros de uma aplicação real, de modo a realizar simulações com resultados que se aproximam da realidade. Apesar de apresentar uma discrepância na execução da aplicação em poucos nós, o tempo simulado tende a se aproximar do tempo real de processamento total da

[7]NTTTCP: https://github.com/Microsoft/ntttcp-for-linux

[8]https://docs.microsoft.com/pt-br/azure/virtual-network/virtual-network-bandwidth-testing

[9]HPL: http://www.netlib.org/benchmark/hpl/

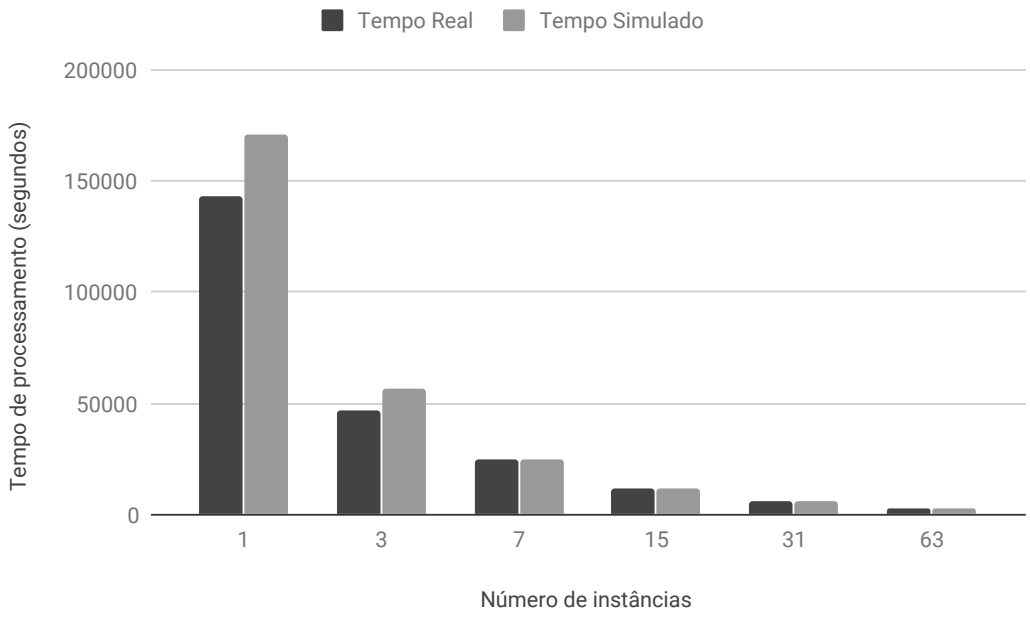


Figura 4.6: Comparativo entre tempo de processamento real e tempo simulado em cada quantidade de instâncias.

aplicação na presença de uma quantidade maior de nós. Isso mostra que os valores aferidos estão próximos do comportamento de um sistema real, o que confere maior credibilidade aos resultados dos experimentos executados.

Instâncias Simuladas		
Tipo	**vCPUs**	**GFlops**
T1	1	4
T2	2	8
T3	4	16
T4	8	32
T5	16	64
T6	32	128
T7	64	256
T8	96	384

Tabela 4.5: Instâncias simuladas nos experimentos de análise de escalabilidade.

De modo geral, os procedimentos dos experimentos desta seção são os mesmos adotados na seção 4.4 deste capítulo. Entretanto, a configuração da simulação segue os valores obtidos na calibragem realizada acima. A plataforma mestre-escravo é composta por instâncias tipicamente comuns em ambientes de computação em nuvem. A Tabela 4.5 apresenta os tipos de instâncias simuladas nos experimentos. Para cada vCPU são atribuídos 4 GFlops de poder computacional à instância e todas as máquinas estão interconectadas por um *link* de 900 Mbps. A aplicação é composta por tarefas homogêneas de tamanho $2,95 \times 10^{10}$ FLOPs, que comparti-

Quantidade de Instâncias					
Potência Agregada	**Nível de Heterogeneidade**				
	0%	**25%**	**50%**	**75%**	**99%**
4 GFLOP/s	1	1	1	1	1
40 GFLOP/s	10	9	7	6	3
400 GFLOP/s	100	78	53	29	5
4 TFLOP/s	1000	756	507	261	25

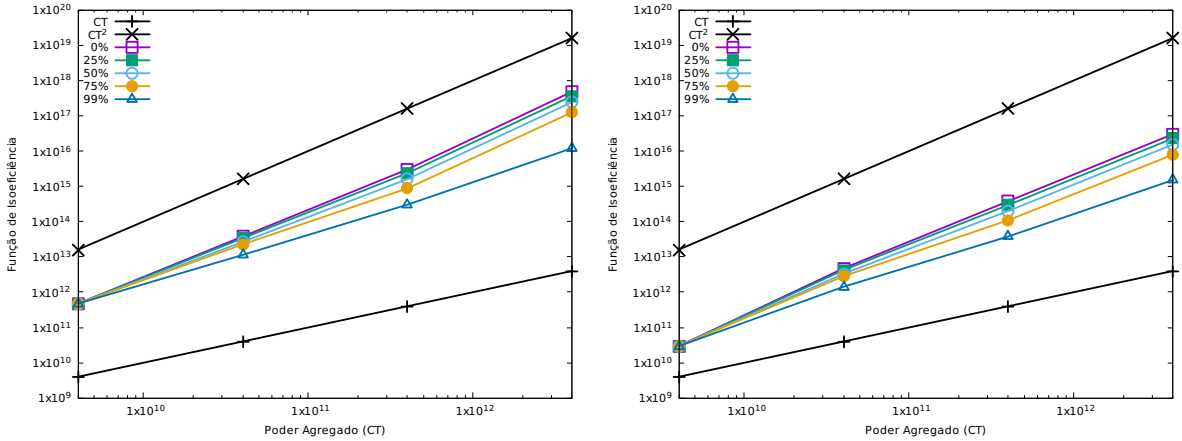
Tabela 4.6: Número de instâncias em cada plataforma simulada.

lham um arquivo de entrada de 805 MB. Nesse experimento, a capacidade da plataforma variou de 4 GFLOP/s até 4 TFLOP/s. As plataformas homogêneas são geradas com 1, 10, 100 e 1000 instâncias com capacidade de 4 GFLOP/s. Neste caso, a instância do tipo T1 (1 vCPU) é a máquina básica de referência. As plataformas heterogêneas são geradas com diferentes níveis de heterogeneidade (25%, 50%, 75% e 99%). O procedimento consiste em: (*i*) seja C_T o poder total da plataforma a ser gerada, e H a porcentagem de C_T destinada para as máquinas heterogêneas; (*ii*) n máquinas grandes (as maiores possíveis) são adicionadas à plataforma até que o poder agregado dos nós heterogêneos seja $H \times C_T$; e (*iii*) o poder computacional restante é distribuído com a adição de nós homogêneos (instâncias T1) até que o poder agregado desses nós seja $C_T \times (1-H)$. A Tabela 4.6 mostra a quantidade total de instâncias em cada plataforma. A função de isoeficiência foi calculada, considerando o nível de eficiência desejado de 0,9.

4.7.3 Resultados e Conclusões

Os resultados são ilustrados na Figura 4.7 e Tabela 4.7. Em cada experimento foi variado o CCR em 1 (Figura 4.7 (a)) e 0,1 (Figura 4.7 (b)), tomando por base a máquina de referência. Como se pode observar em todos os casos avaliados, as plataformas com maiores níveis de heterogeneidade são mais escaláveis, pois apresentam menores valores para a função de isoeficiência. Isso mostra que os resultados obtidos nos experimentos da seção 4.4 são válidos também para plataformas com características mais realistas, como é o caso do ambiente de computação em nuvem.

As aplicações BoT podem ser classificadas em *compute-intensive* e *data-intensive*. Em aplicações *compute-intensive*, o tempo de transferência dos dados de entrada é pequeno (ou até mesmo desprezável) em relação ao tempo de processamento da carga de trabalho da tarefa. Já nas aplicações *data-intensive*, a transmissão dos dados de entrada possuem grande fator de influência no tempo final de execução da tarefa (LEE; ZOMAYA, 2007). Ambos os tipos de aplicações foram testadas com a variação do CCR em 1 para aplicações *data-intensive* e 0,1 para aplicações *compute-intensive*. Na comparação entre as plataformas com diferentes



(a) CCR de 1 (b) CCR de 0,1

Figura 4.7: Função de isoeficiência para execução de tarefas homogêneas em plataformas mestre-escravo na nuvem, mantendo a eficiência acima de 90% e variando o CCR.

Nível de Heterogeneidade	Poder Agregado da Plataforma							
	4 GFLOP/s		40 GFLOP/s		400 GFLOP/s		4 TFLOP/s	
	CCR 1	CCR 0,1	CCR 1	CCR 0,1	CCR 1	CCR 0,1	CCR 1	CCR 0,1
0%	4,72E+11	2,95E+10	3,78E+13	4,72E+12	3,02E+15	3,78E+14	4,83E+17	3,02E+16
25%	4,72E+11	2,95E+10	3,40E+13	4,25E+12	2,36E+15	2,95E+14	3,65E+17	2,28E+16
50%	4,72E+11	2,95E+10	2,64E+13	3,30E+12	1,60E+15	2,00E+14	2,45E+17	1,53E+16
75%	4,72E+11	2,95E+10	2,27E+13	2,83E+12	8,76E+14	1,10E+14	1,26E+17	7,88E+15
99%	4,72E+11	2,95E+10	1,13E+13	1,42E+12	3,02E+14	3,78E+13	1,21E+16	1,51E+15

Tabela 4.7: Valor da função de isoeficiência para execução de tarefas homogêneas em plataformas mestre-escravo na nuvem, mantendo a eficiência acima de 90% e variando o CCR.

níveis de heterogeneidade, o comportamento da função de isoeficiência para cada plataforma simulada se mantém, independentemente do CCR. Os resultados também evidenciam o impacto da transferência dos dados de arquivos de entrada na escalabilidade da aplicação BoT. A redução da quantidade de dados transmitidos (ou do tempo de transmissão) melhora significativamente o desempenho, e consequentemente, a escalabilidade da aplicação.

4.8 Discussão

Um fator que afeta a eficiência, e consequentemente, limita a escalabilidade dos sistemas paralelos é o *overhead* de comunicação. O tempo de execução da aplicação é constituído pelo tempo efetivo para realizar as computações, somado ao tempo utilizado para comunicação e sincronização. No caso das aplicações BoT, o *overhead* decorre dos custos para coordenação e sincronização do sistema, bem como das transmissões de arquivos de entrada para os nós escravos e retorno dos resultados produzidos. A Figura 4.8 ilustra essa situação. Devido ao gargalo do mestre, o escravo 4, por exemplo, só começa a receber a tarefa no instante de tempo

3 e inicia efetivamente a execução no instante de tempo 4. Com isso, é possível inferir que o *overhead* é proporcional à quantidade de escravos. Portanto, quanto maior a quantidade de máquinas escravas, maior também será o *overhead* de comunicação. Isso reduz a eficiência e a escalabilidade do sistema. Desta forma, a simples adição de mais máquinas (escalabilidade horizontal), nem sempre acarreta em melhorias no desempenho de uma determinada aplicação.

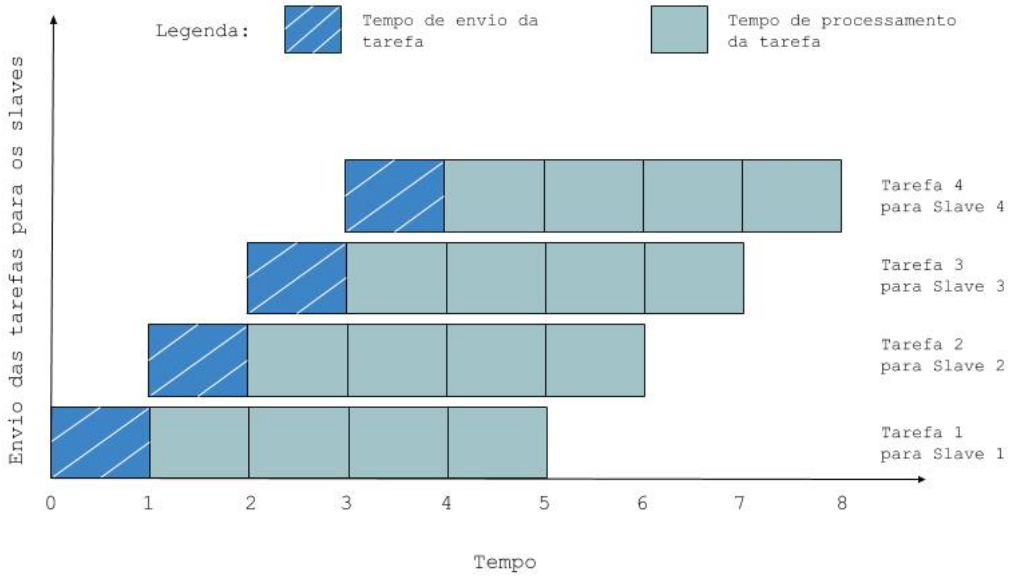


Figura 4.8: Ilustração do *overhead* de comunicação das aplicações BoT.

As plataformas mestre-escravo homogêneas são bastante representativas do que ocorre na prática, quando usuários de nuvem fazem uso da elasticidade de recursos adicionando ou reduzindo nós à plataforma (escalabilidade horizontal). Provavelmente pela simplicidade ou facilidade de implementação, essa tem sido a forma típica e mais usual de se implementar a elasticidade de recursos, sendo que vários provedores oferecem serviços que automatizam o aumento ou redução do número de instâncias alocadas pelo cliente (em função da carga, do orçamento, etc). Contudo, a adição de nós aumenta também o *overhead* para transmissão dos dados e de coordenação da plataforma. Por outro lado, permitir que a plataforma possa ter recursos heterogêneos melhora sua escalabilidade, conforme demonstram os experimentos. Ganhos significativos foram observados mesmo em cenários com cargas de trabalho típicas de plataformas reais, conforme mostrado no segundo experimento.

Nas plataformas heterogêneas, é possível aumentar o poder computacional da plataforma substituindo nós existentes por nós de maior capacidade (escalabilidade vertical), desde que existam instâncias maiores disponíveis. Isto permite o crescimento da plataforma sem a adição de *overhead*, o que reduz o tempo de execução da aplicação e melhora a eficiência do sistema. Em plataformas heterogêneas, não é necessário que todos os nós sejam melhorados, mas apenas

um ou alguns nós específicos. Em termos práticos, quando se executam aplicações BoT em uma plataforma como na nuvem, por exemplo, é quase sempre mais vantajoso substituir instâncias pequenas por uma única instância com capacidade equivalente. A escolha por instâncias de maior capacidade pode ser feita no momento inicial da execução da aplicação, ou posteriormente, à medida que o provedor disponibilize instâncias de maior capacidade computacional. De modo geral, sempre que houver disponibilidade, será mais vantajoso aumentar o poder da plataforma por meio do *scale-up*, e, quando este não for mais possível, aplicar então o *scale-out*. A presente análise não leva em conta o aumento dos custos das instâncias ou mesmo o orçamento para executar as aplicações.

Capítulo 5

PROVISIONAMENTO DE RECURSOS

As aplicações *Bag-of-Tasks* são muito comuns em sistemas de larga escala. Neste contexto, a computação em nuvem oferece uma solução econômica para a execução de aplicações BoT, em que um usuário é livre para escolher o tipo e a quantidade de recursos necessários para sua aplicação (THAI; VARGHESE; BARKER, 2015). Em termos de serviços de computação, os provedores de nuvem oferecem uma grande variedade de opções de instâncias aos clientes. Cada tipo de instância possui capacidade computacional (quantidade de vCPUs e memória) e preços distintos. Provedores como *Amazon Web Services*, *Microsoft Azure* e *Google Cloud Platform* definem a tarifação das instâncias conforme o uso por segundo, sendo que o tempo mínimo de utilização é de 60 segundos. Um dos principais desafios no ambiente de computação em nuvem é o provisionamento de recursos. Devido a natureza dinâmica da nuvem e o rápido crescimento da demanda de recursos, a alocação deve ocorrer de acordo com a demanda da carga de trabalho (AMIRI; MOHAMMAD-KHANLI, 2017). A escolha de um conjunto adequado de máquinas virtuais para a execução de uma determinada aplicação, pode melhorar o desempenho e reduzir os custos.

As análises feitas no capítulo 4 mostram que plataformas heterogêneas são mais escaláveis que plataformas homogêneas, pois permitem a redução da quantidade de instâncias utilizadas, mantendo a capacidade computacional agregada. Isso reduz o *overhead* decorrente da transferência de dados da aplicação. Desta forma, o provisionamento de recursos para a execução de aplicações BoT na nuvem, deve considerar a heterogeneidade da plataforma na seleção de um conjunto de instâncias. Pensando nisso, é proposto neste capítulo, um algoritmo de provisionamento de recursos para a execução de aplicações BoT em plataformas de computação em nuvem. O objetivo é mostrar a aplicabilidade das conclusões obtidas na análise de escalabilidade realizada neste trabalho.

5.1 MinER - *Minimum Equivalent Resources*

Esta seção apresenta o algoritmo de provisionamento proposto, intitulado MinER (*Minimum Equivalent Resources*), pois busca selecionar um conjunto mínimo de recursos equivalentes. O algoritmo não considera o preço das instâncias e visa minimizar o número de máquinas, mantendo o poder computacional da plataforma. O algoritmo recebe como entrada uma lista de instâncias α, contendo todas as instâncias disponíveis no provedor. Cada instância α_i é representada pela tupla (C_i, M_i), onde C_i é o poder computacional da instância em termos de quantidade de vCPUs e M_i é o preço pela utilização da instância (por hora ou segundo). Nos provedores de nuvem, a capacidade computacional de uma instância é definida pela quantidade de vCPUs. Além das instâncias disponíveis na nuvem, o algoritmo recebe como parâmetro a quantidade de vCPUs P desejada para a execução da aplicação. O número de vCPUs define o tamanho da plataforma. A partir do tamanho de plataforma desejado, o algoritmo seleciona na lista α as maiores instâncias possíveis até alcançar a quantidade de vCPUs P esperada. O objetivo é reduzir a quantidade de VMs, mantendo o poder computacional agregado do conjunto P inicial. Com a diminuição da quantidade total de VMs a serem provisionadas, pretende-se reduzir o número de envios dos arquivos compartilhados entre as tarefas, uma vez que, um arquivo precisa ser enviado apenas uma vez para cada VM. Com isso, espera-se reduzir o tempo total de execução da aplicação.

Algoritmo 1: MinER: seleciona um conjunto mínimo de recursos equivalentes a P

Entrada: Lista de instâncias disponíveis na nuvem (α), quantidade de vCPUs desejadas (P)
Saída: Lista de instâncias selecionadas (S)

```
1  início
2      S ← []
3      ordena a lista de instâncias α (ordem crescente)
4      enquanto tamanho de α > 0 e P > 0 faça
5          α_maior ← obtem e remove a maior instância de α
6          n ← número de vCPUs de α_maior
7          se P > n então
8              insere α_maior em S
9              P ← P−n
10         fim
11     fim
12     retorna S
13 fim
```

Inicialmente a lista de instâncias α disponíveis na nuvem é ordenada de forma crescente. Enquanto existirem instâncias disponíveis na nuvem e vCPUs P ainda não alocadas, o algoritmo

remove a maior instância α_{maior} da lista α e verifica se ela pode substituir algumas vCPUs de P. A substituição é possível quando a quantidade de vCPUs P é superior ou igual a quantidade de vCPUs da instância α_{maior}. Neste caso, verifica-se a quantidade de vCPUs a serem alocadas por α_{maior}, subtraindo esse valor de P. Se a substituição é realizada, então α_{maior} é inserida na lista S de instâncias selecionadas. Esse processo é realizado para algumas ou todas as instâncias da lista α. As instâncias selecionadas na lista S são então retornadas como resultado do algoritmo.

5.2 Avaliação e Resultados

O algoritmo apresentado neste capítulo realiza o provisionamento de recursos, buscando reduzir a quantidade de nós da plataforma. Com isso, pode resultar em um conjunto heterogêneo de máquinas. Com a redução do número de nós, há a diminuição do tráfego de dados decorrente da transferência de arquivos do nó mestre para os nós escravos. Para avaliar a solução de provisionamento de recursos, foram realizados experimentos utilizando o Simgrid. Nesses experimentos, a aplicação apresentada na sessão 4.7 do capítulo 4, foi executada em diferentes conjuntos de instâncias, mantendo o mesmo poder computacional agregado. A Tabela 5.1 apresenta os tipos de instâncias disponíveis na plataforma de nuvem simulada, baseados na Microsoft Azure[1].

Instâncias Disponíveis			
Tipo de Instância	**vCPUs**	**RAM**	**Preço/hora**
F1	1	2 GB	R$ 0,18
F2s v2	2	4 GB	R$ 0,32
F4s v2	4	8 GB	R$ 0,63
F8s v2	8	16 GB	R$ 1,26
F16s v2	16	32 GB	R$ 2,53
F32s v2	32	64 GB	R$ 5,05
F64s v2	64	128 GB	R$ 10,10
F72s v2	72	144 GB	R$ 11,37

Tabela 5.1: Tipos de instâncias utilizadas na avaliação do algoritmo de provisionamento.

Quatro conjuntos de instâncias foram testados. Todos os conjuntos possuem quantidade agregada de 252 vCPUs. O primeiro conjunto é composto por instâncias do tipo F1 (1 vCPU), o tipo mais básico disponível. O segundo conjunto é constituído de instâncias de tipo F2s v2 (2 vCPUs). O terceiro conjunto é formado por instâncias do tipo F4s v2 (4 vCPUs), e é equivalente a maior plataforma utilizada nos experimentos de Castro et al. (2017) mencionados no capítulo 4. Esses conjuntos de instâncias são homogêneos. O quarto conjunto é gerado pelo

[1] https://azure.microsoft.com/pt-br/pricing/details/virtual-machines/linux/

algoritmo MinER proposto neste capítulo. Considerando as instâncias disponíveis na Tabela 5.1 e a capacidade de 252 vCPUs da plataforma, o MinER selecionou um conjunto heterogêneo de instâncias composto por 3 instâncias do tipo F72s v2 (72 vCPUs), 1 instâncias do tipo F32s v2 (32 vCPUs) e 1 instância do tipo F4s v2 (4 vCPUs).

Tempo e Custo de Processamento da Aplicação			
Plataforma	**Número de Instâncias**	**Tempo**	**Custo**
1 vCPU	252	4659,97	R$ 60,25
2 vCPU	126	3685,33	R$ 40,73
4 vCPU	63	3198,02	R$ 35,35
MinER	5	2753,27	R$ 30,43

Tabela 5.2: Tempo (segundos) e custo (em real) da execução da aplicação em cada conjunto de instâncias.

Os resultados são ilustrados na Tabela 5.2, Figura 5.1 (tempo de execução) e Figura 5.2 (custo de locação das instâncias). O conjunto de instâncias provisionadas pelo MinER proporciona o melhor desempenho para a aplicação entre todos os conjuntos testados. Além da redução do tempo total de processamento, a solução proposta apresenta o menor custo financeiro, considerando os preços de cada instância apresentada na Tabela 5.1. Vale lembrar que todos os conjuntos possuem a mesma capacidade computacional agregada. A diferença está na quantidade de instâncias. Nos resultados, verifica-se que a execução da aplicação BoT em conjuntos com menos instâncias apresenta melhor desempenho e menor custo.

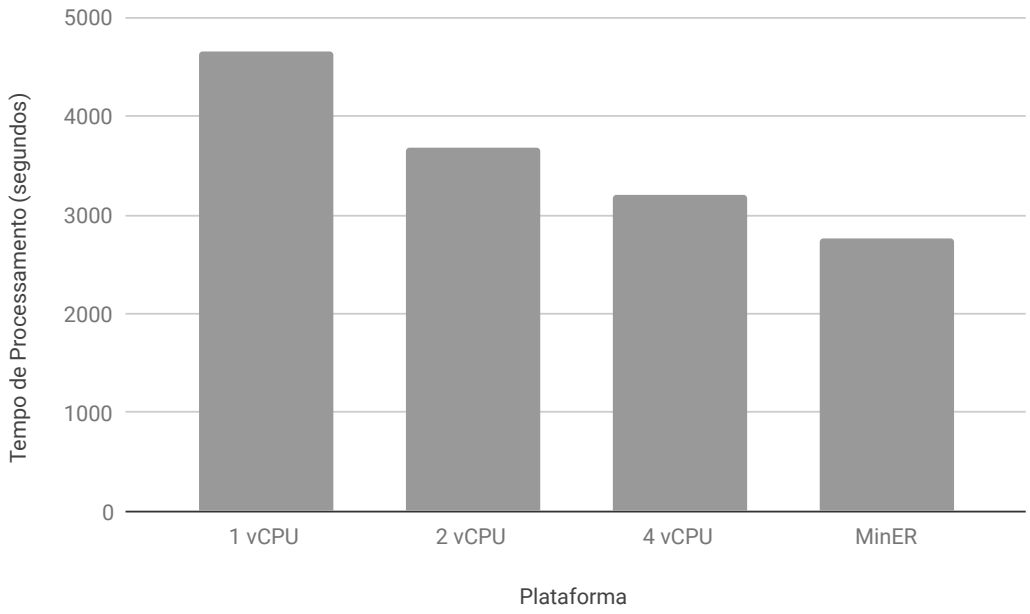


Figura 5.1: Tempo de processamento da aplicação em cada conjunto de instâncias.

Esses resultados corroboram com as conclusões apresentadas no capítulo 4, onde é mos-

trado que a escalabilidade vertical é mais vantajosa do que a escalabilidade horizontal, pois permite o crescimento da plataforma sem a adição de *overhead* de comunicação. Ganhos significativos são observados na execução de aplicações BoT em uma plataforma de nuvem, quando instâncias pequenas são substituídas por uma única instância com capacidade equivalente. Estas observações são válidas para o processamento de aplicações BoT com compartilhamento de arquivos.

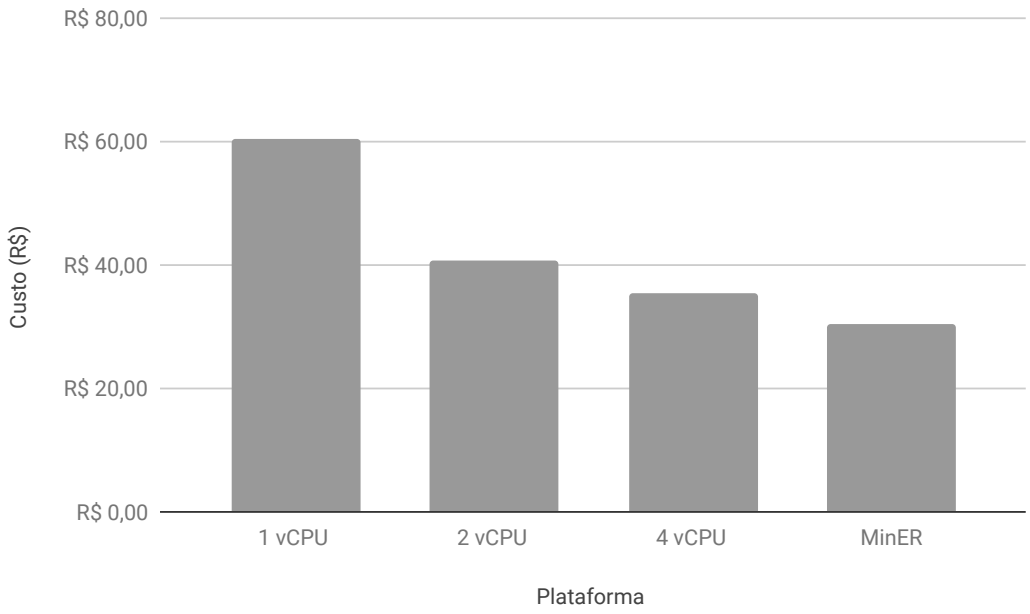


Figura 5.2: Custo do processamento da aplicação em cada conjunto de instâncias.

Capítulo 6

Conclusão

O presente trabalho apresenta uma análise da escalabilidade de aplicações BoT executando em plataformas mestre-escravo heterogêneas. Tais plataformas são implementadas por grandes sistemas de computação distribuída, incluindo os *clusters*, grades computacionais e nuvem. Foi demonstrado anteriormente que plataformas mestre-escravo homogêneas apresentam escalabilidade bastante limitada (com a função de isoeficiência crescendo proporcional a $\Omega(P^2)$ (SILVA; SENGER, 2009), enquanto que as plataformas hierárquicas homogêneas (que são uma extensão do mestre-escravo) são muito mais escaláveis, com isoeficiência crescendo proporcional a $\Omega(P \cdot \log P)$ (SILVA; SENGER, 2011). Os resultados apresentados no presente trabalho demonstram, de forma experimental, que a heterogeneidade pode tornar a plataforma distribuída ainda mais escalável que as plataformas homogêneas citadas.

Como é verificado no primeiro experimento, plataformas distribuídas heterogêneas são mais escaláveis do que plataformas homogêneas de mesmo poder computacional agregado. Essa conclusão foi demonstrada também em ambientes com cargas de trabalho mais realistas (segundo experimento) e plataformas de computação em nuvem com configurações típicas (quarto experimento). Os experimentos sugerem, ainda, que aumentar a capacidade do sistema por meio da escalabilidade vertical pode (sob certas condições) proporcionar escalabilidade próxima do linear. Foi mostrado que as transmissões de dados das aplicações BoT representam um gargalho nas interações entre o nó mestre e os nós escravos, gerando um *overhead* na execução da aplicação. Nesse sentido, a escalabilidade vertical possibilita o crescimento da plataforma sem adição (e até mesmo com redução) do *overhead* de comunicação, o que não é possível com o método de escalabilidade horizontal. Verificou-se que na prática, substituir um ou mais nós por um único de potência equivalente em geral traz melhoria de desempenho ao sistema. As conclusões apresentadas podem ser aplicadas em soluções de provisionamento de recursos para a execução de aplicações BoT. Foi mostrado que soluções de provisionamento que exploram a

heterogeneidade da plataforma distribuída e escalam verticalmente, melhoram o desempenho da aplicação e consequentemente podem reduzir os custos envolvidos.

Como trabalhos futuros, pretende-se elaborar uma prova formal para o limite assintótico da escalabilidade de aplicações BoT executando em plataformas heterogêneas. Além disso, pretende-se avaliar o impacto da variabilidade das cargas impostas por diferentes usuários que concorrem pelo uso de uma infraestrutura de nuvem compartilhada, de modo a propor soluções de escalonamento. Outra proposta de trabalho futuro é a ampliação do estudo sobre provisionamento de recursos, explorando melhor a correlação entre heterogeneidade e desempenho, para o desenvolvimento de novas heurísticas de provisionamento. Nesse contexto também, diferentes políticas de preços dos provedores, e plataformas de nuvem federada serão considerados.

REFERÊNCIAS

ABDI, S.; POURKARIMI, L.; AHMADI, M.; ZARGARI, F. Cost minimization for deadline-constrained bag-of-tasks applications in federated hybrid clouds. *Future Generation Computer Systems*, v. 71, p. 113 – 128, 2017. ISSN 0167-739X. Disponível em: ⟨http://www.sciencedirect.com/science/article/pii/S0167739X17301735⟩.

AL-DHURAIBI, Y.; PARAISO, F.; DJARALLAH, N.; MERLE, P. Elasticity in Cloud Computing: State of the Art and Research Challenges. *IEEE Transactions on Services Computing (TSC)*, v. 11, n. 2, p. 430–447, mar. 2018. Disponível em: ⟨https://hal.inria.fr/hal-01529654⟩.

AMIRI, M.; MOHAMMAD-KHANLI, L. Survey on prediction models of applications for resources provisioning in cloud. *Journal of Network and Computer Applications*, v. 82, p. 93 – 113, 2017. ISSN 1084-8045. Disponível em: ⟨http://www.sciencedirect.com/science/article/pii/S1084804517300231⟩.

AMRANI, C. E.; FILALI, K. B.; AHMED, K. B.; DIALLO, A. T.; TELOLAHY, S.; ELGHAZAWI, T. A compartive study of cloud computing middleware. In: *Proceedings of the 2012 12th IEEE/ACM International Symposium on Cluster, Cloud and Grid Computing (Ccgrid 2012)*. Washington, DC, USA: IEEE Computer Society, 2012. (CCGRID '12), p. 690–693. ISBN 978-0-7695-4691-9. Disponível em: ⟨https://doi.org/10.1109/CCGrid.2012.129⟩.

ARMBRUST, M.; FOX, A.; GRIFFITH, R.; JOSEPH, A. D.; KATZ, R.; KONWINSKI, A.; LEE, G.; PATTERSON, D.; RABKIN, A.; STOICA, I.; ZAHARIA, M. A view of cloud computing. *Commun. ACM*, ACM, New York, NY, USA, v. 53, n. 4, p. 50–58, abr. 2010. ISSN 0001-0782. Disponível em: ⟨http://doi.acm.org/10.1145/1721654.1721672⟩.

ASHWINI, J. P.; DIVYA, C.; SANJAY, H. A. Efficient resource selection framework to enable cloud for hpc applications. In: *2013 4th International Conference on Computer and Communication Technology (ICCCT)*. [S.l.: s.n.], 2013. p. 34–38.

BEAUMONT, O.; CARTER, L.; FERRANTE, J.; LEGRAND, A.; MARCHAL, L.; ROBERT, Y. Centralized versus distributed schedulers for bag-of-tasks applications. *IEEE Transactions on Parallel and Distributed Systems*, IEEE, v. 19, n. 5, p. 698–709, 2008.

BEAUMONT, O.; LEGRAND, A.; ROBERT, Y. Scheduling divisible workloads on heterogeneous platforms. *Parallel Computing*, Elsevier, v. 29, n. 9, p. 1121–1152, 2003.

BERMAN, F.; WOLSKI, R.; CASANOVA, H.; CIRNE, W.; DAIL, H.; FAERMAN, M.; FIGUEIRA, S.; HAYES, J.; OBERTELLI, G.; SCHOPF, J. et al. Adaptive computing on the grid using apples. *IEEE Transactions on Parallel and Distributed Systems*, IEEE, v. 14, n. 4, p. 369–382, 2003.

BOSQUE, J. L.; PEREZ, L. P. Theoretical scalability analysis for heterogeneous clusters. In: *IEEE International Symposium on Cluster Computing and the Grid, 2004. CCGrid 2004.* [S.l.: s.n.], 2004. p. 285–292.

BOSQUE, J. L.; ROBLES, O. D.; TOHARIA, P.; PASTOR, L. Evaluating scalability in heterogeneous systems. *The Journal of Supercomputing*, v. 58, n. 3, p. 367–375, Dec 2011. ISSN 1573-0484. Disponível em: ⟨https://doi.org/10.1007/s11227-011-0593-5⟩.

BUYYA, R.; YEO, C. S.; VENUGOPAL, S.; BROBERG, J.; BRANDIC, I. Cloud computing and emerging it platforms: Vision, hype, and reality for delivering computing as the 5th utility. *Future Generation Computer Systems*, v. 25, n. 6, p. 599 – 616, 2009. ISSN 0167-739X. Disponível em: ⟨http://www.sciencedirect.com/science/article/pii/S0167739X08001957⟩.

CALHEIROS, R. N.; RANJAN, R.; BELOGLAZOV, A.; ROSE, C. A. F. D.; BUYYA, R. Cloudsim: A toolkit for modeling and simulation of cloud computing environments and evaluation of resource provisioning algorithms. *Softw. Pract. Exper.*, John Wiley & Sons, Inc., New York, NY, USA, v. 41, n. 1, p. 23–50, jan. 2011. ISSN 0038-0644. Disponível em: ⟨http://dx.doi.org/10.1002/spe.995⟩.

CARVALHO, M.; BRASILEIRO, F. A user-based model of grid computing workloads. In: *2012 ACM/IEEE 13th Intl. Conf. on Grid Computing*. [S.l.: s.n.], 2012. p. 40–48. ISSN 2152-1093.

CASANOVA, H.; GIERSCH, A.; LEGRAND, A.; QUINSON, M.; SUTER, F. Versatile, Scalable, and Accurate Simulation of Distributed Applications and Platforms. *Journal of Parallel and Distributed Computing*, Elsevier, v. 74, n. 10, p. 2899–2917, jun. 2014. Disponível em: ⟨https://hal.inria.fr/hal-01017319⟩.

CASANOVA, H.; LEGRAND, A.; ZAGORODNOV, D.; BERMAN, F. Heuristics for scheduling parameter sweep applications in grid environments. In: IEEE. *9th IEEE Heterogeneous Computing Workshop.(HCW 2000)*. [S.l.], 2000. p. 349–363.

CASTRO, M. R. de; TOSTES, C. d. S.; DÁVILA, A. M. R.; SENGER, H.; SILVA, F. A. B. da. Sparkblast: scalable blast processing using in-memory operations. *BMC Bioinformatics*, v. 18, n. 1, p. 318, Jun 2017. ISSN 1471-2105. Disponível em: ⟨https://doi.org/10.1186/s12859-017-1723-8⟩.

CHOU, D. C. Cloud computing: A value creation model. *Computer Standards Interfaces*, v. 38, n. Supplement C, p. 72 – 77, 2015. ISSN 0920-5489. Disponível em: ⟨http://www.sciencedirect.com/science/article/pii/S0920548914000981⟩.

CIRNE, W.; PARANHOS, D.; COSTA, L.; SANTOS-NETO, E.; BRASILEIRO, F.; SAUVE, J.; SILVA, F. A.; BARROS, C. O.; SILVEIRA, C. Running bag-of-tasks applications on computational grids: The mygrid approach. In: IEEE. *International Conference on Parallel Processing (ICPP)*. [S.l.], 2003. p. 407–416.

CLOUDSTACK. *Apache CloudStack: Open Source Cloud Computing*. 2018. Acesso em: 23 mai. 2018. Disponível em: ⟨https://cloudstack.apache.org/⟩.

CORDEIRO, T. D.; DAMALIO, D. B.; PEREIRA, N. C. V. N.; ENDO, P. T.; PALHARES, A. V. de A.; GONçALVES, G. E.; SADOK, D. F. H.; KELNER, J.; MELANDER, B.; SOUZA,

V.; MåNGS, J. E. Open source cloud computing platforms. In: *2010 Ninth International Conference on Grid and Cloud Computing*. [S.l.: s.n.], 2010. p. 366–371. ISSN 2160-4908.

GIERSCH, A.; ROBERT, Y.; VIVIEN, F. Scheduling tasks sharing files on heterogeneous master-slave platforms. *Journal of Systems Architecture*, Elsevier, v. 52, n. 2, p. 88–104, fev. 2006. Disponível em: ⟨https://hal.archives-ouvertes.fr/hal-00515330⟩.

GRACE, M. R. K.; PRIYA, S. S.; SURYA, S. A survey on grid simulators. *Int. J. Comput. Sci. Inf. Technol. Secur*, v. 2, n. 6, p. 1224–1230, 2012.

GRAMA, A. Y.; GUPTA, A.; KUMAR, V. Isoefficiency: measuring the scalability of parallel algorithms and architectures. *IEEE Parallel Distributed Technology: Systems Applications*, v. 1, n. 3, p. 12–21, Aug 1993. ISSN 1063-6552.

GUPTA, P.; SEETHARAMAN, A.; RAJ, J. R. The usage and adoption of cloud computing by small and medium businesses. *International Journal of Information Management*, v. 33, n. 5, p. 861 – 874, 2013. ISSN 0268-4012. Disponível em: ⟨http://www.sciencedirect.com/science/article/pii/S026840121300087X⟩.

GUTIERREZ-GARCIA, J. O.; SIM, K. M. Ga-based cloud resource estimation for agent-based execution of bag-of-tasks applications. *Information Systems Frontiers*, v. 14, n. 4, p. 925–951, Sep 2012. ISSN 1572-9419. Disponível em: ⟨https://doi.org/10.1007/s10796-011-9327-8⟩.

HASHEM, I. A. T.; YAQOOB, I.; ANUAR, N. B.; MOKHTAR, S.; GANI, A.; KHAN, S. U. The rise of "big data" on cloud computing: Review and open research issues. *Information Systems*, v. 47, n. Supplement C, p. 98 – 115, 2015. ISSN 0306-4379. Disponível em: ⟨http://www.sciencedirect.com/science/article/pii/S0306437914001288⟩.

HWANG, E.; KIM, K. H. Minimizing cost of virtual machines for deadline-constrained mapreduce applications in the cloud. In: *2012 ACM/IEEE 13th International Conference on Grid Computing*. [S.l.: s.n.], 2012. p. 130–138. ISSN 2152-1085.

IOSUP, A.; SONMEZ, O.; ANOEP, S.; EPEMA, D. The performance of bags-of-tasks in large-scale distributed systems. In: *Proceedings of the 17th International Symposium on High Performance Distributed Computing*. New York, NY, USA: ACM, 2008. (HPDC '08), p. 97–108. ISBN 978-1-59593-997-5. Disponível em: ⟨http://doi.acm.org/10.1145/1383422.1383435⟩.

ISMAEEL, S.; MIRI, A.; CHOURISHI, D.; DIBAJ, S. M. R. Open source cloud management platforms: A review. In: *2015 IEEE 2nd International Conference on Cyber Security and Cloud Computing*. [S.l.: s.n.], 2015. p. 470–475.

JORISSEN, K.; JOHNSON, W.; VILA, F. D.; REHR, J. J. High-performance computing without commitment: Sc2it: A cloud computing interface that makes computational science available to non-specialists. In: *2012 IEEE 8th International Conference on E-Science*. [S.l.: s.n.], 2012. p. 1–6.

JULA, A.; SUNDARARAJAN, E.; OTHMAN, Z. Cloud computing service composition: A systematic literature review. *Expert Systems with Applications*, v. 41, n. 8, p. 3809 – 3824, 2014. ISSN 0957-4174. Disponível em: ⟨http://www.sciencedirect.com/science/article/pii/S0957417413009925⟩.

KHAN, I.; REHMAN, H. u.; ANWAR, Z. Design and deployment of a trusted eucalyptus cloud. In: *2011 IEEE 4th International Conference on Cloud Computing*. [S.l.: s.n.], 2011. p. 380–387. ISSN 2159-6182.

KLIAZOVICH, D.; BOUVRY, P.; AUDZEVICH, Y.; KHAN, S. U. Greencloud: A packet-level simulator of energy-aware cloud computing data centers. In: *2010 IEEE Global Telecommunications Conference GLOBECOM 2010*. [S.l.: s.n.], 2010. p. 1–5. ISSN 1930-529X.

KUMAR, R.; JAIN, K.; MAHARWAL, H.; JAIN, N.; DADHICH, A. Apache cloudstack: Open source infrastructure as a service cloud computing platform. p. 111–116, 2014.

KUMAR, V.; RAO, V. N. Parallel depth first search. part ii. analysis. *Int. J. Parallel Program.*, Kluwer Academic Publishers, Norwell, MA, USA, v. 16, n. 6, p. 501–519, dez. 1987. ISSN 0885-7458. Disponível em: ⟨http://dx.doi.org/10.1007/BF01389001⟩.

LEE, Y. C.; ZOMAYA, A. Y. Practical scheduling of bag-of-tasks applications on grids with dynamic resilience. *IEEE Transactions on Computers*, v. 56, n. 6, p. 815–825, June 2007. ISSN 0018-9340.

MCINTOSH-SMITH, S.; PRICE, J.; DEAKIN, T.; POENARU, A. Comparative benchmarking of the first generation of hpc-optimised arm processors on isambard. *Concurrency and Computation Practice and Experience - Special Issue on the Cray User Group*, p. –, 2018.

MELL, P. M.; GRANCE, T. *The NIST definition of cloud computing*. [s.n.], 2011. Disponível em: ⟨http://dx.doi.org/10.6028/NIST.SP.800-145⟩.

MISHRA, B. Guidelines on how to contribute to the development of openstack. In: *2017 International Conference on Computing Networking and Informatics (ICCNI)*. [S.l.: s.n.], 2017. p. 1–6.

NIMBUS. *Nimbus is cloud computing for science*. 2018. Acesso em: 31 mai. 2018. Disponível em: ⟨http://www.nimbusproject.org/⟩.

NúñEZ, A.; VáZQUEZ-POLETTI, J. L.; CAMINERO, A.; CASTAñé, G. G.; CARRETERO, J.; LLORENTE, I. M. Icancloud: A flexible and scalable cloud infrastructure simulator. v. 10, p. 185–209, 03 2012.

OKI, E.; KANEKO, R.; KITSUWAN, N.; KURIMOTO, T.; URUSHIDANI, S. Cloud provider selection models for cloud storage services to meet availability requirements. In: *2017 International Conference on Computing, Networking and Communications (ICNC)*. [S.l.: s.n.], 2017. p. 244–248.

OPENNEBULA. *OpenNebula documentation*. 2018. Acesso em: 24 mai. 2018. Disponível em: ⟨https://opennebula.org/documentation/⟩.

OPENSTACK. *Get started with OpenStack*. 2018. Acesso em: 22 mai. 2018. Disponível em: ⟨https://docs.openstack.org/install-guide/get-started-with-openstack.html⟩.

PASTOR, L.; BOSQUE, J. L. An efficiency and scalability model for heterogeneous clusters. In: *Proceedings of the 3rd IEEE International Conference on Cluster Computing*. Washington, DC, USA: IEEE Computer Society, 2001. (CLUSTER '01), p. 427–. ISBN 0-7695-1116-3. Disponível em: ⟨http://dl.acm.org/citation.cfm?id=792761.793264⟩.

ROSENBERG, A.; CHIANG, R. Heterogeneity in computing: Insights from a worksharing scheduling problem. v. 22, 11 2011.

ROSENBERG, A. L.; CHIANG, R. C. Toward understanding heterogeneity in computing. In: *2010 IEEE International Symposium on Parallel Distributed Processing (IPDPS)*. [S.l.: s.n.], 2010. p. 1–10. ISSN 1530-2075.

SENGER, H.; HRUSCHKA, E. R.; SILVA, F. A.; SATO, L. M.; BIANCHINI, C. P.; JEROSCH, B. F. Exploiting idle cycles to execute data mining applications on clusters of pcs. *Journal of Systems and Software*, Elsevier, v. 80, n. 5, p. 778–790, 2007.

SENGER, H.; SILVA, F. A. B. da. Bounds on the scalability of bag-of-tasks applications running on master-slave platforms. *Parallel Processing Letters*, World Scientific Pub Co Pte Lt, v. 22, n. 02, p. 1250004, jun 2012. Disponível em: ⟨https://doi.org/10.1142/s0129626412500041⟩.

SILVA, F. A. B. da; SENGER, H. Scalability limits of bag-of-tasks applications running on hierarchical platforms. *J. Parallel Distrib. Comput.*, Academic Press, Inc., Orlando, FL, USA, v. 71, n. 6, p. 788–801, jun. 2011. ISSN 0743-7315. Disponível em: ⟨http://dx.doi.org/10.1016/j.jpdc.2011.01.002⟩.

SILVA, F. A. B. da; SENGER, H. Scalability analysis of large distributed computing systems. In: *Grid Computing: techniques and future prospects*. [S.l.]: Nova Science Publishers, 2015.

SILVA, F. A. da; SENGER, H. Improving scalability of bag-of-tasks applications running on master–slave platforms. *Parallel Computing*, v. 35, n. 2, p. 57 – 71, 2009. ISSN 0167-8191. Disponível em: ⟨http://www.sciencedirect.com/science/article/pii/S0167819108001129⟩.

SILVA, J. a. N.; VEIGA, L.; FERREIRA, P. Heuristic for resources allocation on utility computing infrastructures. In: *Proceedings of the 6th International Workshop on Middleware for Grid Computing*. New York, NY, USA: ACM, 2008. (MGC '08), p. 9:1–9:6. ISBN 978-1-60558-365-5. Disponível em: ⟨http://doi.acm.org/10.1145/1462704.1462713⟩.

SILVA, R. F. da; GLATARD, T. A science-gateway workload archive to study pilot jobs, user activity, bag of tasks, task sub-steps, and workflow executions. In: SPRINGER. *European Conference on Parallel Processing*. [S.l.], 2012. p. 79–88.

SULISTIO, A.; YEO, C. S.; BUYYA, R. Simulation of parallel and distributed systems: A taxonomy and survey of tools. 08 2018.

SUN, X. H.; ROVER, D. T. Scalability of parallel algorithm-machine combinations. *IEEE Trans. Parallel Distrib. Syst.*, IEEE Press, Piscataway, NJ, USA, v. 5, n. 6, p. 599–613, jun. 1994. ISSN 1045-9219. Disponível em: ⟨https://doi.org/10.1109/71.285606⟩.

THAI, L.; VARGHESE, B.; BARKER, A. Budget constrained execution of multiple bag-of-tasks applications on the cloud. In: *2015 IEEE 8th International Conference on Cloud Computing*. [S.l.: s.n.], 2015. p. 975–980. ISSN 2159-6182.

VAQUERO, L. M.; RODERO-MERINO, L.; CACERES, J.; LINDNER, M. A break in the clouds. *ACM SIGCOMM Computer Communication Review*, Association for Computing Machinery (ACM), v. 39, n. 1, p. 50, dez 2008. Disponível em: ⟨http://dx.doi.org/10.1145/1496091.1496100⟩.

VELHO, P. A. M. d. c. v. Madeira de C. *Accurate and Fast Simulations of Large-Scale Distributed Computing Systems*. Tese (Theses) — Université de Grenoble, jul. 2011. Disponível em: ⟨https://tel.archives-ouvertes.fr/tel-00625497⟩.

YERO, E. J. H.; HENRIQUES, M. A. A. Speedup and scalability analysis of master–slave applications on large heterogeneous clusters. *Journal of Parallel and Distributed Computing*, v. 67, n. 11, p. 1155 – 1167, 2007. ISSN 0743-7315. Disponível em: ⟨http://www.sciencedirect.com/science/article/pii/S074373150700072X⟩.

ZHANG, Q.; CHENG, L.; BOUTABA, R. Cloud computing: state-of-the-art and research challenges. *Journal of Internet Services and Applications*, v. 1, n. 1, p. 7–18, May 2010. ISSN 1869-0238. Disponível em: ⟨https://doi.org/10.1007/s13174-010-0007-6⟩.