



Universidade de Brasília

Instituto de Ciências Exatas
Departamento de Ciência da Computação

Escalonador de tarefas com suporte à GPU para a Plataforma de Nuvens Federadas BioNimbuz

Francisco Anderson Bezerra Rodrigues

Monografia apresentada como requisito parcial
para conclusão do Bacharelado em Ciência da Computação

Orientador

Prof.a Dr.a Aletéia Patrícia Favacho de Araújo

Brasília
2018



Universidade de Brasília

Instituto de Ciências Exatas
Departamento de Ciência da Computação

Escalonador de tarefas com suporte à GPU para a Plataforma de Nuvens Federadas BioNimbuz

Francisco Anderson Bezerra Rodrigues

Monografia apresentada como requisito parcial
para conclusão do Bacharelado em Ciência da Computação

Prof.a Dr.a Aletéia Patrícia Favacho de Araújo (Orientador)
CIC/UnB

Prof.a Dr.a Carla Castanho Edward Ribeiro
CIC - UnB Senado Federal

Prof. Dr. Rodrigo Bonifácio de Almeida
Coordenador do Bacharelado em Ciência da Computação

Brasília, 13 de Julho de 2018

Dedicatória

Dedico esse trabalho à comunidade científica, à Universidade de Brasília, ao Departamento de Ciência da Computação, ao Laboratório de Bioinformática de Dados, à minha orientadora, professora Doutora Aletéia, aos amigos que fiz na faculdade e aos que tenho fora dela. Dedico também à minha família, à Goku, ao Zangado, ao MrSmartDonkey e ao CrazyRussianHacker e a todos que vieram assistir à defesa desta monografia.

Agradecimentos

Agradeço à Universidade de Brasília, por ser uma universidade pública; ao Departamento de Ciência da Computação por seus professores e por (em maioria) ser a favor do software livre; à minha família pela paciência e apoio nos momentos difíceis; ao Zangado por suas lições de vida; aos meus amigos do RPG dominical pelos momentos de descontração; e à todos os meus amigos dentro e fora da faculdade, em especial os de longa data, que não citarei aqui para não correr o risco de esquecer alguém.

Resumo

O uso de arquiteturas heterogêneas é uma realidade no ambiente de nuvem. Entretanto, ainda assim é uma funcionalidade não tão explorada, em especial no ambiente de federações nuvens computacionais. Este trabalho propõe um escalonador de tarefas capaz de fazer escalonamento heterogêneo para o BioNimbuZ, uma plataforma para federação de nuvens computacionais desenvolvida no Laboratório de Bioinforática e Dados(LABID) da Universidade de Brasília, do qual este autor faz parte. O escalonador proposto abre possibilidades de novas tarefas para serem executadas em nuvens computacionais de forma viável.

Palavras-chave: Computação em nuvem, Federação de nuvens, escalonamento, GPGPU, BioNimbuZ

Abstract

The use of heterogeneous architectures is a reality in the cloud environment. However, it is still a feature not explored, especially in the federation environment computational clouds. This work proposes a heterogeneous scheduler of tasks capable of scheduling for BioNimbuZ, a platform for federation of computational clouds developed at the Laboratório de Bioinformática e Dados (LABID) of Universidade de Brasília, of which this author is a part. The proposed scheduler opens possibilities for new tasks to be executed in computational clouds in a viable way.

Keywords: Cloud Computing, Cloud Federation, scheduling, GPGPU, BioNimbuZ

Sumário

1	Introdução	1
1.1	Objetivos	2
1.2	Estrutura do Trabalho	2
2	Nuvem Computacional	3
2.1	Definição de Computação em Nuvem	3
2.2	Modelos de Serviço e Tipos de Nuvens	4
2.3	Federações de Nuvens	5
3	Plataforma BioNimbuz	9
3.1	Visão Geral	9
3.2	Arquitetura	10
3.2.1	Camada de Aplicação	12
3.2.2	Camada de Integração	12
3.2.3	Camada de Núcleo	12
3.2.4	Camada de Infraestrutura	14
3.3	Serviço de Escalonamento	14
3.3.1	<i>Analytic Hierarchy Process</i>	14
3.3.2	<i>Ant Colony Optimization</i>	14
3.3.3	<i>Beam Search</i> Multiobjetivo	15
4	Escalonamento	16
4.1	Definição	16
4.2	Escalonamento em Nuvens Computacionais	17
4.3	Trabalhos Relacionados	19
4.4	Escalonador Proposto para Plataformas de Federação	21
5	Implementação	23
5.1	Sistema de Escalonamento do BioNimbuz	23

5.2 Implementação do Escalonador Proposto	24
5.2.1 Interação Java C++	26
5.3 Testes	27
5.4 Implantação no BioNimbuz	28
5.5 Métricas e Testes	31
5.6 Análise de resultados	34
5.7 Considerações Finais	35
6 Conclusão e Trabalhos Futuros	37
Referências	39
Apêndice	43
A Especificação Das Máquinas Utilizadas Nos Testes	44

Listas de Figuras

2.1 Federação de nuvem Horizontal e Vertical.	6
3.1 Arquitetura do BioNimbuZ [?].	11
4.1 Diferentes contextos em que o escalonamento pode acontecer na nuvem (adaptado de [?]).	18
4.2 Diagrama de Funcionamento do Algoritmo de Escalonamento Proposto. . .	22
5.1 Subsistema de Escalonamento do BioNimbuZ.	23
5.2 Subsistema de Escalonamento do BioNimbuZ.	24
5.3 Arquitetura de Classes do Escalonador Implementado.	25
5.4 Captura do <i>Handshake</i> entre a parte Java e C++ do Escalonador.	26
5.5 Handshake entre a parte Java e C++ do Escalonador.	27
5.6 Diagrama do processo de seleção do escalonador C++.	28
5.7 Parte dos <i>Snapshots</i> existentes na VM de Implantação.	29

Listas de Tabelas

5.1	Tempo Gasto No Escalonamento Com o Escalonador Anterior.	32
5.2	Tempo Gasto No Escalonamento Com o Escalonador Desenvolvido.	33
5.3	Tempo Gasto No Escalonamento Na Parte C++ Do Escalonador Desenvolvido.	33
5.4	Tempo Gasto (Des)Serialização Das Partes Java e C++.	33
5.5	Tempo Para Calcular 50 Mil <i>hashes</i> , Utilizando A CPU	34
5.6	Tempo Para Calcular 50 Mil <i>hashes</i> , Utilizando A GPU	34
5.7	Tempo Médio Gasto No Escalonamento.	34
5.8	Distribuição De Tempo Gasto no Escalonador Desenvolvido.	35
5.9	Tempo Médio E Desvio Padrão Da Execução Do XMR-Stak.	35

Listas de Abreviaturas e Siglas

ACO *Ant Colony Optimization.*

AHP *Analytic Hierarchy Process.*

AMD *Advanced Micro Devices.*

API *Application Programming Interface.*

AWS *Amazon Web Services.*

CFS *Completely Fair Scheduler.*

CPU Unidade de Processamento Central, do inglês *Central Processing Unit.*

CSP *Cloud Service Provider.*

CUDA *Compute Unified Device Architecture.*

DNS *Domain Name System.*

FIFO *First In First Out.*

GCC *GNU Compiler Collection.*

GCP *Google Cloud Platform.*

GNU *GNU's Not Unix[?].*

GPGPU Unidade de Processamento Gráfico de Propósito Geral, do inglês *General Purpose Graphics Processing Unit.*

GPU Unidade de Processamento Gráfico, do inglês *Graphics Processing Unit.*

IaaS *Infrastructure as a Service.*

IDE Ambiente Integrado de Desenvolvimento, do inglês *Integrated Development Environment*.

IP *Internet Protocol.*

JNI *Java Native Interface.*

LABID Laboratório de Bioinformática e Dados.

NIST *National Institute of Standards and Technology.*

P2P *Peer-to-Peer.*

PaaS *Platform as a Service.*

PSO *Particle Swarm Optimization.*

QoS *Quality of Service.*

RAM *Random Access Memory.*

REST *REpresentational State Transfer.*

RPC Chamada de Procedimento Remoto, do inglês *Remote Procedure Call.*

SaaS *Software as a Service.*

SFTP *Secure shell File Transfer Protocol.*

SGBD Sistema Gerenciador de Banco de Dados.

SLA *Service Level Agreement.*

SO Sistema Operacional.

TCP *Transfer Control Protocol.*

TCTP *Trusted Cloud Transfer Protocol.*

TI Tecnologia da Informação.

UDP *User Datagram Protocol.*

UnB Universidade de Brasília.

USB *Universal Serial Bus.*

VM Máquina virtual, do inglês *Virtual Machine.*

Capítulo 1

Introdução

A tecnologia se tornou onipresente na sociedade. Com o advento da Internet, a interação dos seres humanos com a tecnologia cresceu bastante, gerando um tráfego imenso de dados, e com isso a necessidade de processamento em larga escala. Nesse cenário, emergiu o conceito de nuvem computacional, um paradigma que permite processamento em larga escala sem ser necessário que o usuário tenha em mãos hardware com tamanha capacidade computacional.

Assim, grandes empresas da área de Tecnologia da Informação, como a *Google* [1] e a *Microsoft* [?], possuem vários *datacenters* com uma imensa quantidade de computadores interligados via rede, os quais disponibilizam esses recursos de forma virtualizada a usuários que necessitem de processamento e de armazenamento em larga escala. A disponibilidade dessa capacidade de computação tem gerado uma revolução na forma como os serviços computacionais são disponibilizados na Internet. Dessa forma, pequenas empresas agora conseguem prover serviços em larga escala sem necessitarem de um grande investimento em infraestrutura computacional, e grandes empresas conseguem reduzir custos com aquisição e manutenção de equipamentos [?].

Como existem vários provedores de nuvem e cada um deles tem seus pontos fortes e fracos, surgiu então a ideia de criar uma plataforma que utilize serviços de vários provedores de nuvem, podendo explorar o ponto forte de cada um deles. Assim, emergiu o conceito de Federação de Nuvens[2], que são plataformas nas quais os usuários conseguem o máximo de flexibilidade provido pela combinação de funcionalidades que os distintos provedores de nuvem disponibilizam a seus usuários.

Todavia, como toda nova tecnologia, ela possui seus próprios desafios, pois desenvolver uma plataforma com tamanha flexibilidade requer uma arquitetura muito bem projetada, implementada e capaz de ser eficiente, concomitantemente em que sua interface seja agradável ao usuário. Existem várias propostas para federações de nuvens, entre as quais é possível citar Demchenko[2] e Buyya[3]. Uma plataforma de Federação de Nuvens que

tem sido continuamente evoluída é o BioNimbuz [4] [5] [6] [7] [8] [9] [10] [11], desenvolvido no Laboratório de Bioinformática e Dados (LABID) da Universidade de Brasília (UnB) por alunos de graduação e pós-graduação.

O BioNimbuz é uma plataforma para nuvens federadas para a execução de *workflows*, inicialmente de Bioinformática, mas atualmente sua arquitetura suporta *workflows* de propósito geral. Atualmente o BioNimbuz conta com vários algoritmos de escalonamento, entretanto, nenhum deles trata do escalonamento para plataformas heterogêneas. Diante do exposto, este trabalho propõe um novo algoritmo de escalonamento para o BioNimbuz, cujo objetivo é distribuir tarefas para CPUs e GPUs das máquinas virtuais providas pelos diversos provedores de nuvem.

1.1 Objetivos

Este trabalho tem como objetivo principal desenvolver um escalonador para a plataforma BioNimbuz, que seja capaz de escalonar tarefas para arquiteturas de *hardware* heterogêneos, ou seja, compostas por CPUs e GPUs. Para cumprir este objetivo geral, os seguintes objetivos específicos devem ser alcançados:

- Analisar algoritmos de escalonamento para plataformas heterogêneas;
- Desenvolver um escalonador capaz de distribuir tarefas para plataformas heterogêneas;
- Integrar o escalonador desenvolvido ao BioNimbuz;
- Analisar o desempenho do escalonador proposto.

1.2 Estrutura do Trabalho

Este trabalho contém, além deste capítulo introdutório, com mais quatro capítulos. O Capítulo 2 discorre sobre computação em nuvem e federações de nuvens computacionais. O Capítulo 3 descreve a plataforma de nuvens Federadas BioNimbuz. O Capítulo 4 aborda a atividade de escalonamento e propõe o algoritmo que será implementado. O Capítulo 5 discorre sobre como foi o processo de implementação do escalonador proposto. Por último, o Capítulo 6 apresenta as conclusões e alguns trabalhos futuros. O Apêndice detalha quais máquinas foram utilizadas nos testes que foram feitos.

Capítulo 2

Nuvem Computacional

Este capítulo apresenta, inicialmente, a definição de nuvem computacional, suas vantagens e desvantagens. Em seguida, busca-se classificar os tipos de nuvem, tanto em termos de quais serviços são providos, quanto em termos de como a infraestrutura computacional é implementada. Por fim, é apresentado o conceito de federação de nuvens e como essas federações podem combinar provedores de nuvens distintos em prol da performance e da relação custo-benefício.

2.1 Definição de Computação em Nuvem

Por alguns anos a computação em nuvem não possuía uma definição bem definida, principalmente, por ser uma tecnologia nova. Muitas vezes sendo confundida com computação em *Grid*[12]. Atualmente, há várias definições para computação em nuvem, dentre essas, umas das mais usadas foi definida pelo *National Institute of Standards and Technology* (NIST), que conceituou computação em nuvem como um modelo de computação distribuída com as seguintes características [13]:

- Serviço *self-service on demand*;
- Fácil acesso via rede;
- Recursos virtualizados;
- Rápida elasticidade;
- Serviço mensurado.

Em contrapartida, Vaquero[12] definiu nuvem como um grande conjunto de recursos virtualizados, de fácil acesso e uso, que podem ser dinamicamente reconfigurados para se ajustarem a uma carga variável de uso, permitindo um uso otimizado dos recursos. Esse

conjunto de recursos são, geralmente, explorados por um modelo de uso *pay-per-use* no qual garantias são providas pelo provedor da infraestrutura, por meio de *Service Level Agreement* (SLA), que é um acordo de serviço que descreve o nível de qualidade e as garantias providas pelo provedor do serviço de nuvem.

Analisando todas as definições supracitadas, percebe-se que ambas concordam que os recursos computacionais, tais como armazenamento, capacidade de processamento, memória e rede são disponibilizados de forma virtualizada, acessível via rede, na qual o uso desses recursos é monitorado, geralmente, com intenção de cobrança. Além de ser possível redimensionar os recursos disponíveis para uso *on-the-fly*, funcionalidade conhecida por elasticidade. Essa característica do ambiente de nuvem pode ser implementada de várias formas [14]: seja redimensionando uma Máquina virtual, do inglês *Virtual Machine* (VM) em execução (chamada elasticidade vertical), ou seja criando novas máquinas virtuais para ajudar a prover o serviço (chamada elasticidade horizontal), geralmente associadas a um distribuidor de carga.

Com base no conjunto de características citadas, é possível visualizar as vantagens desta tecnologia, as quais são: reduz o investimento inicial de empresas em infraestrutura de TI; a elasticidade, além de não desperdiçar investimentos feitos em servidores em momentos de pouca carga. Isso tornou as nuvens computacionais populares, e um bom negócio para as grandes empresas da área de Tecnologia da Informação (TI), que são capazes de prover infraestrutura computacional, como a *Oracle*, a *Microsoft*, a *IBM*, a *Google*, entre outras.

2.2 Modelos de Serviço e Tipos de Nuvens

Os mesmos autores que definem nuvem computacional também buscam categorizar os tipos de nuvens. Assim, em relação aos tipos de serviços oferecidos, existem três classificações para nuvem computacional, que são [12] [13]:

- ***Infrastructure as a Service* (IaaS):** O provedor disponibiliza recursos computacionais virtualizados diretamente, em termos de máquinas virtuais. Esses recursos são capazes de serem reconfigurados dinamicamente pelo serviço de elasticidade, por exemplo a *Oracle Cloud* [15];
- ***Plataform as a Service* (PaaS):** Uma plataforma sobre a qual os usuários podem implantar softwares como serviços é disponibilizada. Os programas implantados podem fazer uso de algumas funcionalidade fornecidas em forma de bibliotecas, linguagens de programação, entre outros. Um exemplo é o serviço de hospedagem

de sites[?], que são disponibilizados, por exemplo, no *Amazon Web Services* (AWS) [16] e no *Google Cloud Platform* (GCP) [17];

- **Software as a Service (SaaS)**: O usuário tem acesso ao software que está sendo executado na nuvem computacional. O que permite migrar requisito computacional do equipamento no qual o serviço executava para a nuvem. Um exemplo bastante conhecido é o *Google Docs* [18].

A categorização de nuvens computacionais por meio dos serviços provados não é a única forma de classificar nuvens. Elas também são categorizadas de acordo com a forma em que são implantadas. Assim, elas podem ser públicas, privadas, híbridas e comunitárias, descritas abaixo [13]:

- **Nuvem pública**: Criada para ser usada pelo público em geral. Ela pode ser disponibilizada e/ou mantida por uma organização, com ou sem fins lucrativos;
- **Nuvem privada**: Quando a infraestrutura da nuvem é provida por uma única organização, para uso interno. Ela pode ser gerenciada e/ou operada por um terceiro. Nuvens privadas são uma solução para o problema de privacidade dos dados, pois ela pode estar disponível apenas internamente na organização;
- **Nuvem comunitária**: Nuvem com infraestrutura desenvolvida com o objetivo de ser utilizada por um grupo de pessoas e/ou organizações para fins comuns. Ela pode ser gerenciada por uma ou mais partes dos interessados na nuvem;
- **Nuvem híbrida**: Combinação de duas ou mais das opções anteriores. Geralmente, unificadas por meio de padronizações de protocolos de comunicação ou uso de tecnologia proprietária.

2.3 Federações de Nuvens

Como cada provedor de nuvem possui seu próprio método de especificação, uma interface própria para comunicação com os usuários de seus serviços e funcionalidades específicas, surgiu o conceito de federações de nuvens computacionais. O objetivo dessa plataforma é minimizar dependências entre os provedores do serviço e também reduzir custos, aproveitando o melhor de cada provedor de nuvem. Assim, federação de nuvem é quando diferentes provedores de serviço de nuvem são agrupados e organizados para interoperarem entre si, seja pela busca de lucro, garantia de SLA, ou mesmo uma melhor relação custo-benefício para o usuário.

Existem duas formas nas quais plataformas de federação podem combinar os serviços das nuvens: horizontalmente e verticalmente [19][20]. No modelo vertical instâncias de

nuvem de um provedor A são criadas a partir de instâncias de nuvem de um provedor B, que pode solicitar serviços de um provedor de nuvem C, e assim por diante. No modelo horizontal as nuvens trabalham sem existir uma hierarquia entre si, assemelhando-se bastante com sistemas de rede *peer to peer*. É possível combinar mais de um estilo em uma mesma federação, conforme mostrado na Figura 2.1

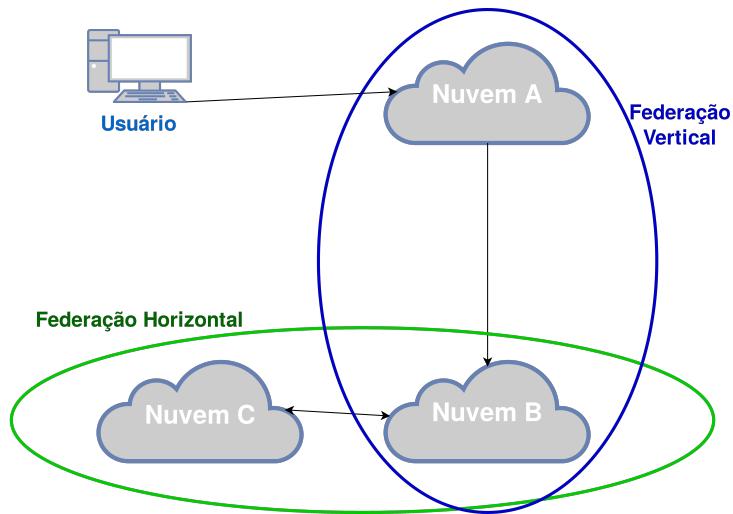


Figura 2.1: Federação de nuvem Horizontal e Vertical.

Buyya *et al.* [3] descrevem uma arquitetura para federações de nuvens que faz uso de um *exchange* para aproximar clientes de provedores de nuvem, detalhando cada um de seus componentes em artigos distintos [21] [22] [23] [24]. Entretanto, o sistema descrito não menciona nada sobre virtualização de máquinas compostas por arquiteturas heterogêneas.

Chen *et al.* [20] exploram as vantagens de uso de nuvens horizontais e verticais concomitantemente, utilizando a teoria dos jogos para auxiliar no processo de decisão sobre quando fazer *outsourcing* de *workload*, explorando a verticalidade da federação e quando remanejar o *workload* para instâncias da coalisão (nuvem horizontal). Um dos resultados obtidos revelou que o melhor para a performance de uma federação de nuvens é explorar tanto verticalmente quanto horizontalmente as nuvens existentes.

Margheri *et al.* [25] propõem uma arquitetura de federações de nuvens completamente distribuída, que utiliza uma *blockchain* para prover um controle democrático da federação. Com grande foco em segurança e estabilidade, a arquitetura proposta provê a cada um dos membros o mesmo nível de controle sobre a federação, utiliza criptografia para tornar anônimo a origem dos serviços de computação. E utiliza *blockchain* para prover democracia na gestão da federação.

Alansari *et al.* [26], propõem um sistema de controle de acesso com alta granularidade com o objetivo de provê compartilhamento seguro de informações, com grande foco na segurança e na privacidade dos usuários, especificando um protocolo de troca de chaves

para processamento remoto de dados na nuvem. Também fazendo uso de *blockchain* para prover integridade de chaves públicas e outras informações necessárias do protocolo. Esse trabalho faz uso de tecnologia proprietária em CPUs da Intel para fazer o processamento, o que significa que não possui nenhum porte para uso em equipamentos compostos por arquiteturas heterogêneas.

Gallico *et al.* [27] apresentam a federação de nuvens CYCLONE, uma federação de nuvens em desenvolvimento com o objetivo, tanto comercial quanto científico, que utiliza as ferramentas *open-source* *OpenNaaS*[?], *OpenStack*[?] e *Trusted Cloud Transfer Protocol* (TCTP)[?].

Jrad *et al.* [28] propõem um *framework* para executar *workflows* baseado no uso de um mediador entre os usuários e os provedores das nuvens. O mediador é capaz de negociar SLAs para buscar provedores que consigam garantir entrega do serviço a um custo menor. Faz uso de um *workflow engine* para processamento e clusterização de *workflows* requisitados. Resultados obtidos experimentalmente mostram redução significativa no custo do processamento dos *workflows*.

Mashayekhy *et al.* [29] propõem o uso da teoria dos jogos para provedores de nuvens se aliarem em federações para situações em que sua infraestrutura não é capaz de lidar com a demanda de serviços. A teoria dos jogos é utilizada com o objetivo de determinar as melhores nuvens externas para *outsourcing* do excesso de demanda.

Demchenko *et al.* [2] apresentam resultados obtidos em uma pesquisa em andamento sobre *framework* para provisionamento de infraestrutura de nuvem *on-demand*. O trabalho complementa a arquitetura da nuvem computacional de forma a simplificar federação horizontal para os tipos principais de serviços de nuvem (IaaS, PaaS e SaaS).

Marosi *et al.* [30] propõem uma arquitetura de camadas que interage com intermediários de intermediários, conhecido como *meta-brokering*, de outras nuvens para interação com o máximo possível de provedores para fornecer infraestrutura sob demanda, tanto para nuvens públicas quanto para nuvens privadas. Zant e Gagnaire [31] exploram federações horizontais do ponto de vista *Cloud Service Provider* (CSP) com o objetivo de maximizar os lucros das mesmas. Essas federações são invisíveis ao cliente da nuvem, e considera que os CSPs participantes da federação possuem equipamentos semelhantes. Os testes provam aumento no faturamento, entretanto, GPUs não são citadas ou utilizadas nos testes.

Saldanha [11] propõe o BioNimbuz, que é uma plataforma de federação de nuvens em contínua evolução, como pode ser visto em [4] [5] [6] [8] [9] [10]. Uma vez que o autor faz parte do grupo de desenvolvimento do BioNimbuz, o BioNimbuz será utilizado como plataforma para o qual o escalonador será desenvolvido. Assim, o capítulo seguinte apresenta em detalhes o BioNimbuz, a plataforma de federação de nuvens computacionais

selecionada para utilização do escalonador desenvolvido neste trabalho.

Capítulo 3

Plataforma BioNimbuZ

Este capítulo possui como objetivo apresentar a plataforma de nuvens federadas BioNimbuZ, sobre o qual a autor pertence ao grupo de pesquisa, e escolheu desenvolver o escalonador para plataformas heterogêneas proposto neste trabalho. A primeira seção apresenta de forma geral as principais características do BioNimbuZ, e descreve sua evolução ao longo do tempo. A segunda seção descreve a arquitetura do BioNimbuZ e seus principais componentes. A última seção foca sobre os escalonadores que existem atualmente no BioNimbuZ.

3.1 Visão Geral

O BioNimbuZ é uma plataforma livre de nuvens federadas para execução de *workflows*, desenvolvido no Laboratório de Bioinformática e Dados (LABID)/UnB por alunos de graduação e pós-graduação. Ele foi originalmente proposto por Saldanha[11] e refinado por alunos de iniciação científica, graduação, mestrado e doutorado[4] [6] [8] [9] [10] [32].

É possível integrar nuvens de diversos tipos de governança (vide Capítulo 2), o que permite que cada provedor mantenha suas políticas e características internas, e isso reduz a dependência dos usuários de provedores específicos de nuvem. Essa funcionalidade é alcançada graças à facilidade e flexibilidade na inclusão de novos provedores na plataforma, o qual é regida por utilização de *plugins* de integração, que traduzem requisições vindas da plataforma para comandos equivalentes específicos de cada servidor. Isso é fundamental para evitar *vendor lock-in*[?], que é a dependência de um serviço a um provedor em específico, por mais que existam outros.

Em sua concepção inicial, a plataforma BioNimbuZ foi implementada toda por meio de comunicação *Peer-to-Peer* (P2P), porém, durante sua evolução optou-se por integrar o Zookeeper[33] à plataforma para gerenciar os serviços distribuídos.

Moura *et al.*[6] desenvolveram a Chamada de Procedimento Remoto RPC[34] para a plataforma. Fazendo uso do Apache Avro[35], que possibilita comunicação de forma transparente entre computadores, para que seja possível a chamada de procedimentos em outros computadores via rede. Além disso, o núcleo do BioNimbuZ foi refatorado para utilização do Apache ZooKeeper[33], e também foi implementada uma política de armazenamento que considera latência e o local em que o serviço será executado, utilizando *Secure shell File Transfer Protocol* (SFTP) para transferência de arquivos.

Barreiros *et al.*[36] refinaram a política de armazenamento, que analisa a viabilidade de compactação de arquivos pré-transferência com base na largura de banda, tempo de transferência de arquivos e tempo de compressão e descompressão, que será melhor descrito na sessão 3.3.3.

Ramos *et al.*[37] desenvolveram um controlador de *jobs* para comunicação entre o núcleo do BioNimbuZ e a Camada de Interface com o Usuário, além de desenvolver uma interface gráfica que tornou a plataforma mais acessível.

Novamente, Barreiros[9] desenvolveu o escalonador que se baseia no *beam search* interativo multiobjetivo, chamado C99, com o objetivo de aumentar a eficiência do escalonamento, que passou a levar em consideração o custo por hora dos recursos a serem alocados.

Santos[38] refinou o serviço de armazenamento do BioNimbuZ, adicionando suporte ao uso de serviços de armazenamentos providos pelos diferentes provedores de nuvem, melhorando a praticidade e a usabilidade da plataforma.

Ao longo dos trabalhos supracitados, que evoluíram o sistema proposto inicialmente por Saldanha[11], o BioNimbuZ adquiriu uma arquitetura em camadas, que será descrita na próxima seção.

3.2 Arquitetura

O BioNimbuZ foi implementado por meio de uma arquitetura em camadas, dispostas de forma hierárquica e distribuída. Ele possui quatro camadas principais: Aplicação, Integração, Núcleo e Infraestrutura, como apresentado na Figura 3.1.

Internamente, o BioNimbuZ utiliza o Apache Zookeeper[33] para prover serviços de coordenação de ambientes distribuídos. O Apache Zookeeper foi desenvolvido pela fundação Apache[39], tem como objetivo ser de fácil manuseio. Ele possui um modelo de dados semelhante a uma estrutura de diretórios.

Uma outra tecnologia que também é utilizada no BioNimbuZ é o Apache Avro[35], para serialização de dados para transmissão pela rede.

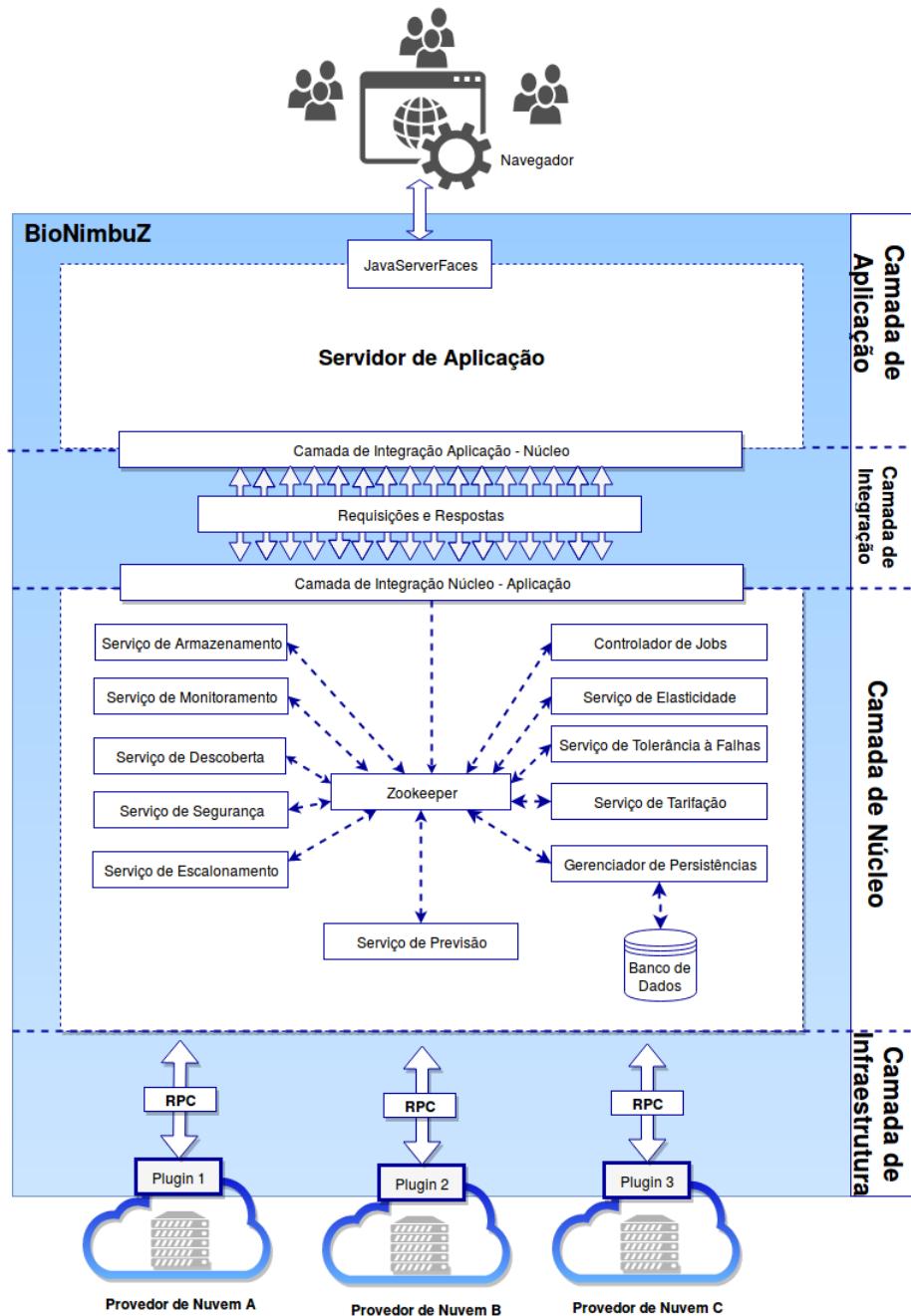


Figura 3.1: Arquitetura do BioNimbuZ [?].

3.2.1 Camada de Aplicação

Esta camada é responsável por prover a interface de comunicação com o usuário, seja via uma interface gráfica (GUI), seja via *web*. Após fazer *login*, o usuário pode submeter *workflows* para serem executados e fazer *upload* dos arquivos necessários. Além de poder acompanhar o andamento de seus *workflows* e obter, caso queira, os resultados que já tiverem sido produzidos.

3.2.2 Camada de Integração

A Camada de Integração tem como objetivo integrar as Camadas de Aplicação e de Núcleo, fazendo uso do *framework REST* para prover de forma prática essa funcionalidade, utilizando operações definidas no protocolo *HTTP*, como *GET*, *DELETE* e *PUT*. Existem três tipos de mensagens trocadas entre o Núcleo e a Camada de Aplicação:

- *Request*: Requisições da camada de Aplicação que contém todos os dados necessários para o seu processamento;
- *Response*: Respostas que definem as mensagens enviadas da camada de Núcleo do BioNimbuz;
- *Action*: Comandos a serem executados pelo núcleo, que são uma requisição enviada ao núcleo para se obter dados na resposta.

3.2.3 Camada de Núcleo

A Camada de Núcleo é a camada mais importante do BioNimbuz. Ela realiza toda a gerência da federação, provendo vários serviços. Entre eles:

- Serviço de Predição: Objetiva orientar o usuário do BioNimbuz a escolher as melhores combinações de máquinas virtuais/provedores a partir da especificação do *workflow* a ser executado e do custo pretendido;
- Serviço de Tarifação: Responsável por calcular o valor que os usuários devem pagar pelos serviços provindos do BioNimbuz. Para tal, comunica-se com o serviço de monitoramento para obter informações tais como tempo de execução e quantidade de máquinas virtuais alocadas. É função deste serviço garantir o cumprimento das métricas de tarifação das nuvens integradas à federação, e repassar o valor ao usuário;
- Serviço de Segurança: Realiza, principalmente, a autenticação de usuário, além de verificar as autorizações do mesmo. Contudo, muitos outros aspectos de segurança

computacional podem ser implementados por este serviço, tais como criptografia na troca de mensagens;

- Serviço de Tolerância a Falhas: Como o nome diz, este serviço é responsável em certificar que todos os serviços do BioNimbuZ estejam disponíveis o máximo de tempo possível. Além de ter a responsabilidade de tratar quaisquer falhas que venham a ocorrer, tira vantagem da arquitetura distribuída do BioNimbuZ para prover redundância de dados;
- Serviço de Armazenamento: Possui a responsabilidade de gerenciar arquivos utilizados como entrada e/ou saída de cada estágio de um *workflow*. Deve desempenhar seu papel de forma eficiente, do ponto de vista de custos de armazenamento e transmissão desses dados entre o local que está armazenado e o local que serão processados;
- Serviço de Elasticidade: Tem como objetivo manipular, dinamicamente, as máquinas virtuais que estão associadas a um tarefa, com o objetivo de otimizar o uso dos recursos, como também, aumentar a capacidade de processamento de equipamentos, além de poder alocar mais de uma máquina para uma tarefa.
- Serviço de Descoberta: É o responsável por identificar os provedores de nuvem disponíveis, e armazenar informações sobre eles tais como latência de rede, capacidade de armazenamento e processamento, além de informações sobre arquivos de entrada e saída dos *workflows* e argumentos para execução das VMs.
- Serviço de Escalonamento: Responsável por fazer o escalonamento dos *jobs* submetidos ao BioNimbuZ. Não é responsabilidade do Serviço de Escalonamento lidar com dependências, pois essa responsabilidade é do Controlador de *Jobs*. Assim o Serviço de Escalonamento só fará a distribuição dos *jobs* que estiverem prontos para serem executados. Diante do objetivo deste trabalho, o Serviço de Escalonamento do BioNimbuZ será detalhado na próxima seção.;
- Serviço de Monitoramento: Responsável por monitorar o estado das VMs instanciadas pela plataforma, e disponibilizar essa informação ao usuário;
- Gerenciador de Persistência: Tem como objetivo organizar o armazenamento de arquivos gerados durante a execução dos *workflows*, além de *uploads* de arquivos que serão utilizados como entrada em futuros *workflows*;
- Controlador de *Jobs*: Responsável por gerir *jobs*, em especial, os que possuem dependência ficam à espera até poderem ser escalonados.

3.2.4 Camada de Infraestrutura

A Camada de Infraestrutura disponibiliza uma interface de comunicação do BioNimbuz com os provedores de nuvem. Esta camada utiliza *plugins* para mapear requisições provenientes da Camada de Núcleo para comandos específicos de cada provedor.

O BioNimbuz é capaz de ser integrado tanto à nuvens públicas quanto privadas, utilizando *plugins* para permitir conexão com vários provedores de nuvem, cada qual com sua própria interface. Os *plugins* não existem apenas na camada de integração, pois vários serviços da camada de núcleos também são disponibilizados como tal, provendo grande flexibilidade à plataforma.

3.3 Serviço de Escalonamento

Em suas primeiras fases de desenvolvimento, o escalonador do BioNimbuz apenas realizava uma associação *First In First Out* (FIFO) entre *jobs* disponíveis e as VMs instanciadas. Em seguida, esta política de escalonamento evoluiu para um escalonamento *Round-Robin*. Desde então, foram propostos vários escalonadores diferentes, que serão explicados a seguir.

3.3.1 *Analytic Hierarchy Process*

Em sua primeira versão oficial, o BioNimbuz possuía um escalonador baseado no *Analytic Hierarchy Process* (AHP)[8], uma técnica de análise de decisões complexas, de amplo uso no meio corporativo. O problema é quebrado em uma estrutura hierárquica que cada elemento da hierarquia interage apenas com níveis imediatamente superiores e inferiores. Após análise comparativa de como cada elemento interfere em todos os outros elementos dos níveis hierárquicos vizinhos, é obtido o resultado da análise, que revela as opções escolhidas com base nos critérios analisados.

3.3.2 *Ant Colony Optimization*

Esta solução é baseada na meta-heurística *Ant Colony Optimization* (ACO)[40] para o problema de otimização combinatória difícil. Esta meta-heurística foi inspirada no comportamento das formigas enquanto andam. Oliveira *et al.*[8] desenvolveram e implementaram um escalonador para o BioNimbuz com esta proposta de solução, que obteve resultados positivos comparados ao escalonador anterior, baseado no AHP.

3.3.3 *Beam Search* Multiobjetivo

O escalonador utilizado atualmente, desenvolvido por Barreiros[9], é um escalonador cuja proposta principal é ser interativo e multiobjetivo, ou seja, que pode ser interrompido durante sua execução e ainda assim retornar um resultado válido. Este algoritmo considera, durante o processamento, a capacidade das máquinas virtuais disponíveis e o preço de cada uma delas. O escalonamento ocorre em três etapas, as quais são:

1. Busca gulosa sobre conjuntos de Pareto, ou seja, conjuntos de resultados considerados equivalentes dado um conjunto de critérios, para poda (remoção) de conjuntos soluções não ideais;
2. *Limited Discrepancy Search*[?], cujo objetivo é impedir que a poda (remoção) do conjuntos soluções que pode levar a um conjunto solução melhor futuramente; e
3. *Beam Search* Iterativo, que busca iterativamente os melhores conjuntos resultado.

Uma funcionalidade que não foi abordada nos escalonadores citados acima é que eles não possuem suporte para executar *jobs* em GPUs, apenas em CPUs. Diante do exposto, o objetivo deste trabalho é considerar no escalonamento as máquinas com CPUs e GPUs. O próximo capítulo discorrerá sobre escalonadores e apresentará o algoritmo de escalonamento proposto neste trabalho.

Capítulo 4

Escalonamento

nEste capítulo apresenta, inicialmente, os diferentes tipos de escalonadores, citando as várias áreas em que a atividade de escalonamento ocorre. Em seguida apresenta a definição do problema de escalonamento que este trabalho busca resolver. Em sequência, é discorrido sobre trabalhos relacionados para então apresentar o algoritmo de escalonamento proposto.

4.1 Definição

O escalonamento é um problema clássico da computação que surgiu junto com os sistemas operacionais multitarefas. A sua complexidade é considerada NP-difícil[41], porém isso não impedi a evolução dos sistemas operacionais, cujos escalonadores são desenvolvidos com objetivos e metodologias diferentes para alcançarem uma boa performance. De qualquer forma, um problema que todos os escalonadores sistemas operacionais multitarefa buscam resolver é o *starvation*, que é quando um processo/*thread* fica esperando indefinidamente por um recurso, que pode ser, inclusive, a CPU [?].

Escalonadores podem ser categorizados de várias formas, com base nas funcionalidades que possuem. Assim, uma importante questão é definir qual é a unidade escalonada, podendo ser[?]:

- processos, *threads* para serem executados;
- *jobs* e *tasks* para serem executados na nuvem;
- páginas da memória *Random Access Memory* (RAM) para *swapping*;
- pacotes para serem transmitidos pela rede;
- requisições de leitura/escrita em memória secundária.

Além da unidade que é escalonada, outras características que ajudam a definir um escalonador é se ele permite que tarefas sejam interrompidas para que outras possa ser executadas, funcionalidade conhecida como preempção. Além disso, o escalonador pode ou não ser capaz de definir prioridade diferentes para cada um dos objetos que está sendo escalonado, e/ou dá suporte para processos que podem requisitar urgência para sua execução, conhecidos como processos de tempo-real[?].

O problema do escalonamento é o método pelo qual o trabalho, definido por algum conjunto de características (como duração e requisitos), é atribuído aos recursos que são capazes de completá-lo.

Atualmente, as CPUs possuem vários núcleos, e são capazes de ter mais de um contexto carregado por núcleo (vide RyzenTM ThreadripperTM[42]). O que faz com que seja necessário que o processo de escalonamento leve em consideração como o mesmo será distribuído (ou não) entre os núcleos.

A atividade de escalonamento pode ser otimizada para vários objetivos, entre os quais é possível citar[?]:

- Maximizar quantidade de trabalho realizada por unidade de tempo;
- Minimizar tempo no qual trabalhos ficam esperando para serem executados;
- Minimizar tempo entre um conjunto de trabalhos estar pronto para ser executado até o fim da execução do conjunto (*makespan*);
- Distribuir de forma justa o tempo que cada um dos trabalhos terá de uso de um recurso escasso.

Esses objetivos são, às vezes, contraditórios. Na prática, prioriza-se um conjunto de métricas como base para otimização. Por exemplo, o GNU/Linux utiliza o *Completely Fair Scheduler* (CFS), que se baseia no algoritmo *Fair queuing*. Como o nome já diz, o foco desse escalonador está em ser justo. Internamente, utiliza-se uma árvore vermelho e preta indexada pelo tempo gasto no processador. Para ser justo, o tempo máximo de cada processo na CPU é o quociente do tempo que o processo ficou aguardando para ser executado pelo número total de processos.

4.2 Escalonamento em Nuvens Computacionais

O escalonamento em nuvens computacionais podem ocorrer em vários contextos e com focos distintos [?], como podemos ver na Figura 4.1. O escalonamento ocorre geralmente de uma camada inferior para uma camada superior da pilha de serviços da nuvem, mas no caso de federações horizontais o escalonamento pode ocorrer numa mesma camada.

No contexto da camada de virtualização o objetivo do escalonamento é mapear recursos virtualizados a recursos físicos, podendo focar no balanço de carga, ou seja, distribuir o máximo possível o processamento entre os recursos físicos disponíveis para evitar que sobrecargas ocorram enquanto outra parte dos recursos estão ociosos, na conservação de energia, buscando alocar no mínimo de máquinas disponíveis, para reduzir custo energético ou então focar no custo-benefício, que, é parecido com o foco em conservação de energia, porém leva em consideração também outros fatores que podem interferir no custo, como a localização do equipamento, como o custo do próprio equipamento, histórico de utilização do usuário (se o mesmo costuma subutilizar os recursos ou não), entre outros fatores [?].

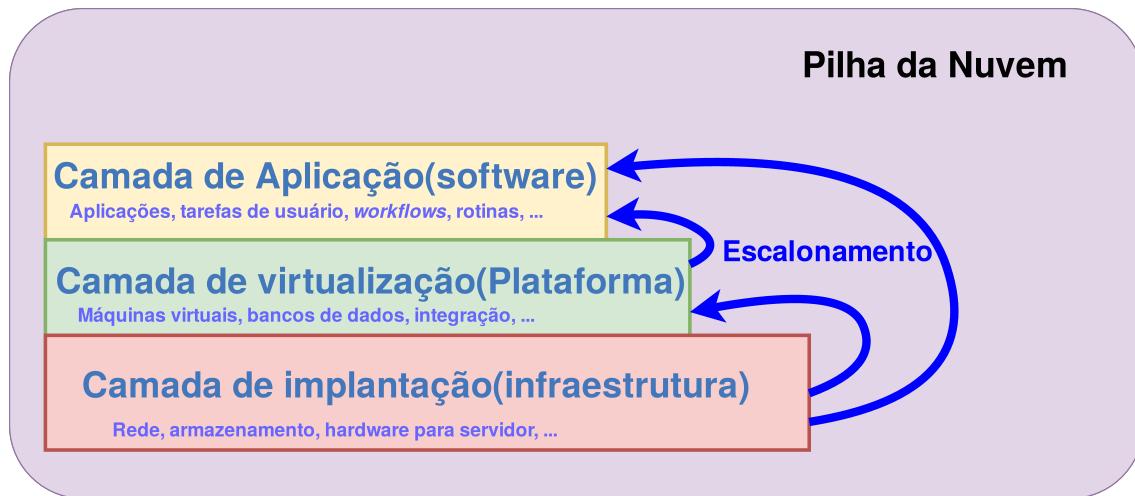


Figura 4.1: Diferentes contextos em que o escalonamento pode acontecer na nuvem (adaptado de [?]).

Tendo a camada de aplicação como contexto, a atividade de escalonamento envolve aplicações de usuário, tarefas, e *workflows* buscando otimizar eficiência do uso dos recursos físicos, podendo focar na garantia do *Quality of Service* (QoS) contratado pelo usuário, em termos de performance, custo, confiabilidade entre outros aspectos, na eficiência do provedor, ou seja, otimizando o uso das máquinas com objetivo de melhorar o uso dos recursos computacionais para o provedor e escalonamento foca em negociação, ou seja, cujo foco é garantir o cumprimento de *Service Level Agreement* (SLA)s, que são contratos de serviço entre o usuário do serviço e o provedor da nuvem [?].

Na camada de implantação, a atividade de escalonamento envolve a organização da infraestrutura de provedores do serviço de nuvem. Também chamado *cluster scheduling*, é neste contexto em que existem os escalonadores Mesos, Omega, Borg, Kubernetes entre outros [?]. Estes escalonadores, como são críticos para grandes empresas da área de computação em nuvem, são escalonadores que ponderam várias métricas, incluindo garantia de serviço, balanço de carga, e performance.

Este trabalho focará no escalonamento em federações de nuvens computacionais. O qual recentemente presenciou a ascensão do uso de Unidade de Processamento Gráfico, do inglês *Graphics Processing Unit* (GPU) para processamento de propósito geral (Unidade de Processamento Gráfico de Propósito Geral, do inglês *General Purpose Graphics Processing Unit* (GPGPU)[43][44]. Nesse contexto, o problema do escalonamento pode ser formalmente definido como, dado:

- Conjunto T de tarefas;
- Conjunto M de máquinas virtuais;
- Conjunto $C, |C| \geq |M|$ de CPUs das máquinas virtuais;
- Conjunto $G, |G| \leq |M|$ de GPUs das máquinas virtuais;
- Função $F : T \times (C \cup G) \rightarrow \mathbb{R}$, o qual estima o tempo de execução da tarefa T_i no recurso designado.

Encontrar uma função injetora $A : T \rightarrow C \cup G$, que minimize $\sum_{t \in T} F(t, A(t))$. Ou seja, buscar uma associação de tarefas para máquinas virtuais que minimize o tempo gasto para o processamento de todo o conjunto de tarefas.

Com base na taxonomia apresentar por Zhan *et al.* [?], o problema de escalonamento abordado neste trabalho é considerado escalonamento com foco em federações parceiras na camada de infraestrutura. Outros tipos de escalonamento na camada de infraestrutura citados no mesmo trabalho são escalonamento para posicionamento do serviço, que busca delegar tarefas a locais próximos de onde o serviço foi solicitado, e o escalonamento focado em roteamento de dados, o qual busca alojar VMs para uma tarefa descentralizada e forma a minimizar o gasto em comunicação entre as partes da aplicação.

4.3 Trabalhos Relacionados

Por mais que o problema do escalonamento seja um clássico na área da Ciência da Computação, são poucos que lidam com escalonamento em nuvens computacionais, e ainda menos os trabalhos que lidam com nuvens compostas por arquiteturas heterogêneas.

Gouasmi *et al.* [45] apresentam um algoritmo de *MapReduce*[46] para escalonamento em nuvens federadas que foca em priorizar a execução de *jobs* em nuvens/máquinas virtuais que já contém os dados necessários para a execução, com o objetivo de evitar transferências desnecessárias na rede. O algoritmo proposto é completamente distribuído e melhor do que o *MapReduce* anteriormente utilizado, porém nada é dito sobre escalonamento para máquinas com arquiteturas heterogêneas.

Nguyen e Thoai [47] propõem um sistema de intermediação para federações de nuvens horizontais, com foco na busca da melhor nuvem para auxiliar sobre carga de serviços para execução. Levando em consideração que provedores de nuvens diferentes podem ter sua infraestrutura customizada com o objetivo de atender diferentes perfis de usuários que mais usam seus serviços, o algoritmo proposto leva em consideração tais características individuais de cada provedor. Novamente, nada é tratado sobre nuvens compostas por equipamentos com arquiteturas heterogêneas.

Jennings e Stadler [48] documentaram metodologias, utilizadas em diferentes contextos que o escalonamento ocorre em ambientes de nuvens computacionais, revelando detalhadamente muitos dos desafios enfrentados durante a concepção. Entretanto, nada é citado sobre federações de nuvens e os problemas de escalonamento enfrentados internamente num provedor de nuvem diferem dos enfrentados em ambientes de federação.

Kumrai *et al.* [49] trata, do escalonamento de tarefas do ponto de vista do intermediador, utilizando *Particle Swarm Optimization* (PSO), que busca imitar um bando de pássaros. Cada elemento da população representa uma possível solução, que ao longo de interações tendem a encontrar o melhor resultado, baseado numa função que avalia quão bom cada elemento da população é. Também analisa uma variação multiobjetivo do *Particle Swarm Optimization*, buscando solução boa tanto para o provedor de nuvem quanto para quem solicita o serviço. O problema do escalonamento apreciado neste artigo possui contexto similar ao que será enfrentado nesta monografia, entretanto, não é tratado o problema criado pelo uso de máquinas compostas por arquiteturas heterogêneas.

Mostageran *et al.* [50] documentam sobre intermediadores, conhecidos como *brokers*, com foco em sua evolução para atenderem a níveis de qualidade de serviços propostos em SLAs. Apresentando definições úteis para termos utilizados na área, como *inter-cloud*, federação de nuvens e *cloud broker*. É importante diferenciar os termos escalonador e intermediário. O intermediário mantém registro de entidades interessadas em um serviço e provedores desse serviço, incluindo busca de provedores e monitoramento das conexões feitas entre os interessados e os provedores. O escalonamento funciona em um escopo menor, que é gerenciar acesso a um recurso escasso buscando otimizar alguma métrica relativa ao seu contexto de uso, como por exemplo acesso de processos ao processador.

Diante dos trabalhos apresentados, nota-se que nenhum deles aborda o escalonamento para federações compostas por arquiteturas heterogêneas. Por isto, este trabalho propõe um escalonador para federações que utilizem equipamentos compostos por arquiteturas heterogêneas, o qual será apresentado na próxima seção.

4.4 Escalonador Proposto para Plataformas de Federação

O escalonador proposto para implementação segue a ideia básica de escalonamento de listas. Dessa forma, haverão três listas: lista de tarefas a serem executadas, lista de CPUs disponíveis e lista de GPUs disponíveis. O uso de listas para escalonamento é uma das abordagens principais para escalonamento, incluindo listas de listas[51], pois é prático associar significados para posições na lista, por exemplo no Linux as posições 0 a 9 são utilizadas para *threads* do sistema e de tempo real, enquanto as posições seguintes são usadas para *threads* de usuário [52].

A lista de tarefas é ordenada por tempo previsto de execução, que é dado por uma estimativa a partir do programa a ser executado e do arquivo de entrada. Essa ordenação será em ordem decrescente de tempo previsto, para que a tarefa que estimada como mais longa tenha alocada para si a melhor VM. A lista de CPUs disponíveis é ordenada em ordem decrescente com base na frequência e no número de núcleos. A lista de GPUs tem sua ordem decrescente determinada pela quantidade de operações em pontos flutuante que consegue realizar por segundo. Dessa forma, o algoritmo proposto é detalhado no Algoritmo 1, e apresentado na forma de fluxo na Figura 4.2.

Algorithm 1 Escalonamento heterogêneo baseado em listas

```
procedure ESCALONAR(listas lTarefas, lCPUs e lGPUs)
    while Existem tarefas que podem ser executadas nos recursos disponíveis? do
        if lTarefas[0] é capaz de executar em GPU then
            if  $T_{previsto}(lGPUs[0]) < T_{previsto}(lCPUs[0])$  then
                Escalone lTarefas[0] para lGPUs[0]
            else
                Escalone lTarefas[0] para lCPUs[0]
        else
            Escalone lTarefas[0] para lCPUs[0]
    Remova a tarefa e o recurso alocado de suas respectivas listas.
    Encerra escalonamento
```

O escalonamento busca associar as tarefas mais longas às máquinas com maior capacidade de processamento, sendo que, para cada tarefa, é analisado se vale a pena alocar a tarefa em uma VM que possua GPU.

Assim sendo, observe que para o algoritmo mostrado na Figura 4.2 ser válido, pressupõe-se que toda tarefa do algoritmo é capaz de executar em CPU. O pressuposto simplifica o primeiro passo do algoritmo, exibido na Figura 4.2, pois se a lista de CPUs não estiver vazia, essa condição é automaticamente satisfeita.

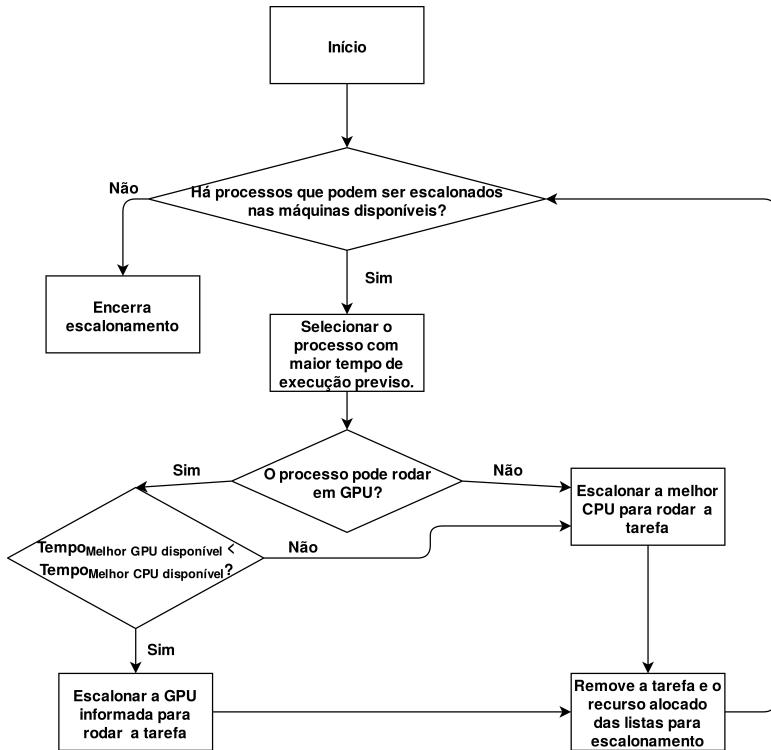


Figura 4.2: Diagrama de Funcionamento do Algoritmo de Escalonamento Proposto.

Dessa forma, sob a ótica do trabalho de Casavant e Kuhl [53], o algoritmo desenvolvido neste trabalho pode ser classificado como global, estático, sub-ótimo e aproximado.

Escalonamento local é o responsável por decidir a execução de processos a *time-slices* de um processador, o escalonador desenvolvido neste trabalho, opera no nível de decidir quais processos serão executados em quais máquinas virtuais, deixando o escalonamento local a cargo do sistema operacional da máquina em questão.

Casavant e Kuhl [53] definem como escalonamento estático, o escalonamento no qual é conhecido durante a inicialização de uma máquina, que *jobs* ela executará. Esta condição é satisfeita, pois um *workflow* precisa ser bem definido para ser executado no BioNimbuz. Vale a pena ressaltar que o escalonamento estático também pode ser denominado escalonamento determinístico.

O escalonador desenvolvido neste trabalho é considerado sub-ótimo e aproximado, porque nem todas as informações sobre os *jobs* de um *workflow* são conhecidas, o BioNimbuz possui o Serviço de Predição para estimar o tempo de execução de um *job*, e por isso este escalonador é considerado aproximado.

O próximo capítulo apresentará os desafios enfrentados na implementação e implantação do escalonador na plataforma BioNimbuz.

Capítulo 5

Implementação

Este capítulo apresenta, inicialmente, os detalhes do módulo de escalonamento do BioNimbuz. Posteriormente, discorre sobre a implementação, mostrando alguns dos desafios encontrados durante o desenvolvimento e como eles foram solucionados. A terceira seção aborda a adição de tarefas que utilizam a GPU no BioNimbuz, e na última parte deste capítulo são descritos os testes que foram feitos e o resultados obtidos a partir do novo escalonador.

5.1 Sistema de Escalonamento do BioNimbuz

O Serviço de Escalonamento é implementado no BioNimbuz como um serviço da Camada de Núcleo (veja a Figura 3.1), de acordo com o subsistema mostrado na Figura 5.1. A interface *services* define métodos para a inicialização, o término de serviços e os métodos para comunicação com o *Zookeeper* [33]. A classe *AbstractBioService* define

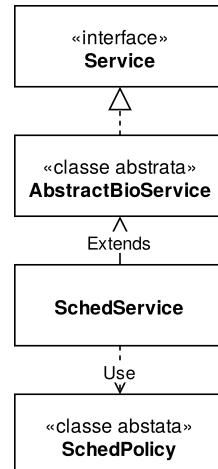


Figura 5.1: Subsistema de Escalonamento do BioNimbuz.

funcionalidades comuns aos serviços do BioNimbuz, em especial, a comunicação entre as máquinas que compõem a plataforma e o padrão de projeto *observer* [?], utilizado para notificação de eventos.

A classe *SchedService* é a responsável por prover o serviço de escalonamento em si, entretanto, para permitir a existência de várias políticas de escalonamento, internamente ela utiliza instâncias da classe abstrata *SchedPolicy*, que implementam cada uma das distintas políticas de escalonamento existentes, conforme mostrado na Figura 5.2.

Essas políticas de escalonamento são implementadas por meio da definição dos seguintes métodos herdados de *SchedPolicy*:

- *schedule*, que realiza o escalonamento inicial propriamente dito;
- *relocate*, o qual realoca um processamento que está em execução;
- *cancelJobEvent*, cujo objetivo é reportar ao escalonador o cancelamento de um *job*;
- *jobDone*, que informa ao escalonador que um *job* terminou.

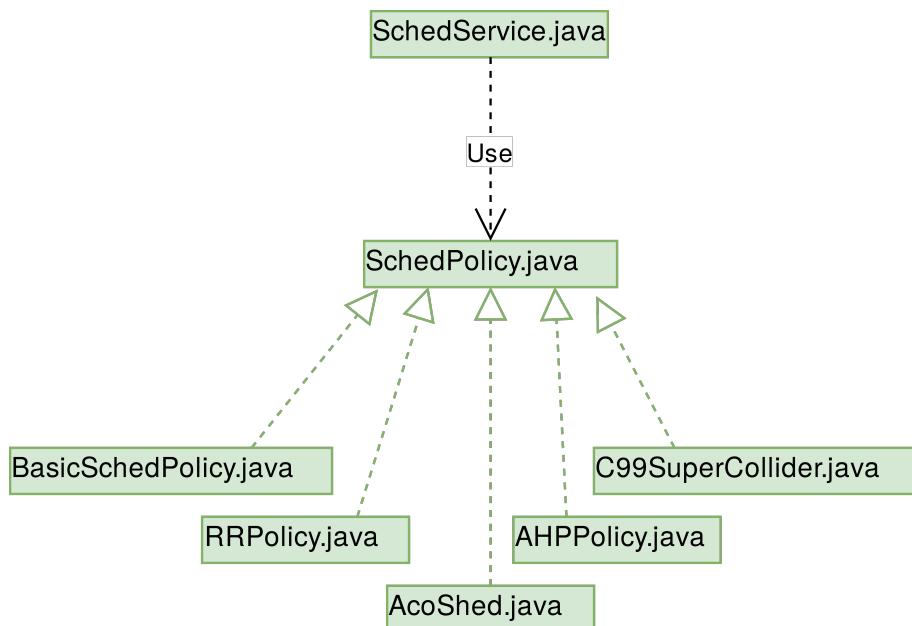


Figura 5.2: Subsistema de Escalonamento do BioNimbuz.

5.2 Implementação do Escalonador Proposto

Por motivos de familiaridade com a linguagem de programação, o escalonador proposto foi implementado em C++ 11. Para ter compatibilidade com os demais componentes do software, classes auxiliares, como a que representa os *Jobs* e as VMs instanciáveis

tiveram equivalentes escritos em C++. Além disso, dois outros desafios surgiram, a saber: como iniciar o escalonador C++, e como ele deve se comunicar com os demais componentes da plataforma. O primeiro desafio foi resolvido por meio de pesquisa na *Application Programming Interface* (API) do Java, utilizando as classes *Process*, *Runtime* e *ProcessBuilder*, os quais permitem executar comandos de terminal, o que possibilitou a execução da parte C++ do escalonador.

O desafio da comunicação do C++ com o Java é mais complexo, pois há várias formas de fazer, e nenhuma que atende ou facilita significativamente a integração. Por exemplo, o uso de classes *wrappers* que usam *handles*, que são objetos cujo objetivo é manipular estruturas que não são nativas da linguagem. Uma outra forma documentada é através do uso do *Java Native Interface* (JNI), que é uma outra forma existente no qual, através do uso de *handles*, é chamado o código C++ num programa Java. Além disso, existem variações deste método utilizando bibliotecas que buscam simplificar o gerenciamento do *handle*. Como as formas pesquisadas para fazer tal comunicação aparentam não chegar a um consenso, decidiu-se então utilizar formas mais genéricas de comunicação entre processos, o qual surgiu a ideia de usar *sockets* para fazer a comunicação.

Socket é uma abstração que sistemas operacionais fazem para permitir que programas tenham acesso à rede. O sistema de portas permite diferenciar em um computador qual dos programas interessados está enviando ou deve receber a mensagem.

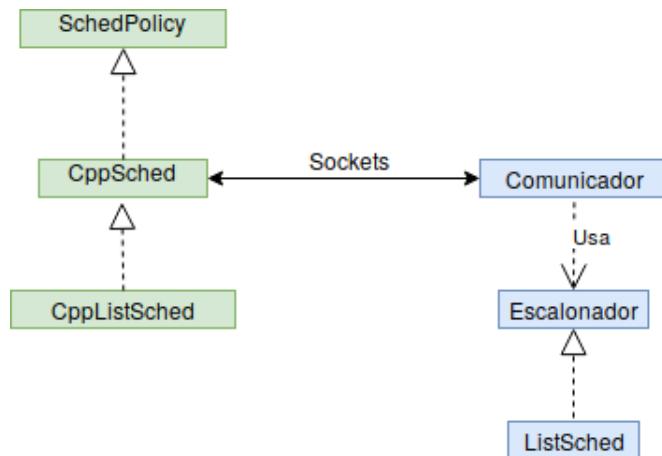


Figura 5.3: Arquitetura de Classes do Escalonador Implementado.

Uma vez decidido o uso de *sockets*, faltava decidir qual seria o protocolo da camada de rede e de transporte. Na camada de rede o principal protocolo existente é o *Internet Protocol* (IP), entretanto, existem duas versões: o IPv4[?] e o IPv6[?], sendo que o último é mais recente e permite um maior número de dispositivos na rede, por isso essa foi a versão utilizada na implementação. Quanto à camada de transporte, os candidatos eram o TCP[?] e o UDP[?]. O TCP provê uma série de garantias ao usuário dos pacotes

que serão transmitidos pela rede com custo de *overhead*, mas como apenas será utilizado para comunicação em *loopback*, o UDP é mais simples e enxuto, sendo o escolhido como protocolo da camada de transporte neste trabalho.

5.2.1 Interação Java C++

Como pode ser observado na Figura 5.3, para a implementação desse sistema de comunicação no BioNimbuZ, desenvolveu-se a classe abstrata *CppSched* que herda de *SchedPolicy*, responsável por fazer a inicialização do programa C++ e pelo *handshake* entre os processos C++ e Java, permitindo que futuros escalonadores em C++ não precisem repetir este processo. Da classe *CppSched* herda o método *GetSchedPolicy*, utilizado para informar qual escalonador C++ deve ser utilizado.

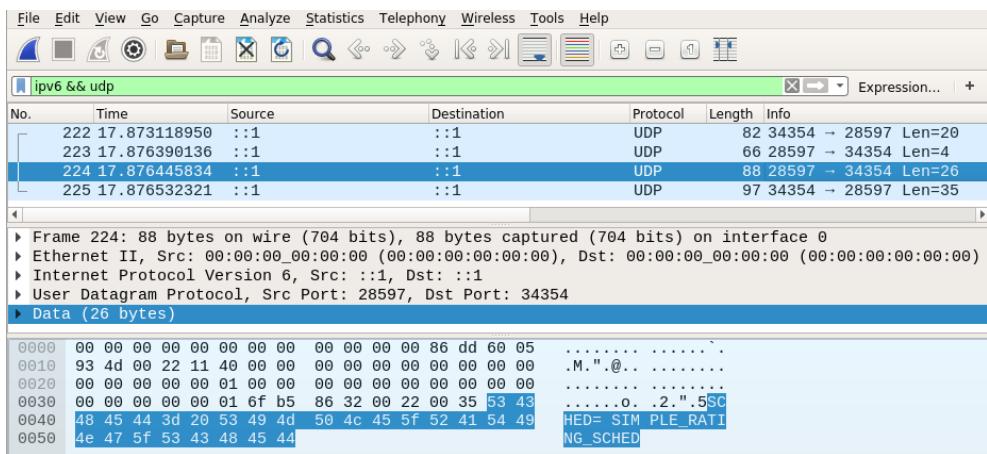


Figura 5.4: Captura do *Handshake* entre a parte Java e C++ do Escalonador.

O *handshake* é feito em três vias, coo pode ser visto na Figura 5.5, sendo que a primeira mensagem é, na verdade, uma inicialização de processo, enviando como argumentos a porta do *socket* que o lado Java disponibilizou para comunicação e um número aleatório de 64 bits, que será utilizado para que o lado Java saiba que a mensagem que ele recebeu veio do lado C++ do escalonador. O lado C++ cria um *socket* e envia o número de 64 bits para a porta informada na inicialização do processo. O lado java recebendo esse número aleatório identifica a porta na qual o lado C++ está, entretanto, o lado C++ não sabe o estado do lado Java, pois ainda não recebeu nenhuma mensagem do mesmo. Para que o lado C++ saiba que a comunicação está inicializada, o lado Java envia um *ack* para o lado C++, confirmando sua existência.

Uma vez que a comunicação entre as partes está inicializada, é necessário definir qual política de escalonamento C++ será utilizada, este passo permite que futuras políticas de escalonamento C++ sejam implementadas reutilizando o processo de interação entre as partes que foi desenvolvido. A definição do escalonador C++ é feita através da troca

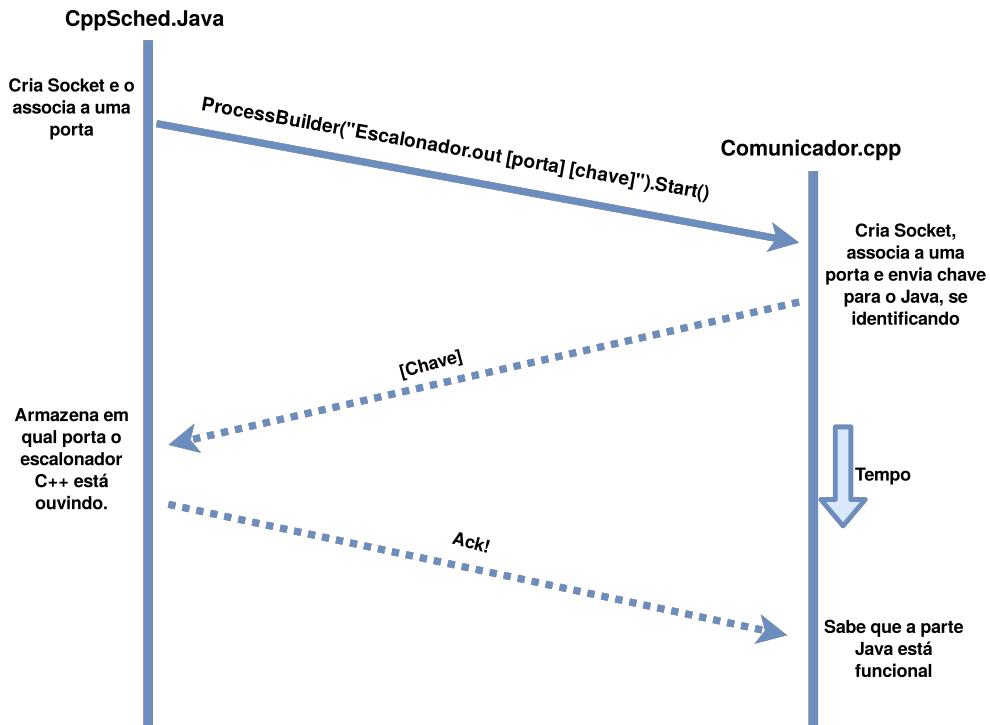


Figura 5.5: Handshake entre a parte Java e C++ do Escalonador.

de duas mensagens, a solicitação e a resposta, como pode ser visto na Figura 5.6. Assim, após a inicialização e escolha do escalonador, o processo C++ se comporta similarmente a um servidor DNS[?], ficando em espera por solicitações de escalonamento, e responde às requisições sem manter estado internamente.

5.3 Testes

Para os testes foi escolhido o programa XMR-Stak[?], um programa de mineração de criptomoedas, que são artefatos digitais desenvolvidos como meio de troca, que utilizam criptografia para prover segurança e integridade às transações[?]. Esse software de mineração, em específico, foi escolhido pelo fato de ser software livre, além de ser multiplataforma e capaz de executar tanto em CPU quanto em GPU.

A mineração de criptomoedas é a atividade de buscar *nouces*, isto é, uma sequência aleatória de bytes, que quando inserido junto de um possível futuro bloco de uma *blockchain*, que é uma lista distribuída para armazenamento de registros, numa função de *hash* criptográfico, gera um *hash* que atenda a algum critério de dificuldade, geralmente um valor específico no início. Quando é encontrado um *digest*, uma saída da função de *hash*, que atende ao critério, tal bloco é adicionado na *blockchain*, e o responsável pela máquina que encontrou a resposta é recompensado em criptomoeda.

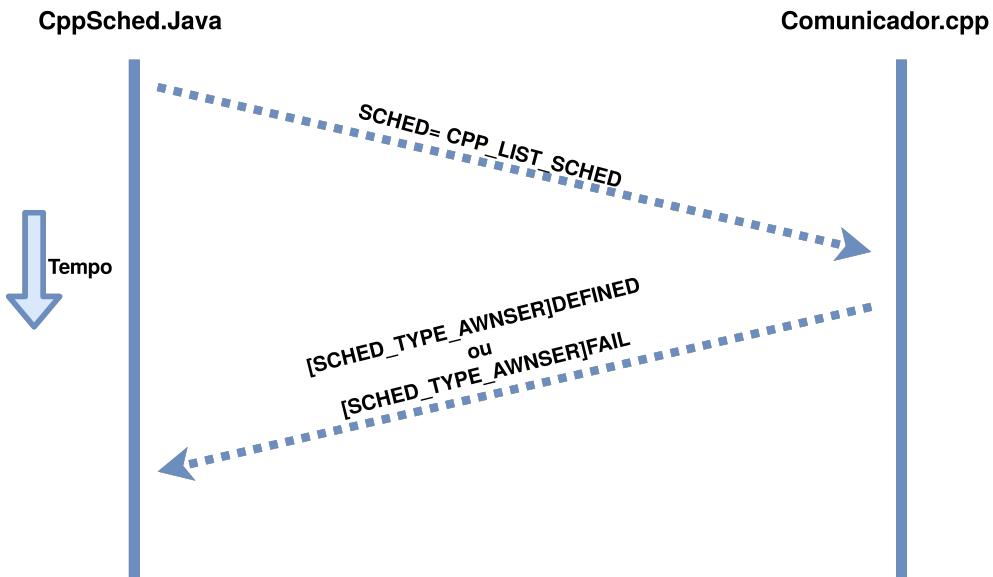


Figura 5.6: Diagrama do processo de seleção do escalonador C++.

Assim, uma vez testado a plataforma na VM, é hora de testar em equipamento real, pois as máquinas virtuais não possuem acesso à GPU, a não ser quando utilizado *GPU Passthrough*, que é uma técnica que permite redirecionar o controle de uma GPU física para uma máquina virtual. Uma das maiores barreiras para o *GPU Passthrough* é que ele requer pelo menos duas placas de vídeos, pois tanto o sistema hóspede quanto o sistema hospedeiro devem ter cada um sua GPU.

Após a plataforma apresentar correto funcionamento, toda a instalação seria clonada num *flash drive* USB capaz de inicializar, para tal o *bootloader*, *software* responsável por determinar como o sistema será inicializado, ser reparado para que seja capaz de inicializar a partir do dispositivo móvel de armazenamento. O sistema precisa ser clonado pois usa pacotes de versões diferentes do sistema operacional Debian[?], porque o XMR-Stak utiliza a biblioteca de processamento paralelo CUDA[?], da Nvidia[?], e essa biblioteca requer o uso da versão 6 do compilador GCC[?], disponível no repositório da versão *stable* do Debian. A biblioteca do CUDA está disponível na versão *unstable* do sistema operacional. O CUDA foi retirado da versão *testing* do Debian por causa dessa dependência da versão 6 do GCC, que é considerada desatualizada [?].

5.4 Implantação no BioNimbuz

O escalonador foi, inicialmente, desenvolvido utilizando apenas o subsistema de escalonamento do BioNimbuz, para que testes fossem feitos com rapidez. Após testes que comprovaram o funcionamento do escalonador, como o da Figura 5.4, utilizou-se uma

máquina virtual para fazer a implantação do escalonador desenvolvido de volta ao BioNimbuZ, pois a instalação do BioNimbuZ requer instalação de programas que podem, mesmo que seja incomum, conflitar ou apresentar erros em interações com o resto do sistema operacional. Um exemplo que ocorreu durante o desenvolvimento foi uma atualização do sistema operacional, Debian[?] Testing[?], que causou erros em tempo de execução no Ambiente Integrado de Desenvolvimento, do inglês *Integrated Development Environment* (IDE) Eclipse[?], relativos à interface gráfica. O mecanismo de *snapshots* providos pelo software de gerenciamento de VMs Virtualbox[?] se revelou realmente útil nessa circunstância, como pode ser visto na Figura 5.7.

Nome	Criado
Pré-atualização	09/04/18 20:04
Sistema atualizado, BioNimbuZ configurado, pré-primeira execução do BioNimbuZ	09/04/18 21:38
eclipse não abre, creio que a atualização do sistema o quebrou	09/04/18 22:17
Eclipse funcionando, mas o BioNimbuZ está com erros em tempo de execução	09/04/18 22:38
BioNimbuZ funcionando	10/04/18 15:14
Antes de rodar o prepare do bionimbuz_web	10/04/18 15:17
Bionimbuz web com problemas em tempo de execução	10/04/18 16:33
EmTeseoClientEstaFunfando	10/04/18 20:48
Cliente bionimbuz rodando, problemas no login	12/04/18 00:14
Antes de instalar o software para Live cd	20/04/18 13:09
AntesDeAtualizarOMySQL	25/04/18 20:55
Utilizando mysql 5.2	25/04/18 21:13
Pré_instatalacao_cuda	26/04/18 14:16

Figura 5.7: Parte dos *Snapshots* existentes na VM de Implantação.

Nos primeiros testes de funcionamento do BioNimbuZ, percebeu-se um erro no qual era possível cadastrar novo usuários, mas não era possível logar na plataforma, apenas aparecendo a mensagem "Erro Interno". Pesquisa nos arquivos de *log* mostraram que o BioNimbuZ não estava conseguido criar a maioria de suas tabelas no Sistema Gerenciador de Banco de Dados (SGBD) MySQL, devido ao tamanho de limite de linha das tabelas, causado pela junção de vários campos para armazenamento de *strings* e o fato do SGBD utilizar 4 bytes para codificar cada caractere. Após pesquisar possíveis soluções, incluindo atualização do MySQL, decidiu-se criar manualmente as tabelas que apresentavam esse desafio, reduzindo o tamanho de alguns campos de texto, solução que resolveu o desafio sem perda aparente de funcionalidade.

Outro desafio enfrentado durante o desenvolvimento foi que *workflows* não eram enviados para execução quando solicitado, os *logs* apenas informavam "Error: NULL", sem maiores informações para solucionar o desafio, que foi identificado como uma inicialização incompleta do BioNimbuZ por causa que a *thread* que inicializava a plataforma ficava à espera de uma mensagem que já havia sido recebida no *socket*, porém incorretamente descartado porque no Java, para se comparar a mensagem recebida com o que se espera, é necessário explicitar a codificação tanto da mensagem recebida quanto do texto esperado.

Mais uma adversidade ocorrida durante a implantação é que quando se seleciona o computador local para executar uma tarefa, esse computador não é listado entre as máquinas

disponíveis para o escalonamento. Assim, se nenhuma outra máquina for escolhida para escalonamento, a requisição que o escalonador recebe não disponibiliza máquinas para escalonar, fazendo com que o escalonamento falhe. O falha que causou esse erro está relacionada com a visibilidade do mapa de nuvens disponíveis na hierarquia de classes, uma vez encontrada a forma correta de acesso a essa informação o desafio foi resolvido.

Com o desafio acima resolvido, o escalonamento estava funcionando corretamente, porém o BioNimbuz não inicializava o *Job* do minerador alegando a falta de um arquivo de entrada de nome vazio e sem informações sobre, por mais que o minerador em si não necessitasse. Após compreender que esse desafio está relativo ao fato de o BioNimbuz trabalhar com *workflows*, para os quais é suposto a existência tanto de entrada, quanto de saída. A solução de contorno para esse desafio foi simplesmente colocar um arquivo qualquer como entrada para o XMR-Stak no BioNimbuz.

Após todos esses desafios terem sido resolvidos, percebeu-se que o *software* modificado de mineração XMR-Stak estava falhando em sua inicialização, descobriu-se então que a criptomoeda utilizada para a mineração teve seu algoritmo de mineração modificado e então foi necessário atualizar a versão modificada do minerador para que voltasse a funcionar corretamente.

Finalmente, após todos os obstáculos enfrentados acima, foi possível, na VM de testes onde o escalonador estava sendo implantado, fazer pelo BioNimbuz a execução do *software* de teste utilizando o escalonador desenvolvido. A próxima etapa então se tornou transplantar a VM utilizada na implantação para máquinas físicas, para que se possa testar corretamente em máquinas físicas com GPU. Para permitir testes em vários equipamentos e minimizar o impacto da instalação do ambiente do BioNimbuz, escolheu-se transplantar a máquina virtual a um *flash drive*. Duas abordagens foram testadas, uma envolveu a criação de uma distribuição *live*, ou seja, uma versão portátil do SO modificada para ter persistência no mesmo *flash drive* em que estava a distribuição, e uma segunda abordagem que se baseia em fazer a instalação completa do sistema no *flash drive*, duplicando a máquina virtual na mídia removível.

Inicialmente buscou-se utilizar a primeira abordagem, utilizando tutoriais existentes na Internet, porém percebeu-se o fato de que a persistência era apenas do diretório do usuário, ou seja, pacotes instalados, removidos ou atualizados da distribuição *live* não persistiam. Tentou-se então, fazer um *script* de preparação da distribuição *live*, que realizaria todas as modificações do sistema necessárias no sistema toda vez em que o Sistema Operacional (SO) fosse inicializado. Infelizmente, percebeu-se que essa abordagem não era viável, pois a distribuição *live* era carregada em memória RAM num disco virtual durante a inicialização, esse disco virtual possui limite de armazenamento, o qual mostrou-se insuficiente pois para a preparação do sistema é necessário fazer atualizações considerá-

veis no sistema por causa que o *software* de teste utiliza pacotes de versões diferentes do Debian.

Utilizando a abordagem de fazer a instalação completa do sistema em um *flash drive* mostrou-se promissora, mas a primeira dificuldade foi conseguir extrair o arquivo de backup do sistema da máquina virtual, pois o arquivo ocupou 11 gigabytes de armazenamento e o sistema de arrastar e soltar do VirtualBox não conseguiu fazer uma transferência tão grande. A solução encontrada foi configurar o adaptador de rede da VM para funcionar em modo bridge e fazer a transferência do sistema convidado, ou seja, o sistema da máquina virtual, para o sistema hospedeiro, que é o sistema que está hospedando a VM, utilizando o programa scp, um programa de transferência de arquivos via rede [?].

Uma vez que a imagem do sistema foi feito, o próximo passo é implantar a imagem gerada um *flash drive*, e então corrigir o *bootloader*, o programa que faz o carregamento inicial do SO no disco para que funcione corretamente em qualquer máquina. Entretanto, após o *flash drive* estar pronto para uso, percebeu-se que o mesmo não conseguia realizar *login*, por motivos ainda não compreendidos. Logo, se tornou necessário uma metodologia de testes que conseguissem mostrar do escalonador mesmo com os desafios apresentados.

5.5 Métricas e Testes

Uma vez preparado o dispositivo portátil de testes, o passo seguinte seria realizar uma série de testes com o objetivo de coletar o máximo de dados possíveis sobre o escalonador desenvolvido no BioNimbuz. O sistema de *log* do BioNimbuz já nos fornece informação sobre a duração da execução dos *jobs* de um *workflows*, além do tempo gasto no subsistema de escalonamento.

O código de instrumentação pode ser inserido na parte C++ do escalonador para calcular o tempo gasto apenas no escalonamento, excluindo tempo gasto na serialização e na desserialização das mensagens entre as partes C++ e Java da plataforma. Dessa forma, se um *software* de captura de pacotes for utilizado, por exemplo o Wireshark[?], é possível verificar o total de tempo gasto pela parte C++ do escalonador, além de calcular o tempo gasto na serialização e desserialização no código C++ da seguinte forma:

$$T_{Csd} = T_{msg} - T_{ci}$$

Onde:

- T_{Csd} é o tempo gasto em serialização e desserialização das mensagens pela parte C++;
- T_{msg} é o tempo entre a mensagem de solicitação de escalonamento e a resposta;

- T_{ci} é o tempo gasto no escalonamento calculado pelo código de instrumentação no programa C++;

Também é possível calcular o tempo gasto na serialização e desserialização da parte Java do escalonador utilizando os dados informados pelo sistema de *logs* do BioNimbuz, e pelo software de captura de pacotes da seguinte forma:

$$T_{Jsd} = T_{log} - T_{msg}$$

Onde:

- T_{Jsd} é o tempo gasto em serialização e desserialização pela parte Java;
- T_{log} é o tempo gasto no escalonamento informado pelo sistema de log do BioNimbuz;
- T_{msg} é o tempo entre a mensagem de solicitação de escalonamento e a resposta;

Com os tempos gastos pelo escalonador calculados, é importante calcular o tempo gasto na conclusão dos *jobs* e do *workflow*, no qual se espera que a funcionalidade desenvolvida neste trabalho revele sua utilidade, mas como não foi possível o teste de toda a plataforma em funcionamento em uma máquina real com GPU. Logo se optou por testar o algoritmo de escalonamento desenvolvido e implantado no BioNimbuz, comparando-o com o utilizado anteriormente, e testar a eficiência do uso GPU executando a versão modificada do XMR-Stak utilizando e sem utilizar a GPU.

Foi realizado uma série de testes comparativos entre o escalonador antigo e o atual para verificar o desempenho do escalonador desenvolvido perante seu antecessor. As descrições das máquinas utilizadas nos testes estão no Apêndice A.

O primeiro teste, cujo os dados coletados estão na Tabela 5.1 consistiu em, utilizando a Máquina A.3, executar o BioNimbuz com o escalonador antigo para escalonar 1 *job* do XMR-Stak em *loopback* e coletar o tempo gasto no escalonamento.

# Da Execução	Tempo De Execução (em segundos)
Execução 1	0,94830
Execução 2	0,9441
Execução 3	0,9562
Execução 4	0,9573

Tabela 5.1: Tempo Gasto No Escalonamento Com o Escalonador Anterior.

O segundo teste foi, a partir Máquina A.3, executar o BioNimbuz com o escalonador desenvolvido para escalonar 1 *job* do XMR-Stak em *loopback* e coletaram-se, o tempo gasto no escalonamento(mensurado como tempo entre a chamada de execução do escalonamento

na parte Java e a resposta), tempo gasto na execução da parte C++ do escalonador (mensurado pela diferença no Wireshark entre a resposta da solicitação de escalonamento e a solicitação), tempo gasto no processamento da parte C++ do escalonador e tempo gasto na serialização e desserialização tanto da parte C++ quanto da parte Java do escalonador desenvolvido. Os dados coletados estão nas Tabelas 5.2, 5.3 e 5.4.

# Da Execução	Tempo Total de Escalonamento (em segundos)
Execução 1	0,9767
Execução 2	0,9907
Execução 3	1,0011
Execução 4	0,9751

Tabela 5.2: Tempo Gasto No Escalonamento Com o Escalonador Desenvolvido.

# Da Execução	Tempo Gasto Na Parte C++	Tempo Gasto Escalonando No C++
Execução 1	0,516241 seg	0,196053 seg
Execução 2	0,589873 seg	0,200189 seg
Execução 3	0,655906 seg	0,212694 seg
Execução 4	0,579292 seg	0,164812 seg

Tabela 5.3: Tempo Gasto No Escalonamento Na Parte C++ Do Escalonador Desenvolvido.

# Da Execução	Tempo De (Des)Serialização C++	Tempo De (Des)Serialização Java
Execução 1	0,320188 seg	0,460459 seg
Execução 2	0,389683 seg	0,400827 seg
Execução 3	0,443212 seg	0,345194 seg
Execução 4	0,414490 seg	0,395808 seg

Tabela 5.4: Tempo Gasto (Des)Serialização Das Partes Java e C++.

O penúltimo teste foi com o uso da Máquina A.1, utilizar o XMR-Stak modificado para calcular 50 mil *hashes*, utilizando apenas a CPU. Os dados coletados estão na Tabela 5.5.

# Da Execução	Tempo De Execução
Execução 1	514,4689 seg
Execução 2	503,8278 seg
Execução 3	494,5142 seg
Execução 4	489,9267 seg

Tabela 5.5: Tempo Para Calcular 50 Mil *hashes*, Utilizando A CPU

Finalmente, o último teste consistiu em, com a Máquina A.1, utilizar o XMR-Stak modificado para calcular 50 mil *hashes*, utilizando apenas a GPU. Os dados coletados estão na Tabela 5.6.

# Da Execução	Tempo De Execução
Execução 1	227,3487 seg
Execução 2	230,4235 seg
Execução 3	215,5216 seg
Execução 4	224,3782 seg

Tabela 5.6: Tempo Para Calcular 50 Mil *hashes*, Utilizando A GPU

5.6 Análise de resultados

Como podemos observar na Tabela 5.7, percebe-se que o tempo de execução de ambos os escalonadores são próximos e suficiente para serem considerados iguais, o que não era esperado, pois se imaginava que o custo de comunicação entre as partes Java e C++ do novo escalonador o tornaria mais lento que o escalonador já existente.

Escalonamento	Tempo médio de execução (em segundos)	Desvio-padrão
Escalonador anterior	0,951475	+/- 0,006344
Escalonador desenvolvido	0,9859	+/- 0,0123

Tabela 5.7: Tempo Médio Gasto No Escalonamento.

Uma análise da distribuição de tempo do escalonador desenvolvido mostrado na Tabela 5.8, como imaginado que o tempo gasto na comunicação é responsável por quatro quintos do tempo no escalonador desenvolvido, revelando-se como o mesmo é rápido e o principal ponto para melhora do mesmo está na comunicação entre as partes Java e C++.

Escalonador proposto	Proporção Tempo Médio(%)
Serialização e desserialização Java	40%
Serialização e desserialização C++	40%
Escalonamento	20%

Tabela 5.8: Distribuição De Tempo Gasto no Escalonador Desenvolvido.

Como não foi possível exportar a plataforma para um dispositivo de armazenamento portátil para testar o ganho de desempenho no uso de GPU, decidiu-se então testar apenas o XMR-Stak modificado em uma máquina física, isso é possível pelo fato do *software* permitir, via argumento de linha de comando, informar se o mesmo deve ser executado em CPU ou GPU. Para realizar este teste foi utilizado uma versão *live* de Debian, que após inicializado, se instala os pacotes necessários para executar o minerador modificado.

XMR-Stak	Tempo Médio De Execução	Desvio Padrão
Utilizando CPU	500,65 seg	+/- 10,87 seg
Utilizando GPU	224,36 seg	+/- 6,41 seg

Tabela 5.9: Tempo Médio E Desvio Padrão Da Execução Do XMR-Stak.

Os resultados expostos na Tabela 5.9 relevam a vantagem da exploração da GPU processamento de determinados *jobs*, sendo executado 50% mais rápido quando comparado ao tempo de execução na CPU, e confirmando as expectativas.

5.7 Considerações Finais

Neste capítulo mostrou-se os desafios e as dificuldades encontradas durante o desenvolvimento do algoritmo apresentado no capítulo anterior e a integração dele no BioNimbuz. Como foi mostrado, o escalonador implementado é executado em bem menos tempo que o anterior, porém a troca de dados entre os processos torna o tempo de escalonamento próximo ao do escalonador utilizado anteriormente no BioNimbuz. Além disso, o testes do *software* XMR-Stak mostraram a utilidade do uso de arquiteturas heterogêneas, restando enorme ganho de performance, reduzindo pela metade o tempo de cálculo de 50 mil *hashes*.

Entretanto, houveram dificuldades na hora de exportar o BioNimbuz para fora da máquina virtual em que houve a implantação, o que impediu testes mais significativos. Porém é possível que esperando a evolução da tecnologia CUDA, para facilitar o processo

de compilação do escalonador no sistema operacional GNU/Linux, e também o desenvolvimento do *driver* das GPUs da marca AMD, para quem tenham suporte apropriado para OpenCL, o XMR-Stak tenha menos dificuldade de ser compilado e executado, permitindo mais fácil integração do mesmo no BioNimbuZ em outras máquinas. O próximo capítulo sugere outras evoluções para o BioNimbuZ e para o escalonador proposto neste trabalho.

Capítulo 6

Conclusão e Trabalhos Futuros

Neste trabalho foi proposto um algoritmo de escalonamento para federações de nuvens compostas por arquiteturas heterogêneas, com o objetivo de tirar proveito de placas de processamento gráfico, que possuem grande capacidade de processamento paralelo. Para tal, foi proposto um algoritmo de escalonamento baseado em listas, que leva em consideração o tempo estimado de execução da tarefa, o benefício de rodar a tarefa em GPU, e a capacidade de processamento tanto da CPU quanto da GPU.

O escalonador proposto foi implementado em C++, e integrado, utilizando *sockets* IPv6, à plataforma de federação de nuvens computacionais BioNimbuZ, desenvolvida em Java. Para a integração foi desenvolvido um mecanismo de inicialização e *handshake* entre as partes C++ e Java do novo escalonador, junto com um sistema de comunicação entre essas partes, que simplifica integração de futuros escalonadores em C++ ao BioNimbuZ. Além disso, também foi desenvolvido um esquema de serialização e desserialização de requisições de escalonamento via rede.

Para testes da plataforma o software de mineração de criptomoedas XMR-Stak foi adaptado e integrado ao BioNimbuZ, aumentando o catálogo de tarefas disponíveis para execução na plataforma. Entretanto, problemas na migração da máquina virtual no qual todo o ecossistema do Escalonador com o BioNimbuZ estava integrado impediram que a plataforma fosse testada integralmente, por outro lado, testes de desempenho dos escalonador desenvolvido mostra que sua performance é equivalente a do escalonador que estava em uso anteriormente, e testes preliminares do *software* de XMR-Stak fora da plataforma mostraram ganho de desempenho em torno de 55%, confirmando a relevância do escalonador desenvolvido.

Como trabalho futuro, após terminar integração do escalonador com o BioNimbuZ, a metodologia de previsão de duração das tarefas pode ser refinada, através do uso de regressão linear, entre outras técnicas. Se o BioNimbuZ obtiver suporte a execução de uma mesma tarefa de forma distribuída, o algoritmo desenvolvido pode ser adaptado

para dar suporte a essa nova funcionalidade. Além disso, o processo de comunicação por *sockets* pode ser otimizado, pois atualmente as requisições são transmitidas em forma de texto legível, para facilitar a depuração da comunicação. Como também pode-se buscar evoluir os demais escalonadores existentes no BioNimbuZ para que eles possam tratar o caso das nuvens compostas por arquiteturas heterogêneas, além de adição de novas tarefas ao BioNimbuZ que explorem essa tecnologia.

Referências

- [1] GNU: *The GNU Operating System and the Free Software Movement*. <https://www.gnu.org/>, 2017. [Online; acessado em 23 de Novembro de 2017]. xi
- [2] Google: *Quem somos / google*. 1
- [3] Demchenko, Y., M. X. Makkes, R. Strijkers e C. de Laat: *Intercloud architecture for interoperability and integration*. Em *4th IEEE International Conference on Cloud Computing Technology and Science Proceedings*, páginas 666–674, Dec 2012. 1, 7
- [4] Buyya, Rajkumar, Rajiv Ranjan e Rodrigo N. Calheiros: *Intercloud: Utility-oriented federation of cloud computing environments for scaling of application services*. Em *Proceedings of the 10th International Conference on Algorithms and Architectures for Parallel Processing - Volume Part I*, ICA3PP’10, páginas 13–31, Berlin, Heidelberg, 2010. Springer-Verlag, ISBN 3-642-13118-2, 978-3-642-13118-9. https://doi.org/10.1007/978-3-642-13119-6_2. 1, 6
- [5] Bacelar, Breno Rodrigues Moura; Deric Lima: *Política para armazenamento de arquivos no zoonimbus*. Biblioteca de monografias da UnB, 2013. Monografia (Licenciatura em Ciência da Computação). 2, 7, 9
- [6] Hugo Saldanha, Edward de Oliveira Ribeiro, Maristela Holanda Aleteia PF Araujo Genaína Nunes Rodrigues Maria Emilia Telles Walter Jo btxfnamespacelong ao Carlos Setubal Alberto MR Dávila: *A cloud architecture for bioinformatics workflows*. CLOSER, 11:477, 2011. 2, 7
- [7] Lima, D., B. Moura, G. Oliveira, E. Ribeiro, A. Araujo, M. Holanda, R. Togawa e M. E. Walter: *A storage policy for a hybrid federated cloud platform: A case study for bioinformatics*. Em *2014 14th IEEE/ACM International Symposium on Cluster, Cloud and Grid Computing*, páginas 738–747, May 2014. 2, 7, 9, 10
- [8] Saldanha, Hugo, Edward Ribeiro, Carlos Borges, Aletéia Araújo, Ricardo Gallon, Maristela Holanda, Maria Emília Walter, Roberto Togawa e Jo btxfnamespacelong ao Carlos Setubal: *Towards a hybrid federated cloud platform to efficiently execute bioinformatics workflows*. Em Pérez-Sánchez, Horacio (editor): *Bioinformatics*, capítulo 05. InTech, Rijeka, 2012. <http://dx.doi.org/10.5772/50289>. 2
- [9] Oliveira, G. S. S. de, E. Ribeiro, D. A. Ferreira, A. P. F. Araújo, M. T. Holanda e M. E. M. T. Walter: *Acosched: A scheduling algorithm in a federated cloud infras-*

- tructure for bioinformatics applications.* Em *2013 IEEE International Conference on Bioinformatics and Biomedicine*, páginas 8–14, Dec 2013. 2, 7, 9, 14
- [10] Barreiros Júnior, Willian de Oliveira: *Escalonador de tarefas para o plataforma de nuvens federadas bionimbuz usando beam search iterativo multiobjetivo.* <http://bdm.unb.br/handle/10483/13146>, 2016. Monografia (Bacahrelado em Engenharia da Computação). 2, 7, 9, 10, 15
- [11] Borges, C. A. L., H. V. Saldanha, E. Ribeiro, M. T. Holanda, A. P. F. Araujo e M. E. M. T. Walter: *Task scheduling in a federated cloud infrastructure for bioinformatics applications.* Em *Proceedings of the 2nd International Conference on Cloud Computing and Services Science - Volume 1: CLOSER,,* páginas 114–120. INSTICC, SciTePress, 2012, ISBN 978-989-8565-05-1. 2, 7, 9
- [12] Saldanha, Hugo Vasconcelos: *Bionimbus: uma arquitetura de federação de nuvens computacionais híbrida para a execução de workflows de bioinformática.* Tese de Mestrado, Universidade de Brasília, 2013. <http://repositorio.unb.br/handle/10482/12046>. 2, 7, 9, 10
- [13] Vaquero, Luis M., Luis Rodero-Merino, Juan Caceres e Maik Lindner: *A break in the clouds: Towards a cloud definition.* SIGCOMM Comput. Commun. Rev., 39(1):50–55, dezembro 2008, ISSN 0146-4833. <http://doi.acm.org/10.1145/1496100.1496100>. 3, 4
- [14] Mell, Peter e Timothy Grance: *The nist definition of cloud computing.* Relatório Técnico 10.6028/NIST.SP.800-145, National Institute of Standards and Technology: U.S. Department of Commerce, Sep 2011. <http://nvlpubs.nist.gov/nistpubs/Legacy/SP/nistspecialpublication800-145.pdf>. 3, 4, 5
- [15] Coutinho, Emanuel Ferreira, Flávio Rubens de Carvalho Sousa, Paulo Antonio Leal Rego, Daniel Gonçalves Gomes e José Neuman de Souza: *Elasticity in cloud computing: a survey.* annals of telecommunications - annales des télécommunications, 70(7):289–309, Aug 2015, ISSN 1958-9395. <https://doi.org/10.1007/s12243-014-0450-7>. 4
- [16] Oracle: *Infrastructure as a Service / Oracle Cloud.* <https://cloud.oracle.com/iaas>, 2018. [Online; acessado em 23 de Fevereiro de 2018]. 4
- [17] Services, Amazon Web: *Amazon EC2 Instance Types - Amazon Web Services(AWS).* <https://aws.amazon.com/ec2/instance-types/>, 2017. [Online; acessado em 21 de Novembro de 2017]. 5
- [18] Platform, Google Cloud: *Google Compute Engine Pricing / Compute Engine Documentation / Google Cloud Platform.* <https://cloud.google.com/compute/pricing>, 2017. [Online; acessado em 21 de Novembro de 2017]. 5
- [19] Google: *Google Docs - create and edit documents online, for free.* <https://www.google.com/docs/about/>, 2018. [Online; acessado em 6 de Fevereiro de 2018]. 5

- [20] Celesti, A., F. Tusa, M. Villari e A. Puliafito: *How to enhance cloud architectures to enable cross-federation*. Em *2010 IEEE 3rd International Conference on Cloud Computing*, páginas 337–345, July 2010. 5
- [21] Chen, H., B. An, D. Niyato, Y. C. Soh e C. Miao: *Workload factoring and resource sharing via joint vertical and horizontal cloud federation networks*. IEEE Journal on Selected Areas in Communications, 35(3):557–570, March 2017, ISSN 0733-8716. 5, 6
- [22] Calheiros, Rodrigo N., Adel Nadjaran Toosi, Christian Vecchiola e Rajkumar Buyya: *A coordinator for scaling elastic applications across multiple clouds*. Future Gener. Comput. Syst., 28(8):1350–1362, outubro 2012, ISSN 0167-739X. <http://dx.doi.org/10.1016/j.future.2012.03.010>. 6
- [23] Garg, Saurabh Kumar, Christian Vecchiola e Rajkumar Buyya: *Mandi: a market exchange for trading utility and cloud computing services*. The Journal of Supercomputing, 64(3):1153–1174, Jun 2013, ISSN 1573-0484. <https://doi.org/10.1007/s11227-011-0568-6>. 6
- [24] Venugopal, S., X. Chu e R. Buyya: *A negotiation mechanism for advance resource reservations using the alternate offers protocol*. Em *2008 16th International Workshop on Quality of Service*, páginas 40–49, June 2008. 6
- [25] Toosi, A. N., R. N. Calheiros, R. K. Thulasiram e R. Buyya: *Resource provisioning policies to increase iaas provider’s profit in a federated cloud environment*. Em *2011 IEEE International Conference on High Performance Computing and Communications*, páginas 279–287, Sept 2011. 6
- [26] Margheri, A., M. S. Ferdous, M. Yang e V. Sassone: *A distributed infrastructure for democratic cloud federations*. Em *2017 IEEE 10th International Conference on Cloud Computing (CLOUD)*, páginas 688–691, June 2017. 6
- [27] Alansari, S., F. Paci e V. Sassone: *A distributed access control system for cloud federations*. Em *2017 IEEE 37th International Conference on Distributed Computing Systems (ICDCS)*, páginas 2131–2136, June 2017. 6
- [28] Gallico, D., M. Biancani, C. Blanchet, M. Bedri, J. F. Gibrat, J. I. A. Baranda, D. Hacker e M. Kourkouli: *Cyclone: A multi-cloud federation platform for complex bioinformatics and energy applications (short paper)*. Em *2016 5th IEEE International Conference on Cloud Networking (Cloudnet)*, páginas 146–149, Oct 2016. 7
- [29] Jrad, Foued, Jie Tao e Achim Streit: *A broker-based framework for multi-cloud workflows*. Em *Proceedings of the 2013 International Workshop on Multi-cloud Applications and Federated Clouds*, MultiCloud ’13, páginas 61–68, New York, NY, USA, 2013. ACM, ISBN 978-1-4503-2050-4. <http://doi.acm.org/10.1145/2462326.2462339>. 7
- [30] Mashayekhy, L., M. M. Nejad e D. Grosu: *Cloud federations in the sky: Formation game and mechanism*. IEEE Transactions on Cloud Computing, 3(1):14–27, Jan 2015, ISSN 2168-7161. 7

- [31] A. Marosi, G. Kecskemeti, A. Kertesz P. Kacsuk: *Fcm: an architecture for integrating iaas cloud systems.* The Second International Conference on Cloud Computing, GRIDs, and Virtualization, páginas 7–12, 2011. http://www.intercloudtestbed.org/uploads/2/1/3/9/21396364/fcm_-_an_architecture_for_integrating_iaas_cloud_systems.pdf. 7
- [32] Zant, B. El e M. Gagnaire: *New pricing policies for federated cloud.* Em *2014 6th International Conference on New Technologies, Mobility and Security (NTMS)*, páginas 1–6, March 2014. 7
- [33] Vergara, Guilherme Fay: *Arquitetura de um controlador de elasticidade para nuvens federadas.* 2017. 9
- [34] Foundation, The Apache Software: *Apache Zookeeper - Home.* <https://zookeeper.apache.org/>, 2017. [Online; acessado em 24 de Novembro de 2017]. 9, 10, 23
- [35] Lin, Kwei Jay e J. D. Gannon: *Atomic remote procedure call.* IEEE Transactions on Software Engineering, SE-11(10):1126–1135, Oct 1985, ISSN 0098-5589. 10
- [36] Foundation, The Apache Software: *Welcome to Apache Avro!* <https://avro.apache.org/>, 2017. [Online; acessado em 24 de Novembro de 2017]. 10
- [37] Azevedo, Diego Rodrigues; Freitas Júnior, Tarcísio Batista de: *Biocirrus : uma nova política de armazenamento para a plataforma bionimbuz de nuvem federada.* <http://bdm.unb.br/handle/10483/13199>, 2015. Online, acessado em 28 de Março de 2018. 10
- [38] Ramos, Vinícius de Almeida: *Um sistema gerenciador de workflows científicos para a plataforma de nuvens federadas bionimbuz.* <http://bdm.unb.br/handle/10483/13145>, 2016. Online, acessado em 28 de Março de 2018. 10
- [39] Santos, Lucas Facundo Neiva: *Novo serviço de armazenamento na plataforma de nuvem federada bionimbuz.* <http://bdm.unb.br/handle/10483/16668>, 2016. Online, acessado em 28 de Março de 2018. 10
- [40] Foundation, The Apache Software: *Welcome to Apache Software Foundation!* <https://apache.org/>, 2017. [Online; acessado em 15 de Dezembro de 2017]. 10
- [41] Dorigo, Marco e Christian Blum: *Ant colony optimization theory: A survey.* Theoretical Computer Science, 344(2):243 – 278, 2005, ISSN 0304-3975. <http://www.sciencedirect.com/science/article/pii/S0304397505003798>. 14
- [42] Ullman, J.D.: *Np-complete scheduling problems.* Journal of Computer and System Sciences, 10(3):384 – 393, 1975, ISSN 0022-0000. <http://www.sciencedirect.com/science/article/pii/S0022000075800080>. 16
- [43] AMD: *RyzenTM ThreadripperTM Processors .* <http://www.amd.com/en/products/ryzen-threadripper>, 2017. [Online; acessado em 21 de Novembro de 2017]. 17

- [44] Dimitrov, Martin, Mike Mantor e Huiyang Zhou: *Understanding software approaches for gpgpu reliability*. Em *Proceedings of 2Nd Workshop on General Purpose Processing on Graphics Processing Units*, GPGPU-2, páginas 94–104, New York, NY, USA, 2009. ACM, ISBN 978-1-60558-517-8. <http://doi.acm.org/10.1145/1513895.1513907>. 19
- [45] Yang, Yi, Ping Xiang, Jingfei Kong e Huiyang Zhou: *A gpgpu compiler for memory optimization and parallelism management*. SIGPLAN Not., 45(6):86–97, junho 2010, ISSN 0362-1340. <http://doi.acm.org/10.1145/1809028.1806606>. 19
- [46] Gouasmi, T., W. Louati e A. H. Kacem: *Cost-efficient distributed mapreduce job scheduling across cloud federation*. Em *2017 IEEE International Conference on Services Computing (SCC)*, páginas 289–296, June 2017. 19
- [47] Dean, Jeffrey e Sanjay Ghemawat: *Mapreduce: Simplified data processing on large clusters*. Commun. ACM, 51(1):107–113, janeiro 2008, ISSN 0001-0782. <http://doi.acm.org/10.1145/1327452.1327492>. 19
- [48] Nguyen, P. D. e N. Thoai: *Drbcf: A differentiated ratio-based approach to job scheduling in cloud federation*. Em *2016 10th International Conference on Complex, Intelligent, and Software Intensive Systems (CISIS)*, páginas 31–37, July 2016. 20
- [49] Jennings, Brendan e Rolf Stadler: *Resource management in clouds: Survey and research challenges*. J. Netw. Syst. Manage., 23(3):567–619, julho 2015, ISSN 1064-7570. <http://dx.doi.org/10.1007/s10922-014-9307-7>. 20
- [50] Kumrai, T., K. Ota, M. Dong, J. Kishigami e D. K. Sung: *Multiobjective optimization in cloud brokering systems for connected internet of things*. IEEE Internet of Things Journal, 4(2):404–413, April 2017, ISSN 2327-4662. 20
- [51] Mostajeran, E., B. I. Ismail, M. F. Khalid e H. Ong: *A survey on sla-based brokering for inter-cloud computing*. Em *2015 Second International Conference on Computing Technology and Information Management (ICCTIM)*, páginas 25–31, April 2015. 20
- [52] Arpaci-Dusseau, Remzi H.; Arpaci-Dusseau, Andrea C.: *Scheduling: The multi-level feedback queue*. <http://pages.cs.wisc.edu/~remzi/OSTEP/cpu-sched-mlfq.pdf>, 2014. Online, acessado em 20 de Março de 2018. 21
- [53] cn.opensolaris.org: *Comparasion between solaris, linux and freebsd kernels*. https://web.archive.org/web/20080807124435/http://cn.opensolaris.org/files/solaris_linux_bsd_cmp.pdf, 2008. Online, acessado em 20 de Março de 2018. 21
- [54] Casavant, T. L. e J. G. Kuhl: *A taxonomy of scheduling in general-purpose distributed computing systems*. IEEE Transactions on Software Engineering, 14(2):141–154, Feb 1988, ISSN 0098-5589. 22

Apêndice A

Especificação Das Máquinas Utilizadas Nos Testes

- Máquina A.1:

- CPU Intel i3 3220
- GPU Nvidia 750 Ti 1 Gb VRAM
- Quantidade de memória RAM: 8Gb (2x4Gb) 1333Mhz DDR3
- Sistema Operacional: Debian Testing
- Placa mãe: Gigabyte B75M D3H BIOS versão F15
- HD 80Gb usando para montagem da raiz do sistema de arquivos
- HD 1Tb usando para montagem do */home*
- GCC 7.3
- Linux 4.16
- Driver Nvidia versão 390.67
- CUDA versão 9.1.85
- GCC 6.4.0 utilizado para compilação do XMR-Stak

- Máquina A.2:

- Notebook modelo Vaio VPCSB35FB
- CPU Intel i5 2430M
- GPU Intel HD Graphics 3000
- GPU AMD Radeon 6470M
- Quantidade de memória RAM: 12Gb (4+8 Gb) 1333Mhz DDR3

- Sistema Operacional: Debian Testing
 - HD 500Gb usando para montagem da raiz do sistema de arquivos
 - GCC 7.2
 - Linux 4.13
- Maquina A.3(Máquina virtual sobre a Máquina 2):
 - Software utilizado para virtualização: VirtualBox
 - Notebook modelo Vaio VPCSB35FB
 - CPU: 2 cores do hospedeiro, limitados em 60% de cada
 - Memória de vídeo: 16 Mb
 - GPU AMD Radeon 6470M
 - Quantidade de memória RAM:6500Mb DDR3
 - Sistema Operacional: Debian Misto
 - HD 30 Gb usando para montagem da raiz do sistema de arquivos
 - GCC 7.2
 - Linux 4.13
 - Driver Nvidia versão 390.48
 - CUDA versão 9.1.85
 - GCC 6.4.0 utilizado para compilação do XMR-Stak