Universidade Federal de Campina Grande
Centro de Engenharia Elétrica e Informática
Coordenação de Pós-Graduação em Ciência da Computação

Melhorando o processamento de dados com Hadoop na nuvem através do uso transparente de instâncias oportunistas com qualidade de serviço

Telles Mota Vidal Nóbrega

Dissertação submetida à Coordenação do Curso de Pós-Graduação em Ciência da Computação da Universidade Federal de Campina Grande - Campus I como parte dos requisitos necessários para obtenção do grau de Mestre em Ciência da Computação.

Área de Concentração: Ciência da Computação
Linha de Pesquisa: Computação na Nuvem

Andrey Elísio Monteiro Brito
(Orientador)

Campina Grande, Paraíba, Brasil

Resumo

Nuvens computacionais oferecem para usuários a facilidade de aquisição de recursos por meio da internet de forma rápida, barata e segura. Entretanto, grande parte das nuvens se mantém ociosa devido à reserva de recursos. Visando a aumentar a utilização da nuvem, provedores de nuvem criaram um modelo de instâncias que reusam recursos ociosos, conhecidas como instâncias oportunistas. Essas instâncias são mais baratas que as instâncias de recursos dedicados, porém voláteis, podendo ser preemptadas do usuário a qualquer momento, o que as torna inadequadas para alguns tipos de aplicação. Processamento de dados, seguindo a tendência de outras aplicações, tem sido migrado para nuvem e pode ser beneficiado por instâncias oportunistas, devido à sua natureza tolerante à falha, resultando na criação de *clusters* a um custo menor comparado à instâncias com recursos dedicados.

Este trabalho propõe a utilização dos recursos ociosos para a criação de um outro modelo de instâncias oportunistas. Esse modelo visa a criação de instâncias oportunistas com qualidade de serviço, que são instâncias criadas baseadas em uma predição do estado da nuvem. A predição é realizada a partir de dados históricos de utilização de recursos como CPU e memória RAM e assim diminuindo o risco de perder instâncias antes do fim do processamento. Ainda com a existência do preditor, o risco de perda de uma máquina existe e para esse caso propomos a utilização de migração viva, movendo a máquina virtual de servidor, evitando assim a destruição da mesma.

Com nossa abordagem, utilizando apenas duas instâncias oportunistas durante os experimentos, obtivemos uma diminuição no tempo de processamento de dados de 10% em um cluster com 2 *workers* e 1 *master*. Além disso, ao utilizar a migração, temos uma melhora de aproximadamente 70% no tempo de processamento em comparação com os casos onde uma instância é perdida.

Palavras-chaves: computação na nuvem, processamento de dados, Hadoop, instâncias oportunistas.

Abstract

Cloud computing offers the users the ease of resources acquisition through the Internet in a fast, cheap and safe manner. However, these clouds have a lot of idle resources due to resource reservation. Aiming to increase resources usage, cloud providers have created an instance model that uses these idle resources, known as opportunistic instances. These instances are cheaper than the dedicated resources instances, but are volatile and can be destroyed at any time, which makes them unsuitable for some types of application. Data processing, following the trend of other applications, have been migrated to the cloud and can be benefited by the use opportunistic instances, due to its fault tolerant nature, resulting in the creation of clusters at a lower cost compared to instances with dedicated resources.

In this work, we propose the use of idle resources to create another model of opportunistic instances. This model aims to create opportunistic instances with quality of service, which are created instances based on a prediction of the state of the cloud. The prediction is made from historical data of resource usage such as CPU and RAM, thus reducing the risk of losing instances before the end of the processing. Even with the existence of a predictor, the risk of losing a machine still exists, and for this case we propose the use of live migration, moving the virtual machine to a different server, thus avoiding the its destruction.

With our approach, using only two opportunistic instances during the experiments, we found a decrease in 10% in the data processing time in a cluster with 2 workers and 1 master. Furthermore, when using the migration, we have an improvement of approximately 70% in processing time compared with the case where one instance is lost.

Keywords: cloud computing, data processing, Hadoop, opportunistic instances.

Agradecimentos

Em primeiro lugar gostaria de agradecer a Deus por Ele ter me proporcionado a capacidade de estar aqui escrevendo este trabalho.

Em segundo lugar minha esposa, Rebeca, por todo o apoio e compreensão durante estes anos me dedicando para a completude deste trabalho, por ela entender que cada sacrifício feito até aqui valeu a pena. Gostaria de agradecer também meus pais, Rômulo e Rosemary, por tudo que fizeram por mim durante toda a minha a vida, pelas oportunidades que me deram de estudar e me formar, pelo apoio incondicional, pela saudade nas viagens longas, pelo cuidado e dedicação que tiveram em tudo.

Em terceiro lugar, quero agradecer meus tios, Marinaldo e Rosângela (tio Naldo e tia Rosa), por terem me recebido na sua casa de braços abertos e nunca terem medido esforços para me ajudar em tudo que precisei durante os 7 anos que vivemos juntos, nunca me esquecerei do que vocês fizeram por mim.

Quero agradecer ao professor Andrey, por ter me orientado tanto neste trabalho como em outras questões, por ter acreditado em mim e investido no meu crescimento profissional. Ao professor Francisco Brasileiro (Fubica) que me recebeu no Laboratório de Sistemas Distribuídos (LSD) ainda muito novo, onde pude aprender praticamente tudo que sei e a professora Raquel que foi a mediadora para eu tivesse a oportunidade de trabalhar durante 7 anos no laboratório.

A meus irmãos, Thalysor e Thiago, e minha cunhada Anne, tenho que agradecer pelo apoio, pela compreensão e pedir desculpa por muitas vezes estar ausente nas suas vidas devido a este trabalho.

Minhas primas, Jacy e Jana, obrigado por serem como irmãs para mim e sempre me motivarem a ser sempre melhor. Minhas gurias e meu guri, Luíza, Izadora, Isabela e Victor vocês tem sido muito mais que primos/sobrinhos, são fontes de inspiração e desejo de me tornar cada mais capaz. Marcelo, Patrícia, Acácio e Victor obrigado pela companhia de vocês durante todo este tempo e por acreditarem em mim.

Por fim, gostaria de encerrar com um versículo que se encontra em 1 Tessalonicenses 5:18 - Em tudo dai graças, porque esta é a vontade de Deus em Cristo Jesus para convosco.

Conteúdo

Lista de Símbolos

API - *Application Programming Interface*

AWS - *Amazon Web Services*

CLI - *Command Line Interface*

DPaaS - *Data Processing as a Service*

EDP - *Elastic Data Processing*

EMR - *Elastic MapReduce*

IaaS - *Infrastructure as a Service*

PaaS - *Platform as a Service*

SaaS - *Software as a Service*

SLA - *Service Level Agreement*

SLO - *Service Level Objective*

VCPU - *Virtual CPU*

VM - *Virtual Machine*

OS - *Operating System*

Lista de Figuras

Lista de Tabelas

Lista de Códigos Fonte

Capítulo 1

Introdução

Neste capítulo introduzimos os conceitos fundamentais da dissertação. Contextualizando o cenário onde esta proposta é aplicada, bem como o problema que buscamos solucionar. Aqui discutimos também sobre a contribuição e relevância do trabalho, e a metodologia utilizada nessa pesquisa. Por fim, é descrito como está organizado o restante do trabalho.

1.1 Contextualização

Nesta seção, buscamos contextualizar os conceitos fundamentais para a pesquisa. Na Seção 1.1.1 discutimos sobre computação na nuvem e seus conceitos. Na Seção 1.1.2 falamos sobre o processamento de dados, seus conceitos e sua tendência de migração para nuvem. Na Seção 1.1.3 descrevemos o problema da ociosidade da nuvem e falamos sobre algumas maneiras de solucionar esse problema. Por fim, na Seção 1.1.4 falamos sobre o OpenStack e o Sahara, que foram utilizadas na nossa pesquisa e na solução proposta.

1.1.1 Computação na nuvem

A computação na nuvem tornou-se uma realidade nos últimos anos, passando a ser uma das principais fontes de recursos computacionais para empresas e desenvolvimento de pesquisas em geral [60]. Isso se deve à sua flexibilidade e escalabilidade na provisão de recursos de infraestrutura computacional, além do baixo custo quando comparado ao custo de aquisição e manutenção de um *data center*. Podemos classificar computação na nuvem em (i) nuvens

públicas, que têm como alvo o público geral e possibilitam a obtenção de recursos segundo o modelo ”pague conforme uso”(do inglês *pay-as-you-go*), (ii) nuvens privadas, que visam a atender demandas internas de empresas e universidades, por exemplo, e, finalmente, (iii) nuvens híbridas, que são nuvens privadas que fazem uso de recursos de nuvens públicas quando necessário. Cada ambiente desses pode oferecer diferentes categorias de serviços que se diferem nas responsabilidades do provedor e do usuário. Na categoria infraestrutura como serviço (*IaaS*, do inglês *Infrastructure as a Service*), o provedor é responsável por oferecer ao usuário recursos computacionais na forma de máquinas virtuais (*VM*, do inglês *Virtual Machine*), e o usuário é responsável pela escolha de sistema operacional, dados e configuração da aplicação. Um exemplo é o serviço oferecido pela *Amazon EC2* [1]. Já na categoria plataforma como serviço (*PaaS*, do inglês *Plataform as a Service*), como no *Amazon Elastic MapReduce* [3], o provedor oferece uma infraestrutura pré-configurada e o usuário é responsável apenas pela manipulação da aplicação e dos dados. O usuário pode requisitar nessa categoria, por exemplo, um ambiente pronto para desenvolvimento. Por último, temos a categoria *software* como serviço (*SaaS*, do inglês *Software as a Service*), onde o provedor é responsável pela gerência de todos os recursos, e apenas uma interface *web* é disposta para usuários interagirem com a aplicação, Google Apps é um exemplo dessa categoria de serviços. A Figura 1.1 mostra em detalhes os modelos e a divisão de responsabilidade no provisionamento de serviços.

1.1.2 Processamento de dados na nuvem

Devido ao aumento e facilidade de conectividade entre pessoas e dispositivos, grande massas de dados, são gerados de forma muito rápida. Em forma bruta esses dados não possuem valor significativo, porém, quando processados, podem conter informações valiosas para tomada de decisões. Dentro deste contexto, surgiu *Big Data*, um conjunto de ferramentas e tecnologias para o processamento de dados em grande escala.

Existem duas principais categorias de processamento de dados: processamento em lote (do inglês, *batch processing*) e processamento em tempo real (do inglês, *stream processing*). Processamento em lote consiste em processar grandes blocos de dados (lotes) de forma distribuída, de modo que esse processamento é muito mais rápido do que seria com o processamento comum, sequencial. O principal paradigma utilizado para processamento em lote hoje

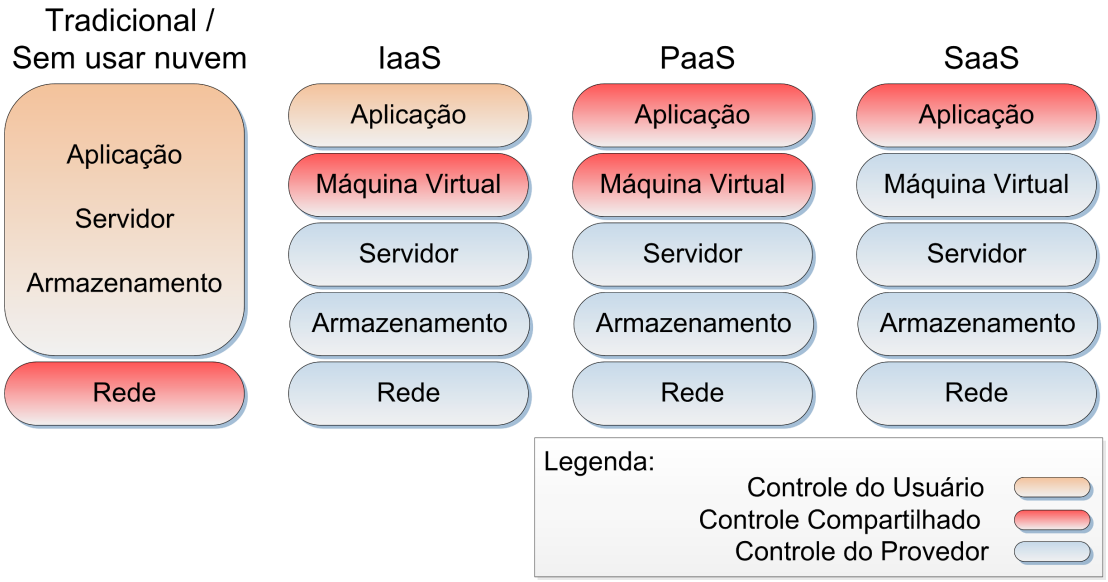

Figura 1.1: Categorias de computação na nuvem e a separação de responsabilidades no provisionamento dos mesmos.[1]

é o *MapReduce*, tendo o Hadoop como sua implementação mais comum e estável. Esse paradigma consiste em duas fases: *Map* e *Reduce*. Na fase *map*, é feito o primeiro processamento e os dados são filtrados e ordenados, enquanto na fase *reduce*, é realizada a sumarização dos dados. A Figura 1.2 mostra como funciona o paradigma em um exemplo comum de contar palavras. Nela observa-se a divisão do texto e filtragem dos dados na fase *Map*, e na fase *Reduce* podemos ver o agrupamento dos dados previamente filtrados e assim chegando ao resultado. Já o processamento em tempo real se caracteriza pela entrada contínua de dados, sem formação de lotes, e o processamento instantâneo dos mesmos.

Seguindo a tendência de outros serviços e pelas características favoráveis como elasticidade, rapidez no provisionamento e escalabilidade, o processamento de dados também seguiu na direção da utilização de nuvens computacionais. A priori, recursos adquiridos eram utilizados para configuração de *clusters* (conjuntos de máquinas virtuais) dedicadas à aplicação Hadoop, sendo de responsabilidade do usuário configurar comunicação entre cada nó do *cluster*, instalar o Hadoop em cada um, e todos os outros requisitos necessários para o funcionamento correto da aplicação, como armazenamento e questões de acesso entre os nós. Para facilitar ainda mais o processamento de dados para a nuvem, provedores passa-

[1] http://docplayer.com.br/3276405-Computacao-em-nuvem-openstack.html

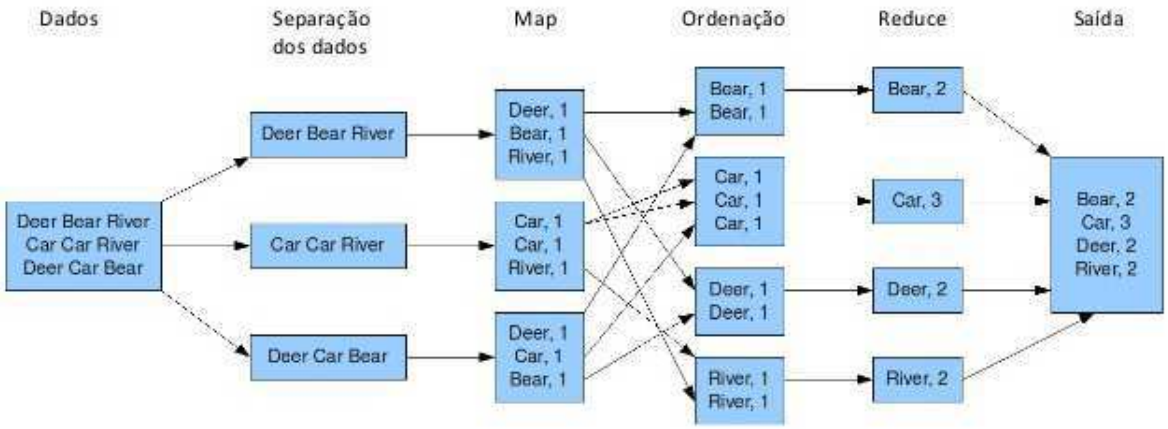

Figura 1.2: Exemplo do funcionamento do paradigma *MapReduce* no seu exemplo clássico de contagem de palavras.[2]

ram a oferecer o serviço de processamento de dados no modelo PaaS, isentando o usuário da necessidade de configurar o *cluster*, deixando-o responsável apenas por alimentar esse *cluster* com a submissão de aplicações dados para o processamento. *Amazon Elastic MapReduce* [3] é o serviço da *Amazon Web Services* com este fim. O *Amazon EMR* permite a criação de *clusters* com poucos cliques.

Aplicações de processamento de dados, em sua grande maioria, possuem características como tolerância a falha e ressubmissão. Essas características possibilitam que o processamento de dados na nuvem seja feito utilizando não somente instâncias sob demanda, mas também instâncias com menor qualidade de serviço, como as oportunistas pois a aplicação não é totalmente interrompida quando há a falhas. O Hadoop, citado previamente, possui essas características, sendo capaz de se recuperar da perda de máquinas durante o processamento. No entanto, Chohan *et. al* mostram que a utilização de instâncias oportunistas para o Hadoop apesar de inicialmente se mostrarem favoráveis, podem acarretar em uma acréscimo significativo no tempo de execução da tarefa, o que conduz ao questionamento sobre o real benefício na utilização desse tipo de instâncias no processamento de dados na nuvem com Hadoop [23].

[2]http://pt.slideshare.net/GuilhermeArajo/mapreduce-37085946

1.1.3 Otimizando a utilização de data centers

Nuvens computacionais oferecem, nas categorias PaaS e IaaS, recursos para os usuários em forma de máquinas virtuais. Essas máquinas podem ser classificadas de acordo com configuração, quantidade de memória, VCPUs, tamanho do disco, como também pelo modelo em que são oferecidas para o usuário. O modelo mais comum é o de instâncias sob-demanda, no qual essas máquinas têm seus recursos alocados para o usuário e oferecem qualidade de serviço (QoS, do inglês *Quality of Service*), em torno de 99% de disponibilidade [7]. Reiss *et. al* [44], em uma análise de um *cluster* do Google com aproximadamente 12K servidores dedicados, mostram que a alocação de recursos em qualquer intervalo de tempo é maior que 80% para memória e mais que 100% para CPU, entretanto, devido ao fato de que os usuários que contratam esses recursos nem sempre utilizam toda a capacidade de processamento das máquinas virtuais, a utilização real desses recursos não passa, em média, de 50% para memória e de 60% para CPU.

Diante dessa baixa utilização, provedores passaram a investigar como melhorar a utilização da nuvem. Uma das propostas é a utilização de máquinas virtuais oportunistas, como no caso das instâncias *Amazon Spot* [2]. Essas máquinas são criadas utilizando recursos ociosos reservados para instâncias sob-demanda, podendo ser interrompidas a qualquer momento, já que têm prioridade menor que as instâncias sob-demanda. Carvalho *et al.* [21] introduzem um novo modelo de *VMs* para a utilização na nuvem. Esse modelo, chamado de Econômico (do inglês *Economy*) combina os modelos sob-demanda e oportunistas, e realiza a provisão de *VMs* oportunistas com base em uma previsão do estado da nuvem, fazendo uso de dados históricos de utilização dos recursos. Com isso, é dada uma qualidade de serviço probabilística ao usuário. Esse modelo oferece ao usuário instâncias mais baratas que as sob demanda e com uma qualidade de serviço superior às instâncias oportunistas comuns.

Ambos tipos instâncias oportunistas aqui descritos podem ser utilizados tanto no cenário de nuvens públicas quanto em nuvens privadas. Em nuvens públicas essas instâncias são oferecidas por um preço inferior às instâncias sob demanda, enquanto no cenário privado, onde não se paga pelos recursos mas existe uma limitação por cotas de recursos, essas instâncias podem ser utilizadas para aumentar utilização da nuvem mesmo quando o limite já teria sido atingido.

A existência desse modelo de instância abre portas para que diversas aplicações pos-

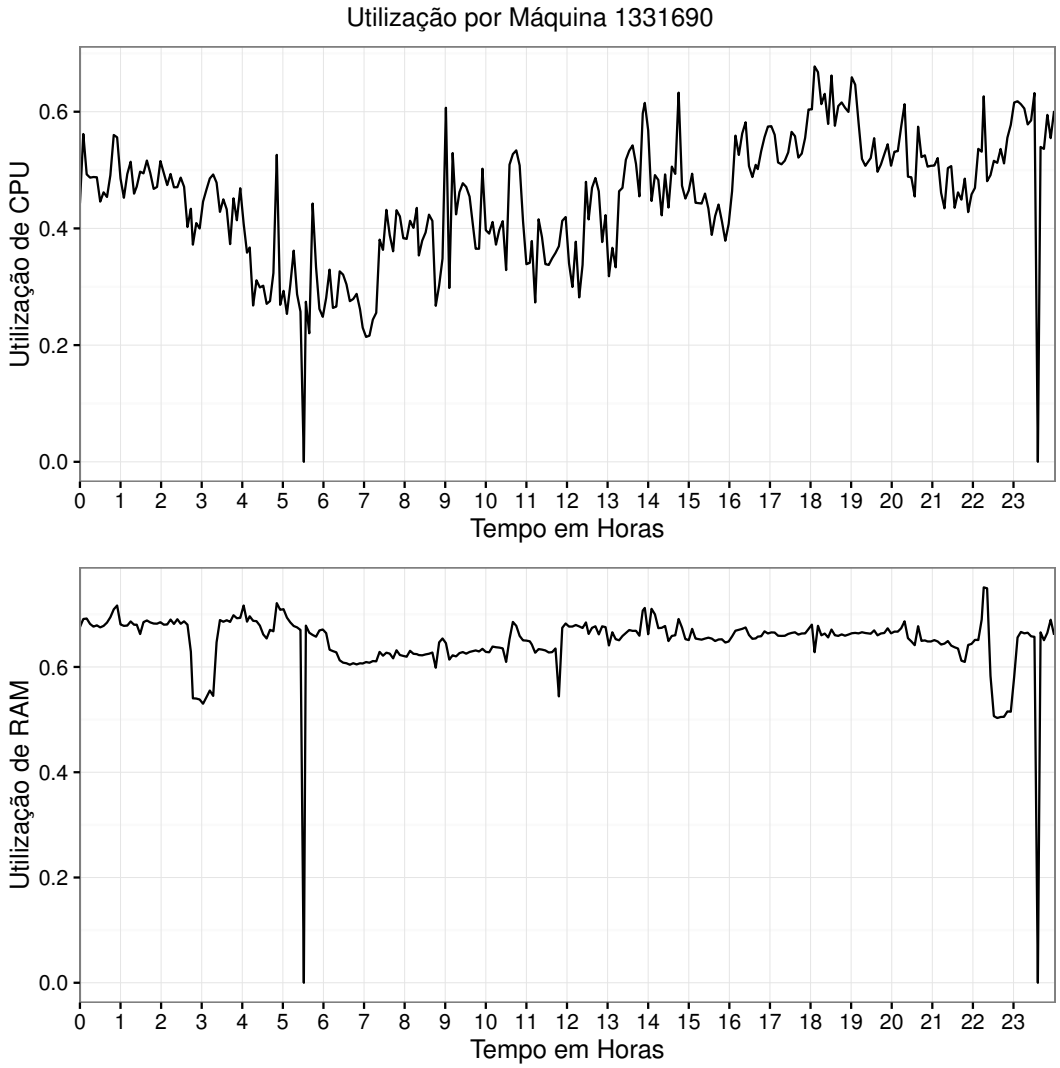

Figura 1.3: Gráfico de utilização de CPU e memória RAM por hora, do primeiro dia do registro do Google para máquina com ID 1331690.

sam ser utilizadas na nuvem de forma mais barata. Devido à probabilidade de perda da máquina virtual, nem todas as aplicações podem ser executadas utilizando instâncias oportunistas, especialmente as que têm contato mais próximo de usuários humanos, podendo ocasionar *downtime* em serviços. Aplicações distribuídas, que em sua maioria têm tolerância a falhas, são boas candidatas a utilizarem esse tipo de máquina virtual. Um exemplo de aplicação que pode fazer uso de instâncias oportunistas é o Hadoop [38; 39; 40], porém essa utilização não é necessariamente vantajosa, apesar de o Hadoop implementar tolerância a falhas a nível de aplicação. Devido à volatilidade desse modelo de instâncias, já que elas podem ser eliminadas em um momento da execução, a chance de existir uma falha aumenta em relação à utilização apenas de instâncias sob demanda. A ocorrência de apenas uma falha, utilizando as configurações padrão do Hadoop, pode acarretar em um acréscimo de até 50% no tempo de completude de uma tarefa Hadoop de duração típica [60; 35], devido ao tempo de detecção da falha, reorganização dos dados para reprocessamento e continuação da tarefa.

Há ainda outras formas de melhorar a utilização da nuvem, como consolidação de máquinas virtuais, uma abordagem que busca fazer o gerenciamento de máquinas virtuais nas máquinas físicas, com objetivo de, por exemplo, minimizar o número de máquinas físicas ativas. Barroso *et al.* [15] mostram que o consumo de energia se comporta de forma diferente em diferentes níveis de utilização de CPU, tendo seu crescimento sublinear, e que a eficiência energética é melhor em utilizações de CPU em torno de 80% ou mais, levando provedores de nuvem a tentarem manter uma alta taxa de utilização de CPU sem prejudicar as aplicações do cliente. Uma das formas de atingir esse recurso por meio de consolidação é fazer a migrações de máquinas virtuais de servidores com baixa utilização, podendo desligalos e manter a nuvem operando com a mesma eficiência de desempenho, com um menor número de servidores, o que acarreta em um aumento da utilização dos recursos ativos.

Existem dois tipos de migração de máquinas virtuais, migração *offline* e a migração viva. A migração *offline* consiste em parar a máquina virtual, fazer a transferência de todo seu disco e configurações para outra máquina física e reiniciar a máquina. Apesar da baixa probabilidade de falha, essa técnica é inviável em alguns cenários devido à paralisação do serviço (*downtime*). Há também a migração viva (do inglês *Live Migration*), que tenta resolver o problema do *downtime* realizando a migração com a máquina virtual em funcionamento,

com apenas uma pequena paralisação, de milissegundos a poucos segundos, que para muitos cenários é considerada irrelevante. O uso de armazenamento compartilhado, comum em ambientes de nuvem, facilita a migração, seja ela viva ou não, mas de todo jeito a migração offline, precisa de um reinicio da máquina, que aumenta o tempo de recuperação. Migrar máquinas virtuais também é uma solução para balanceamento de carga, de modo a evitar interferências nos serviços e até a perda de *VMs* devido à superutilização de uma máquina física. Finalmente, migração também é utilizada com intuíto de otimizar desempenho, colocando máquinas que precisam trocar dados mais próximas diminuído tráfego na rede e também na minimização de custos, agrupando máquinas virtuais para diminuir a quantidade de máquinas físicas ativas.

1.1.4 OpenStack e Sahara

Com a crescente popularização e alta demanda por melhores serviços de computação na nuvem, tecnologias surgiram com objetivo de oferecer maior controle de nuvens computacionais. O OpenStack é uma implementação de código aberto de um sistema operacional de nuvem, construído e mantido majoritariamente por grandes empresas. Sua natureza de código aberto permitiu o rápido crescimento e maturação, se tornando hoje a base da nuvem de grandes empresas como Yahoo!, Disney, Paypal e outros [11].

O OpenStack é dividido em vários subcomponentes, sendo uma parte deles conhecidos como *core*, ou componentes essenciais. Dentre esses componentes essenciais, temos o Nova, responsável por prover recursos computacionais em forma de máquinas virtuais, o Keystone, responsável por autenticação e autorização de usuários, o Neutron responsável pela gerência da rede, Glance, responsável por guardar imagens utilizadas na criação de instâncias, e o Cinder, responsável pelo armazenamento persistente em bloco. Ainda temos outros componentes importantes mas não essenciais, como Swift, responsável pelo armazenamento de objetos e Ceilometer, responsável por coletar métricas de utilização dos servidores e instâncias. Além desses serviços *core,* existem outros componentes não-essenciais, que podem ser adicionados conforme a necessidade do usuário. Há componentes que oferecem serviços no modelo PaaS, como o Sahara, o componente de processamento de dados como serviço. O Sahara, utilizado nessa pesquisa, tem como objetivo oferecer ao usuário a capacidade de criar *clusters* configurados para processamento de dados, possibilitando a criação de *clusters*

com as ferramentas Spark [8], Storm [4] e Hadoop [6], bem como execução de tarefas nestes *frameworks*, de maneira mais simples do que com o processo manual.

O Sahara é dividido em duas partes, sendo a API responsável pela comunicação com outros serviços e também por receber comandos do usuário e a *engine* responsável pelo processamento das requisições em si. Dentre as responsabilidades da *engine* estão a criação do *cluster*, a gerencia da configuração da rede e da comunicação entre as máquinas virtuais, configuração da aplicação, podendo ser Hadoop, Spark ou Storm, além da submissão e monitoramento de tarefas em seus devidos *frameworks*. Na Figura 1.4 podemos ver uma versão simplificada da arquitetura do Sahara. O ponto de entrada é o cliente Python ou interface gráfica, que repassam a chamada para a API REST do Sahara, que, por sua vez, faz as solicitações para criação do *cluster*. Em seguida, a *engine* de provisionamento faz a comunicação com o Nova para a criação das máquinas virtuais. A autenticação desse processo é feita em comunicação com o Keystone, enquanto a execução de tarefas é feita pelo EDP, que se comunica com o Swift para carregar os devidos dados para execução da tarefa.

O Sahara foi utilizado nesta pesquisa para criação de ambientes de processamento de tarefas Hadoop, e também foi utilizado como base da instanciação da nossa solução.

1.2 Problema investigado e relevância

Neste trabalho atacamos o problema de diminuir o tempo de processamento do Hadoop na nuvem em cenários onde utilizar instâncias sob demanda pode se tornar inviável. Para tal, propomos a utilização de máquinas virtuais oportunistas com qualidade de serviço, criadas a partir da utilização de um preditor de carga da nuvem.

Apesar da utilização de instâncias oportunistas com qualidade de serviço, o risco de falha, apesar de reduzido, ainda não pode ser desconsiderado devido à possibilidade de mudanças bruscas de carga ou por erros do preditor. Com o objetivo de evitar a perda das máquinas oportunistas com qualidade de serviço, monitoramos os servidores da nuvem e, ao atingir um determinado limiar (definido pelo administrador da nuvem), realizamos a migração das instâncias oportunistas, evitando sobrecarga no servidor e assim uma falha no processamento.

Essa abordagem minimiza consideravelmente a possibilidade de perda de uma instância

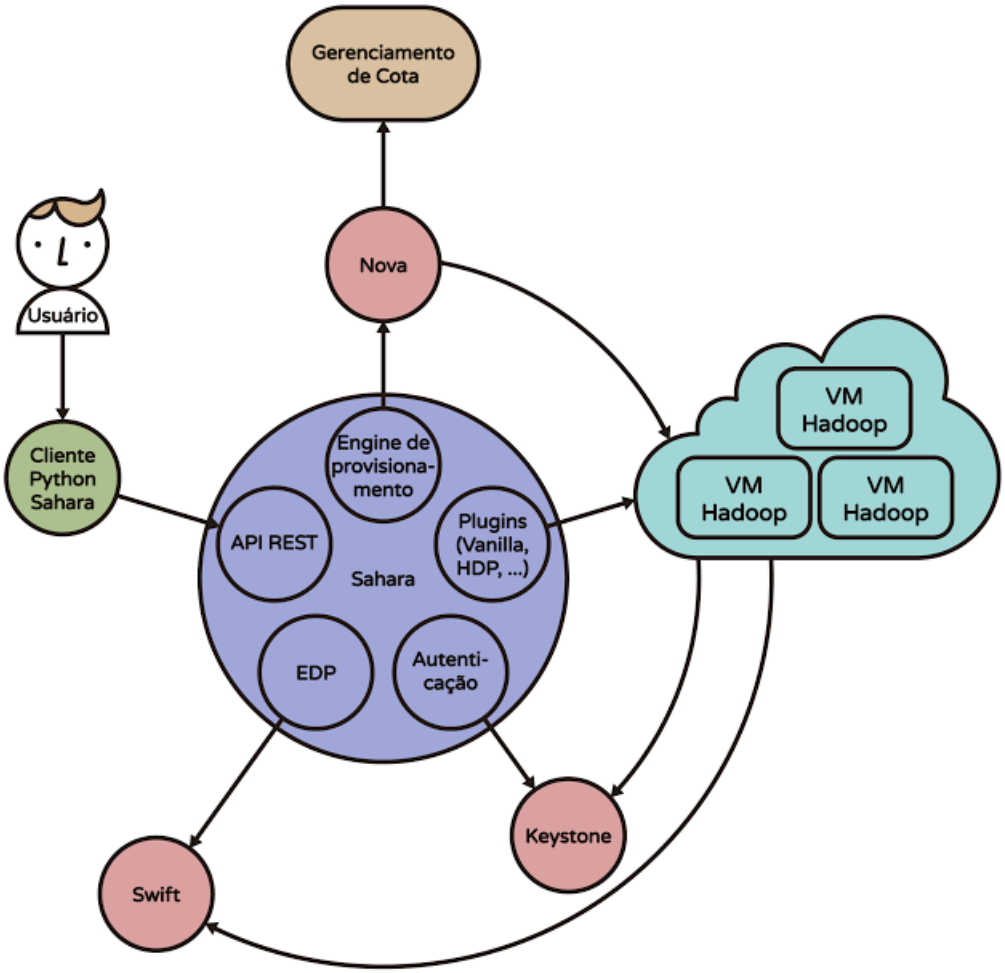

Figura 1.4: Arquitetura do OpenStack Sahara.

oportunista durante o processamento da tarefa e garante também que a instância, apesar de ter uma prioridade mais baixa, não irá prejudicar o processamento das instâncias regulares.

1.3 Metodologia

A partir de um levantamento bibliográfico do estado da arte e da prática, notamos uma lacuna no estado atual do processamento de dados na nuvem, no qual foi definido o escopo deste trabalho. Vemos que processamento de dados na nuvem é algo recente, e alguns conceitos e técnicas de gerenciamento da nuvem são utilizadas para melhorar a vazão no processamento de dados.

Com base nisso, fizemos experimentos que serviram para evidenciar os problemas existentes, mostrando que o processamento pode ser melhorado com oportunismo, ao mesmo tempo que pode ser consideravelmente prejudicado no cenário adverso, ou seja, quando uma falha se apresenta durante a tarefa. Criamos, então, um preditor de carga com objetivo de servir como suporte para nossa hipótese de que o oportunismo pode ser melhorado se houver uma diminuição do risco de perda da máquina durante o processamento. Por fim, fizemos experimentos que mostram que o uso de um preditor não prejudicou o desempenho das aplicações e apresentou evidências de baixa presença de falhas mostrando o real melhoramento do processamento de dados na nuvem. Finalmente, considerando ainda o risco de falhas, realizamos experimentos com migração de máquinas virtuais. Para esse cenário comparamos o tempo de execução de tarefas Hadoop utilizando oportunismo sem falhas e com falhas, evitando a perda da máquina pela migração.

É importante apontar que todos os experimentos foram realizados numa nuvem privada utilizando OpenStack como provedor e o *Sahara* para criar *clusters* Hadoop e executar tarefas.

1.4 Organização do Documento

O restante do documento encontra-se organizado da seguinte forma. No Capítulo 2, fazemos uma discussão sobre trabalhos relacionados que serviram de base e motivação para a pesquisa. Já no Capítulo 3 é detalhada a abordagem proposta para solucionar os proble-

mas identificados. Neste capítulo há informações sobre a utilização de oportunismo para processamento de dados, os preditores utilizados na predição de carga das máquinas físicas da nuvem e sobre como migração pode ser utilizada para melhorar ainda mais a solução proposta. A avaliação da solução, juntamente com os experimentos que evidenciaram o problema encontram-se no Capítulo 4. Por fim, no Capítulo 5 são apresentadas as considerações finais e levantados possíveis trabalhos futuros e limitações desta pesquisa.

Capítulo 2

Trabalhos Relacionados

Neste capítulo, fazemos um levantamento de trabalhos relacionados, detalhando trabalhos sobre utilização de recursos de nuvem, migração, predição de utilização de recursos e como Hadoop pode se beneficiar de oportunismo. Por fim, fazemos um sumário de como nosso trabalho utiliza os conhecimentos aprendidos e como se diferencia dos trabalhos existentes.

2.1 Utilização de recursos da nuvem

Existem várias técnicas disponíveis para melhorar a utilização de ambientes de computação na nuvem. Uma das abordagens mais conhecidas é a utilização de *overcommit* ou *oversubscription* que visa oferecer mais recursos virtuais que recursos físicos realmente presentes na nuvem, porém essa técnica apresenta alguns riscos. Householder *et al.* [29] nos apresentam os perigos existentes na utilização nessa técnica pois ela assume um comportamento dos usuários que pode não ser verdade em todo o tempo, causando queda do desempenho da nuvem e gerando custos para o provedor de nuvem. Ghosh *et al.* [28] reforçam os riscos existentes do *overcommit* e propõe a utilização de uma margem de segurança (intervalo de confiança) baseada para a taxa de *overcommit* diminuindo significativamente a chance de ocorrer excesso de utilização e assim prejudicar usuários e provedores de computação na nuvem.

Outra técnica utilizada, frequentemente em conjunto com *overcommit* é o uso de instâncias oportunistas. Estas instâncias são criadas baseadas em recursos que estão alocados para outras instâncias porém não são utilizados. Reiss *et al.* [44] nos mostram que

a utilização da nuvem é em torno de 50% o que mostra que existe bastante espaço para a utilização de oportunismo. Um exemplo funcional desse modelo é visto na Amazon com instâncias do tipo *Spot* [2]. Essas instâncias possuem um modelo de taxação diferenciado. Seu valor é calculado dinamicamente, o usuário faz uma oferta inicial por ela, e o preço da instância vai aumentando com o tempo, quando o preço atinge um valor maior que o valor que o usuário pagou pela instância, essa instância é perdida. Carvalho *et al.* [21] introduzem um novo modelo de instância oportunistas nomeadas de *Economy*, essas instâncias são também criadas utilizando recursos não utilizados da nuvem, porém seu provisionamento é feito baseado em uma predição de utilização de longo prazo da nuvem, que viabiliza uma garantia de qualidade de serviço para as mesmas. Com esse modelo de instância, os autores mostraram um acréscimo de lucro em torno de 60% enquanto o usuário tem instâncias com baixo custo com maior garantia contra a perda.

2.2 Migração

Uma área de estudo que está em grande evidência é conhecida como *Green Computing*, que tem como objetivo a evolução da computação sustentável visando reduzir o impacto ambiental. Um dos estudos feitos nessa área é a de redução de consumo energético em *data centers*. Nessa direção existem diversos estudos que visam consolidação de máquinas virtuais com objetivo de reduzir a utilização de servidores físicos. Uma das técnicas utilizadas para essa manipulação de máquinas virtuais é conhecida como migração. Tratando apenas de migração viva destacamos o trabalho de Wood et al. [54], que criou *Sandpiper*, um sistema que visa eliminar pontos de alta utilização de recursos, chamados de *hotspots*, pois são considerados pontos com risco de falha e violação de *SLAs*. Esse sistema possui uma estratégia *black-box*, que é agnóstico com relação ao SO e à aplicação, e uma estratégia *gray-box* que considera informações da aplicação. Assim que um *hotspot* é detectado, a migração viva é ativada para mover um conjunto de máquinas virtuais para servidores subutilizados, eliminando assim os *hotspots*. Como resultados, eles mostram que, mesmo com a estratégia *black-box* eles são capazes de mitigar a quantidade necessária de *hotspots* para não violar *SLA*. Nós usamos uma abordagem similar para detectar se a migração de instâncias oportunistas precisa ser ativada baseada nos limites de utilização de recursos.

Ye *et al.* [57] propõem uma técnica de migrar grupos de *clusters* virtuais, que consiste em um grupo de máquinas virtuais que servem a mesma aplicação. Eles investigam várias abordagens para fazer a migração de *clusters* mais rápida e segura que se tratassem de cada VM separadamente. Eles concluem que a migração de um *cluster* de máquinas virtuais não é recomendada, especialmente se o método utilizado é migração viva em bloco (fazendo transferência de disco).

Ainda tratando de migração, existem estudos que investigam como migração viva interfere nas instâncias durante e depois da migração. Xu *et al.* [56] propõem iAware, uma estratégia de migração de máquina virtual que analisa métricas de VM através de um conjunto de *workloads* visando evitar violação de *SLAs*. Eles mostram que é possível conseguir alguns objetivos da migração viva (consolidação, balanceamento de carga) sem degradar performance. No nosso trabalho, quando tratamos da migração, nós buscamos evitar interferências nas máquinas sob demanda devido a sua alta prioridade e não nas máquinas que estão sendo migradas.

2.3 Predição de uso de recursos

Farahnakian *et al.* [27] trazem um novo fator para migração que é a predição de utilização de recursos da nuvem. Essa predição é feita utilizando uma função de regressão baseada no algoritmo KNN. Eles mostram que com a predição e migração trabalhando em conjunto é possível minimizar custos energéticos no *data center*. Ainda na área de predição nós podemos destacar o trabalho de Tomas *et al.* [51] e Islam *et al.* [32]. Tomas *et al.* propõem o aumento de utilização da nuvem através de *overcommit* porém eles propõem que esse *overcommit* seja feito de acordo com uma predição de utilização de recursos. A predição proposta é feita utilizando *triple exponential smoothing function*, uma função baseada em dados históricos que atribui pesos aos dados de entrada baseado em sua relevância para predição e que considera fatores de sazonalidade e tendências, combinada com modelagens estatísticas para inferir a utilização futura. Eles mostram que a técnica proposta é capaz de aumentar utilização de recursos em até 40% sem que a capacidade total do *data center* seja super utilizada. Já Islam *et al.* tem uma abordagem diferente, onde eles fazem a predição de utilização de recursos com foco nas aplicações, auxiliando para que estas possam deci-

dir por requisitar mais recursos para manter performance e disponibilidade. Eles propõem a predição através de dados históricos gerados e utilizam algoritmos de estatísticos de aprendizagem (Rede Neural com correção de erros – *ECNN* – e regressão linear) em conjunto com janela deslizante. Eles realizaram experimentos e avaliaram a qualidade de cada método de predição com e sem janela deslizante utilizando como indicadores erro porcento médio absoluto (*MAPE*), raiz quadrada do quadrado médio do erro, R^2 acurácia de predição e por fim *PRED(25)* e mostraram que utilização da rede neural se mostrou superior em todos os cenários em relação a regressão linear.

2.4 Hadoop utilizando oportunismo

Uma parte importante do nosso trabalho é a utilização de instâncias oportunistas para otimização de tarefas Hadoop na nuvem. Lin *et al.* [39] desenvolveram um sistema chamado de *MOON* (*MapReduce in Opportunistic Instances*) que é uma variação da implementação Hadoop que adota uma arquitetura híbrida de máquinas dedicadas em conjunto com instâncias oportunistas. Essa arquitetura faz alocação de dados em máquinas sob demanda, obtendo uma grande disponibilidade dos dados sem colocar um custo alto de replicação em máquinas oportunistas. Também faz análise do tempo das tarefas para avaliar se as interrupções serão prejudiciais para a tarefa. A estratégia proposta consiste em replicar subtarefas do Hadoop que estão executando em instâncias oportunistas em máquinas dedicadas, tirando proveito dos recursos mais seguros caso haja necessidade. Os autores mostram que numa avaliação geral *MOON* tem um desempenho melhor que o Hadoop puro em quase todos os cenários, sendo pior apenas em um caso devido a limitações de rede.

2.5 Considerações finais

Neste capítulo fizemos um levantamento de trabalhos que visam otimizar utilização de recursos da nuvem e também minimizar o tempo de processamento de tarefas Hadoop. Nossa proposta se fundamenta em diversos conceitos obtidos dos trabalhos citados, onde consideramos a subutilização da nuvem como motivação e também utilizamos técnicas de predição e migração e instâncias oportunistas com qualidade de serviço para aumentar essa utilização

no contexto específico de tarefas Hadoop.

Nosso trabalho se difere dos já citados, pois ao considerarmos a predição para criação de instâncias oportunistas com qualidade de serviço, não consideramos o *cluster* e sim máquinas individuais, assim também como consideramos janelas de predição de 15 minutos, enquanto o trabalho utilizado como referência faz previsões de janelas de meses. Outra diferença que podemos ver é que no contexto mais específico de Hadoop, nós não estamos propondo nenhuma alteração na implementação do escalonador ou disposição de dados do Hadoop, e sim fazemos monitoramento da nuvem para que o Hadoop seja otimizado sem prejuízo para outros ou si próprio. Essa abordagem facilita a aceitação da proposta devido a não alteração de ferramentas para o usuário. Não foi encontrado nenhum trabalho na literatura com considere esses fatores em conjunto.

No próximo capítulo será descrito em detalhes a abordagem proposta para nossa pesquisa.

Capítulo 3

Abordagem Proposta

Este capítulo detalha a abordagem utilizada neste trabalho, que envolve a utilização de oportunismo para o processamento de dados, juntamente com métodos preditivos do estado da nuvem, podendo assim garantir uma qualidade de serviço para instâncias oportunistas. Sugerimos também a utilização de migração para os casos com erro na previsão.

3.1 Arquitetura da solução proposta

Como já discutido no Capítulo 1, nosso trabalho visa a melhorar o processamento de dados com Hadoop na nuvem utilizando instâncias oportunistas com qualidade de serviço. Em outras palavras, consideramos instâncias oportunistas cuja expectativa de existência é maior que o tempo previsto para a realização da tarefa. Para tanto, é proposto um componente que será colocado entre o usuário e o sistema de provisionamento de *clusters*. Neste modelo, o usuário é responsável pela solicitação de criação de um *cluster* Hadoop, selecionando também se deseja a utilização de oportunismo quando possível. O componente desenvolvido é responsável pela tomada de decisão sobre a utilização de oportunismo, passando uma solicitação modificada para o sistema de provisionamento de *cluster* quando necessário, além de monitorar e avaliar a necessidade de migrações.

Desta forma, o objetivo deste componente é identificar a existência de recursos ociosos na nuvem e assim poder solicitar mais máquinas virtuais para o *cluster* Hadoop do usuário, criando *clusters* heterogêneos, onde se apresentam instâncias sob demanda e instâncias oportunistas com qualidade de serviço.

Este componente é composto por 2 partes, um preditor de carga e um tomador de decisões como visto na Figura 3.1. A primeira é o preditor de carga, responsável por, a partir de dados históricos de utilização de CPU e RAM das máquinas físicas que compõem a nuvem, fazer previsões de utilização desses recursos nas próximas janelas de tempo. Mais detalhes do preditor estão na Seção 3.2. A segunda parte do componente é o tomador de decisões, responsável por receber a solicitação de criação de *cluster* do usuário e os dados do preditor de carga e decidir qual ação tomar para a criação do *cluster* para o usuário, bem como a decisão sobre migrações. Mais detalhes sobre ele na Seção 3.3

Para avaliar o desempenho da nossa solução, utilizamos o OpenStack e o Sahara como provedores de nuvem e de provisionamento de *clusters* Hadoop, respectivamente, e implementamos uma instanciação do modelo proposto.

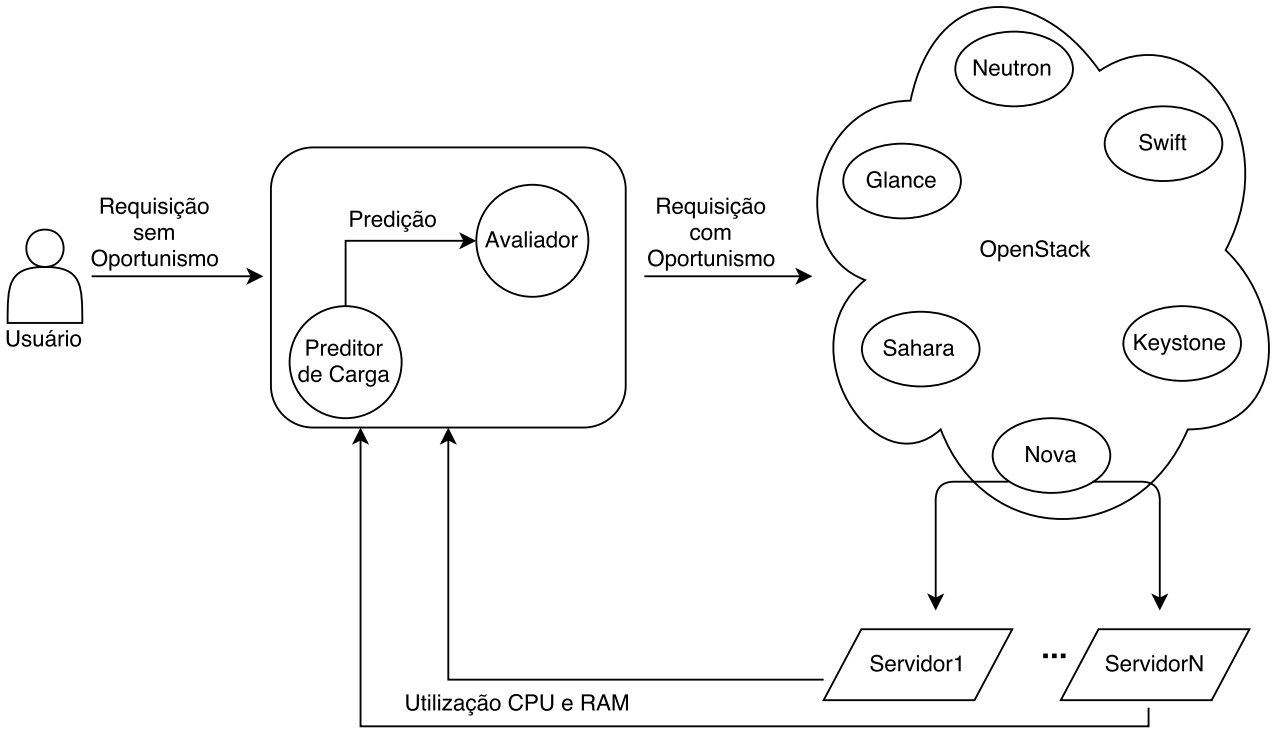

Figura 3.1: Arquitetura da solução proposta. Um novo componente é inserido entre o usuário e o Sahara e é responsável por avaliação do estado da nuvem para criação de *clusters* Hadoop utilizando oportunismo ou não.

3.2 Preditor de carga

Prever carga de utilização é uma técnica já conceituada na área e utilizada em diversas pesquisas com intuíto de melhor gerenciar recursos da nuvem. Em sua grande maioria, os estudos disponíveis na literatura fazem predição da utilização agregada da nuvem [21; 45], podendo a predição também ser feita de máquinas físicas individuais na nuvem. A predição também varia no que se refere aos horizontes de predição, podendo ser feita para um curto prazo, minutos ou horas, ou prazos grandes como semanas ou até meses.

O foco do nosso trabalho não é desenvolver um preditor de carga robusto, porém fazer uso de preditores simples de uso geral e mostrar que, mesmo com utilização desses preditores simples, a proposta de utilização de instâncias oportunista com qualidade de serviço é relevante para acelerar o processamento de dados na nuvem. Assim, utilizamos 4 preditores disponíveis no pacote *forecast* da linguagem de programação R. Cada um desses preditores é detalhados abaixo.

- Ingênuo: prevê o valor futuro como sendo igual ao último observado. Apesar de ser um preditor muito simples, é comumente usado como *benchmark* em relação a preditores mais complexos;
- Média Móvel: utiliza os dados passados como entrada e prevê como futuro a média desses dados;
- *RWF*: similar ao ingênuo, porém faz uma predição por caminhada aleatória (do inglês, Random Walk Forecast) escolhendo uma tendência de crescimento ou queda aleatoriamente;
- *RWF* com *Drift*: similar ao *RWF*, porém a tendência de queda ou crescimento é baseado numa média dos dados históricos.

É comum que os preditores acima tenham resultados de predição diferentes e, por isso temos que tomar uma decisão de qual predição utilizar. Com o objetivo de ser mais conservador, utilizamos sempre a predição que nos retorna o maior valor de utilização de recursos, ou seja, nos dá a menor quantidade de recursos ociosos. Esse conservadorismo é adequado para que evitemos extrapolar a capacidade dos servidores, diminuindo assim o risco de falhas e degradação de desempenho em instâncias com prioridade mais alta. Como entrada,

nosso preditor recebe dados de utilização de CPU e memória RAM das máquinas físicas em intervalos de 5 minutos. Esse valor foi selecionado pois seguimos o modelo dos registros de utilização do *cluster* do Google [9]. Também é passado para o preditor o valor total de CPUs e memória RAM de cada servidor. Podemos ver no Código 3.1 um exemplo do arquivo de entrada do preditor. A primeira coluna contém a quantidade CPUs utilizadas no servidor em número absoluto, por exemplo, no momento da coleta estavam em atividade 23 CPUs, o valor retornado será 23. Na segunda coluna traz a utilização percentual de memória RAM que varia entre 0 a 1. A terceira coluna é a informação do total de CPUs do servidor, ou seja, a quantidade de CPUs disponíveis na máquina física e na quarta é o total de memória RAM em GB, e por fim a quinta coluna nos traz um índice de medição, que é incrementado a cada 3 medições. Esse índice é de suma importância, pois nós utilizamos o preditor para prever em janelas de 15 minutos, sendo necessário o agrupamento de medições para formação de janelas de entrada do mesmo tamanho das janelas de saída. Utilizamos janelas de 15 minutos, pois buscamos um tamanho de janela de fosse mais adequado ao tamanho médio de uma tarefa Hadoop, que dura em média de 20 a 30 minutos [33], e que fosse flexível para o caso de tarefas um pouco maiores que essa média. Esse tamanho da janela também foi utilizado pois prevemos apenas duas janelas no futuro sem perder muitos dados agregados na entrada.

Ao receber o arquivo de entrada, o preditor seleciona o maior valor de CPU e o maior valor de memória RAM para cada conjunto de 15 minutos, como informação para ser feita a predição, mesmo que os valores pertençam a medições distintas. Por exemplo, para o índice 0 o preditor selecionaria 23 como o número de CPUs utilizadas e 0.1235 como a memória utilizada. Usamos essa abordagem novamente com o intuito de diminuir o risco de tentarmos utilizar recursos que não estão disponíveis por erro do preditor. A Figura 3.2 exemplifica como o preditor trata os dados de entrada para fazer a predição no janelamento correto.

Código Fonte 3.1: Exemplo do arquivo de entrada para o preditor

```
1 23.0;0.123541636364;80.0;44.0;0
2 19.0;0.123013090909;80.0;44.0;0
3 19.0;0.122898272727;80.0;44.0;0
4 22.0;0.123594454545;80.0;44.0;1
5 20.0;0.123201818182;80.0;44.0;1
6 22.0;0.123129454545;80.0;44.0;1
```

```
20.0;0.123568090909;80.0;44.0;2
18.0;0.123186272727;80.0;44.0;2
21.0;0.123768;80.0;44.0;2
21.0;0.123080727273;80.0;44.0;3
22.0;0.123011636364;80.0;44.0;3
22.0;0.123700909091;80.0;44.0;3
20.0;0.123193;80.0;44.0;4
24.0;0.123241454545;80.0;44.0;4
18.0;0.123127818182;80.0;44.0;4
19.0;0.123297363636;80.0;44.0;5
```

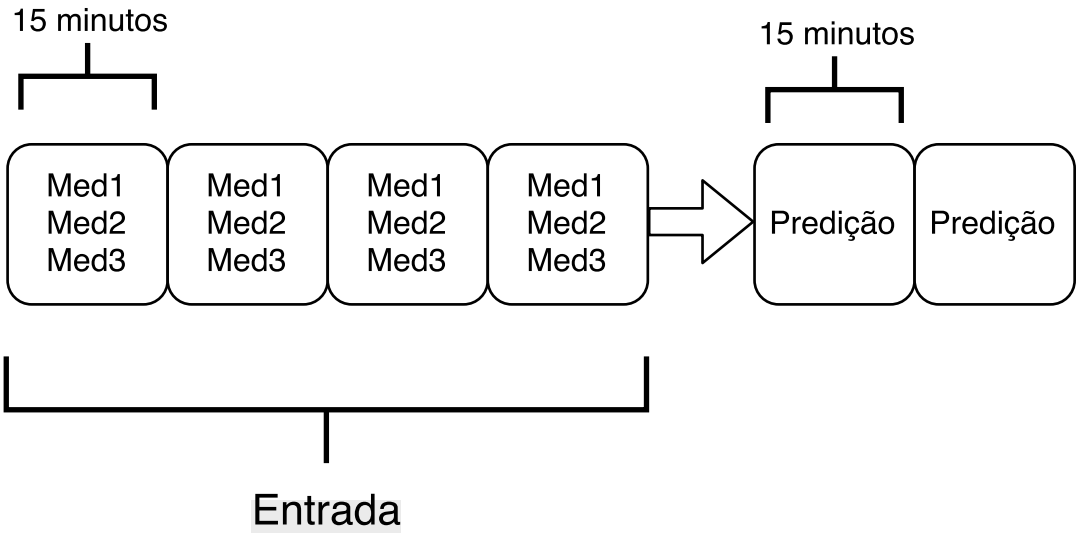

Figura 3.2: Exemplo do janelamento utilizado na predição de recursos disponíveis. Onde Med1, Med2 e Med3 representam as três medições, em intervalos de 5 minutos, utilizadas na criação da janela utilizada pelo preditor. O maior valor individual de utilização de CPU e de memória RAM entre esses 3 é selecionado para ser considerado na predição.

O resultado final do preditor é a quantidade de máquinas disponíveis para criação de um *cluster* Hadoop. Calculamos o número de máquinas baseado em um tamanho de VMs que consideramos apropriado para processamento de tarefas Hadoop, sendo 2 CPUs e 4 GB de memória RAM.

3.3 Tomador de decisões

O tomador de decisões recebe a informação do preditor, que contém a quantidade de servidores que podem ser utilizados na criação do *cluster*, juntamente com a solicitação do usuário. Com essas informações o tomador de decisões solicita ao Sahara a criação de um *cluster* Hadoop com a quantidade de máquinas disponíveis.

Uma vez submetidas as aplicações, o tomador de decisões é responsável também por monitorar os servidores onde existem instâncias oportunistas do *cluster* e identificar possíveis picos que causariam a perda de instâncias oportunistas. Essa função é de suma importância pois é a partir dela que podemos acionar a migração viva [57] de instâncias oportunistas de uma máquina física para outra. Como mencionado na Seção 1.1.3, migração é uma técnica utilizada em ações de consolidação de máquinas virtuais em máquinas físicas para diversos fins. Neste trabalho consideramos migração como uma forma de reparar um erro de predição ou algum aumento inesperado de carga que cause uma das máquinas oportunistas do *cluster* Hadoop falhar, ou seja, ser perdida. Utilizamos migração viva, onde a máquina virtual é transferida de um servidor físico para outro sem que a máquina seja desligada, pois com a migração *offline*, ao desligar a máquina, teremos o mesmo efeito da terminação da mesma, elevando o tempo de completude da tarefa Hadoop.

Ao detectar o perigo de falha, que aqui consideramos como sendo quando a utilização da máquina física chega em um limiar de 90% de CPU e/ou memória RAM, o tomador de decisões é responsável por acionar o componente responsável pela migração. No nosso cenário, onde utilizamos OpenStack, esse componente é conhecido como *Nova*, passando como informação a máquina virtual a ser migrada, e qual a máquina física de destino.

Capítulo 4

Avaliação

Neste capítulo avaliamos a abordagem proposta. Primeiro detalhamos os pré-experimentos executados na preparação do ambiente dos experimentos principais. Em seguida, discutimos o ambiente onde executamos os pré-experimentos e os experimentos principais. Por fim, mostramos os resultados obtidos e fazemos uma análise dos mesmos.

4.1 Pré-experimentos

Nesta seção mostramos como foi avaliado o ambiente utilizado nos experimentos desta pesquisa. Essa avaliação envolve a validação da injeção de carga feita durante os experimentos utilizando como referência os dados do Google e também a validação do preditor de carga. As próximas subseções detalham cada um desses pré-experimentos.

4.1.1 Dados do Google e injeção de carga

Todos os experimentos foram executados em uma nuvem privada no Laboratório de Sistemas Distribuídos, da Universidade Federal de Campina Grande, e como a utilização dessa nuvem é bastante restrita (pouca variação nos tipos de carga e horários) foi necessário gerar uma carga sintética para simular um ambiente de nuvem real. Essa carga foi gerada baseada em dados públicos do *data center* do Google [9] que contém, entre outros dados, informações de utilização de CPU e memória RAM de aproximadamente 12.500 máquinas em um período de 29 dias.

Com base nos dados de utilização de máquinas individuais do *cluster* do Google, geramos uma carga de CPU nos servidores da aplicação. Essa geração foi feita usando a ferramenta lookbusy [10]. O lookbusy é um *script* Python que pode gerar carga de CPU, memória RAM e I/O, com valores definidos pelo usuário ou aleatoriamente. Para validar essa geração de carga, fizemos um experimento gerando uma carga baseada no *trace* de uma máquina selecionada aleatoriamente do *cluster*, e coletamos, através do pacote *sysstat* [13], a utilização real de CPU. Nossa coleta foi feita em intervalos de 5 minutos, assim como os dados do Google.

A Figura 4.1 mostra que a carga sintética criada é praticamente igual a carga original. Podemos assim considerar a utilização da carga sintética confiável para o ambiente experimental.

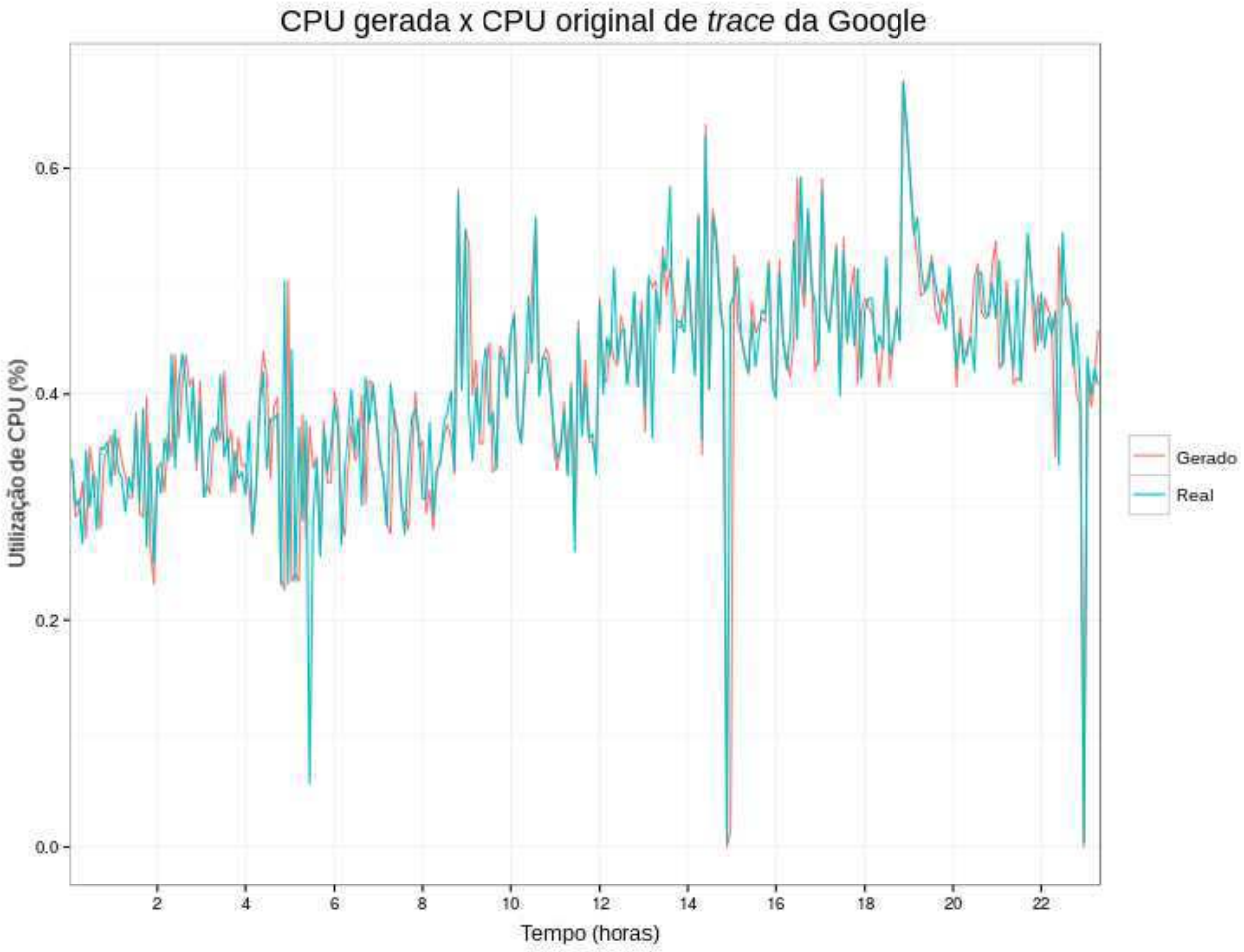

Figura 4.1: Comparação entre sintética e carga real, baseada em dados de *data center* do Google.

4.1.2 Predição da carga

O preditor de carga exerce uma função primordial na nossa solução. Embora não seja o foco do trabalho a criação de um preditor de carga mais robusto para nosso cenário de instâncias oportunistas, precisamos validar que os preditores utilizados apresentam erros aceitáveis. Como já discutido, utilizamos 4 preditores (média, ingênuo, RWF, e RWF com *drift*), alimentando-os com os dados de máquinas físicas do Google, e executamos a predição em forma de janela deslizante. Em seguida, nós calculamos o erro de predição:

$$Erro = V_r - V_p,$$

onde V_r é o valor real de recursos e V_p é o valor previsto.

Nós utilizamos duas abordagens de predição, uma abordagem comum, onde é utilizado o exato valor da predição e uma abordagem pessimista, onde a predição é feita e é utilizado o valor maior do intervalo de confiança. Ambas abordagens são válidas e podem ser utilizadas de acordo com a necessidade do provedor de nuvem. Se o provedor acredita que é melhor aproveitar ao máximo, sugerimos a utilização da abordagem de predição comum, caso ele queria ser mais conservador, então sugerimos a abordagem de predição pessimista.

A Figura 4.2 nos mostra um *boxplot* dos erros de predição por preditor quando utilizamos a abordagem de predição comum. Podemos observar que o 1º quartil, média e 3º quartil se encontram em 0, nos dando segurança que nosso preditor erra em poucos casos. Na Figura 4.3 nós temos *boxplots* para o abordagem de predição pessimista. Neste caso podemos ver quem o 1º quartil está em 0, enquanto média e 3º quartil estão em 1, ou seja, esse preditor erra mais, como esperado.

Nas Figuras 4.4 e 4.5 temos as CDFs das abordagens de predição, comum e pessimista, respectivamente separadas por preditor. Como podemos observar para todos os preditores temos mais de 80% de chance do valor do erro ser 0 ou menos, e menos de 20% de chance de ser menor que 0, o que nos reafirma os dados da Figura 4.2 sobre a taxa de acerto do preditor com abordagem comum. Já na Figura 4.5 observamos que a probabilidade de um valor ser abaixo de 0 é quase 0%, porém a probabilidade de ser 0 gira em torno de 40% apenas. Essa abordagem de predição nos traz mais confiança que as instâncias não serão perdidas, pois tende a retornar uma quantidade de recursos ociosos menor que realmente estão disponíveis.

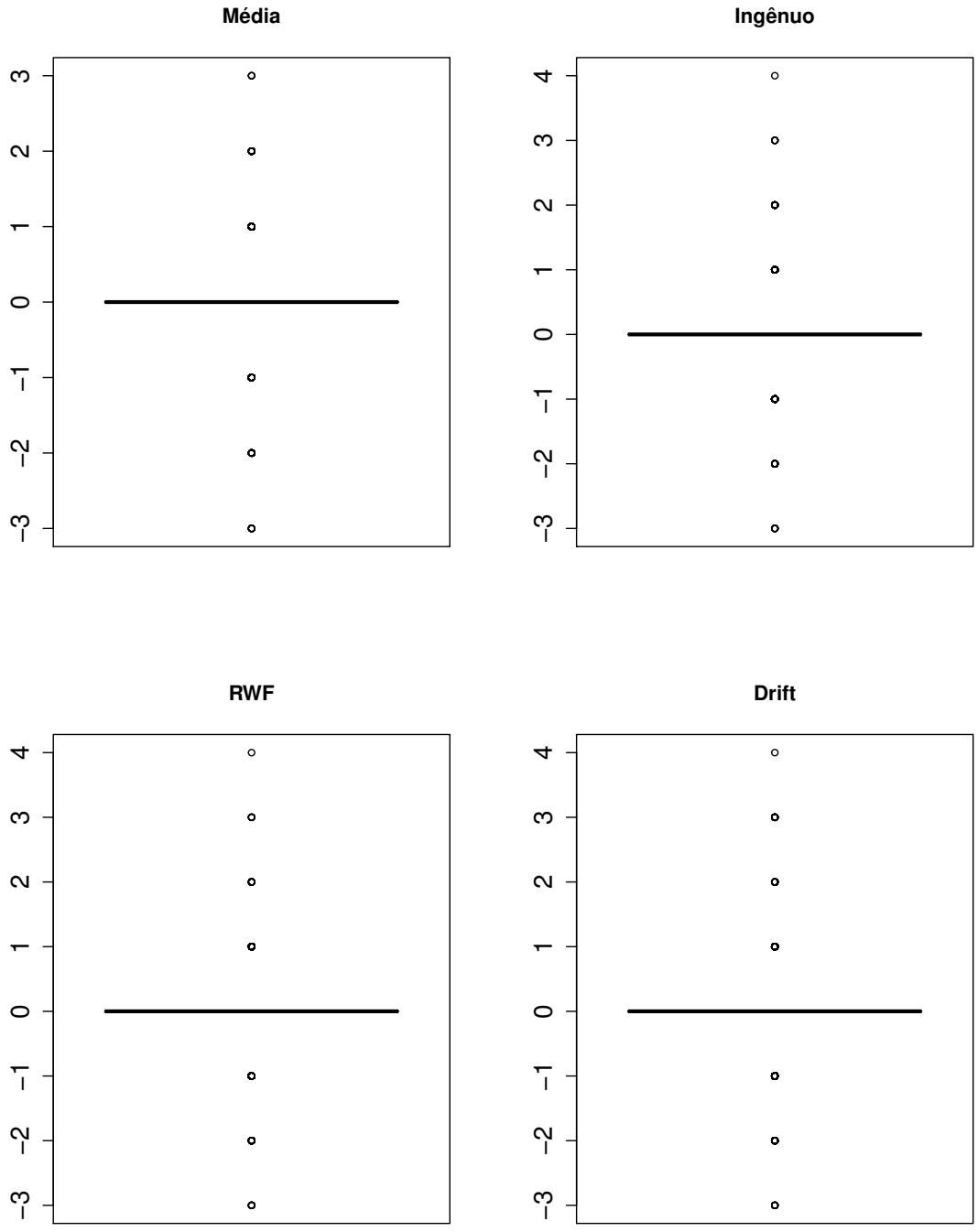

Figura 4.2: Boxplot dos erros de predição de recursos ociosos para máquina com ID 1331690 do *cluster* do Google, utilizando a abordagem comum.

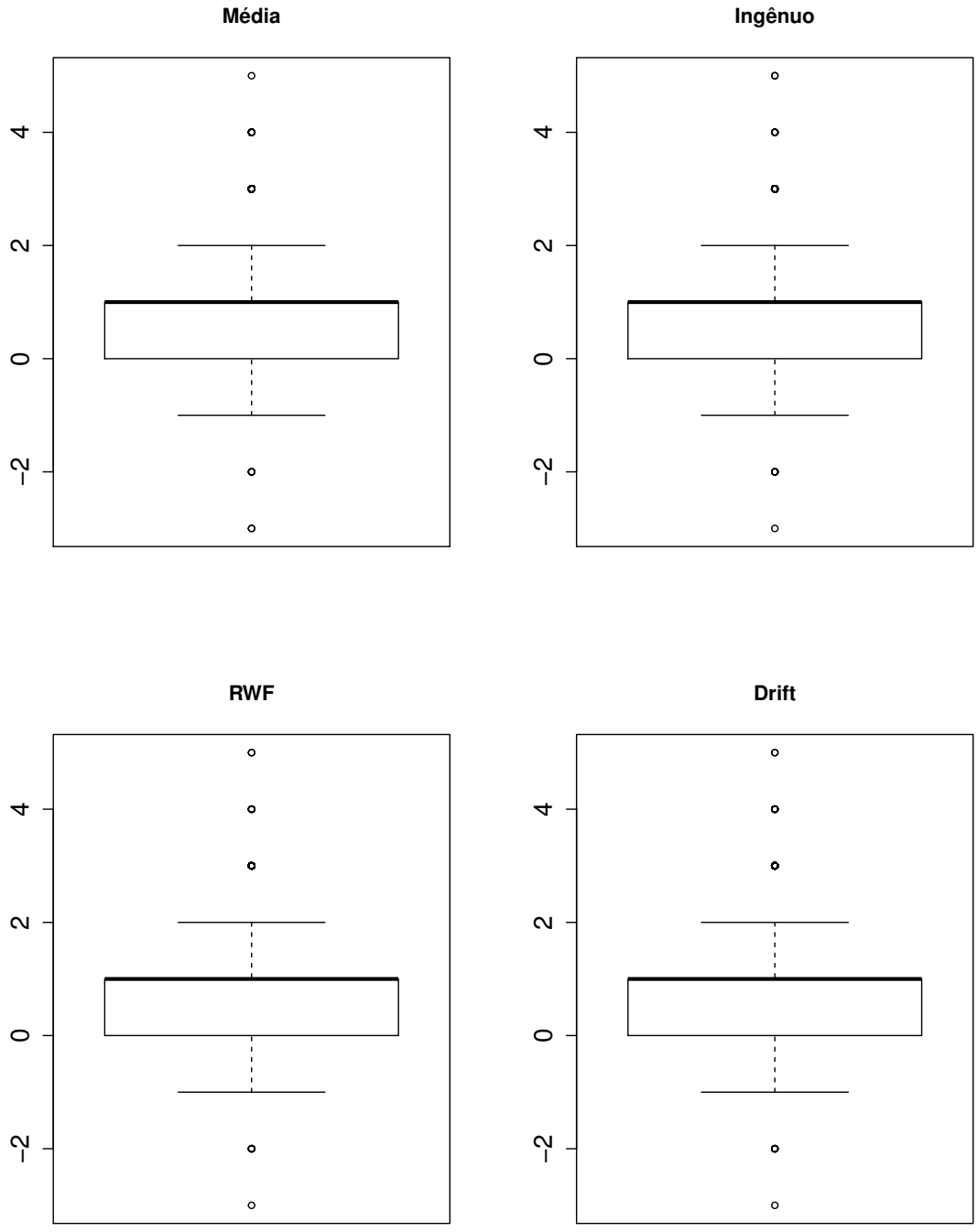

Figura 4.3: Boxplot dos erros de predição de recursos ociosos para máquina com ID 1331690 do *cluster* do Google, utilizando a abordagem pessimista.

Na Tabela 4.1 vemos em mais detalhes o intervalo de confiança dos erros para 20 máquinas do *cluster* do google, em duas variáveis. A primeira temos o percentual de acertos do preditor, na segunda temos os erros negativos, ou seja, erros em que obtivemos mais máquinas no preditor que no registro real. É importante notar que erros positivos não são prejudiciais para nossa solução, pois eles indicam a existência de mais recursos ociosos.

Podemos ver que temos uma taxa de acerto variando entre 43,3% a 66.9% para o preditor comum enquanto no preditor pessimista a taxa de acerto é bem baixa, indo de 19,5% a 37,4%. Entretanto, focando nos erros negativos, que são os que queremos evitar, o preditor pessimista é bem superior, ficando com erros de 1,2% a 4,4% enquanto o preditor comum tem erros de 16,3% até 29,8%.

Tabela 4.1: Intervalo de confiança

Preditor	Comum		Pessimista	
	Acertos	Erros Negativos	Acertos	Erros Negativos
Média Móvel	59,2% - 43,3%	29,8% - 21.2%	35,7% - 19,5%	4,4% - 1,2%
Ingênuo	66,9% - 52,9%	23,2% - 16,3%	37,4% - 21%	2,9% - 2,1%
RWF	66,9% - 52,9%	23,2% - 16,27%	36,7% - 20%	2,7% - 1,9%
Drift	65% - 50.6%	23,9% - 16,9%	35,9% - 19,6%	2,7% - 1,9%

4.1.3 Hadoop na presença de falha

O Hadoop é capaz de se recuperar de uma falha, porém a presença de apenas uma falha pode aumentar significativamente o tempo de completude de uma tarefa [33]. Em tarefas pequenas, esse tempo pode aumentar em mais de 100% como mostra a Figura 4.6. Neste experimento, utilizamos 3 tipos de tarefas Hadoop que representam as categorias de tarefas usualmente executadas no Hadoop e forçamos a falha de um dos nós do *cluster* Hadoop durante o processamento. As tarefas Hadoop utilizadas foram:

- Estimador Pi – essa tarefa tenta estimar o número Pi, sendo uma tarefa de alta utilização de CPU, mas sem acesso a disco;
- Contador de palavras – essa é a tarefa mais clássica do Hadoop, sendo menos intensa em processamento e mais intensa em acesso a disco;

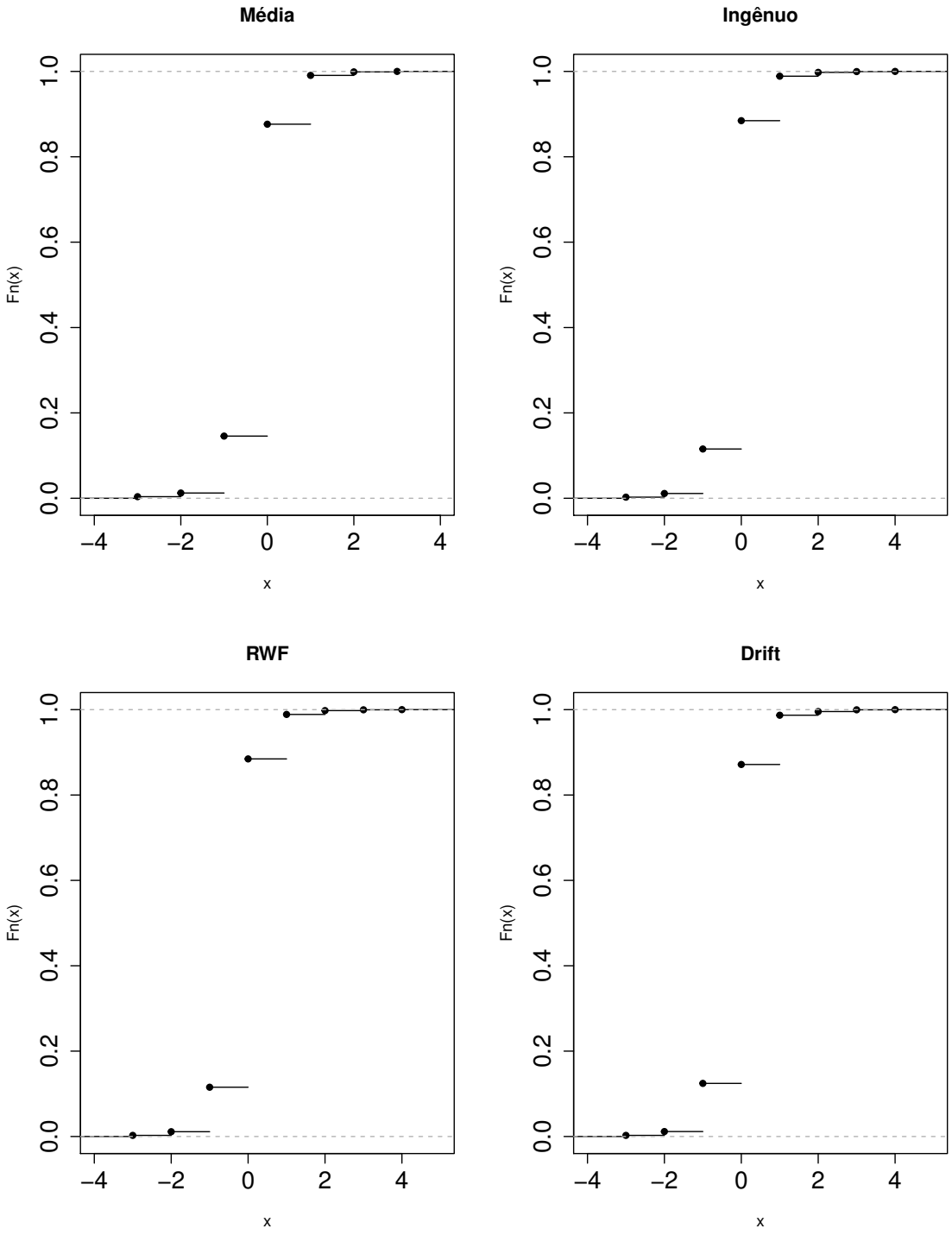

Figura 4.4: CDF dos erros de predição por preditor para a máquina com ID 1331690 do *cluster* do Google, utilizando a abordagem comum.

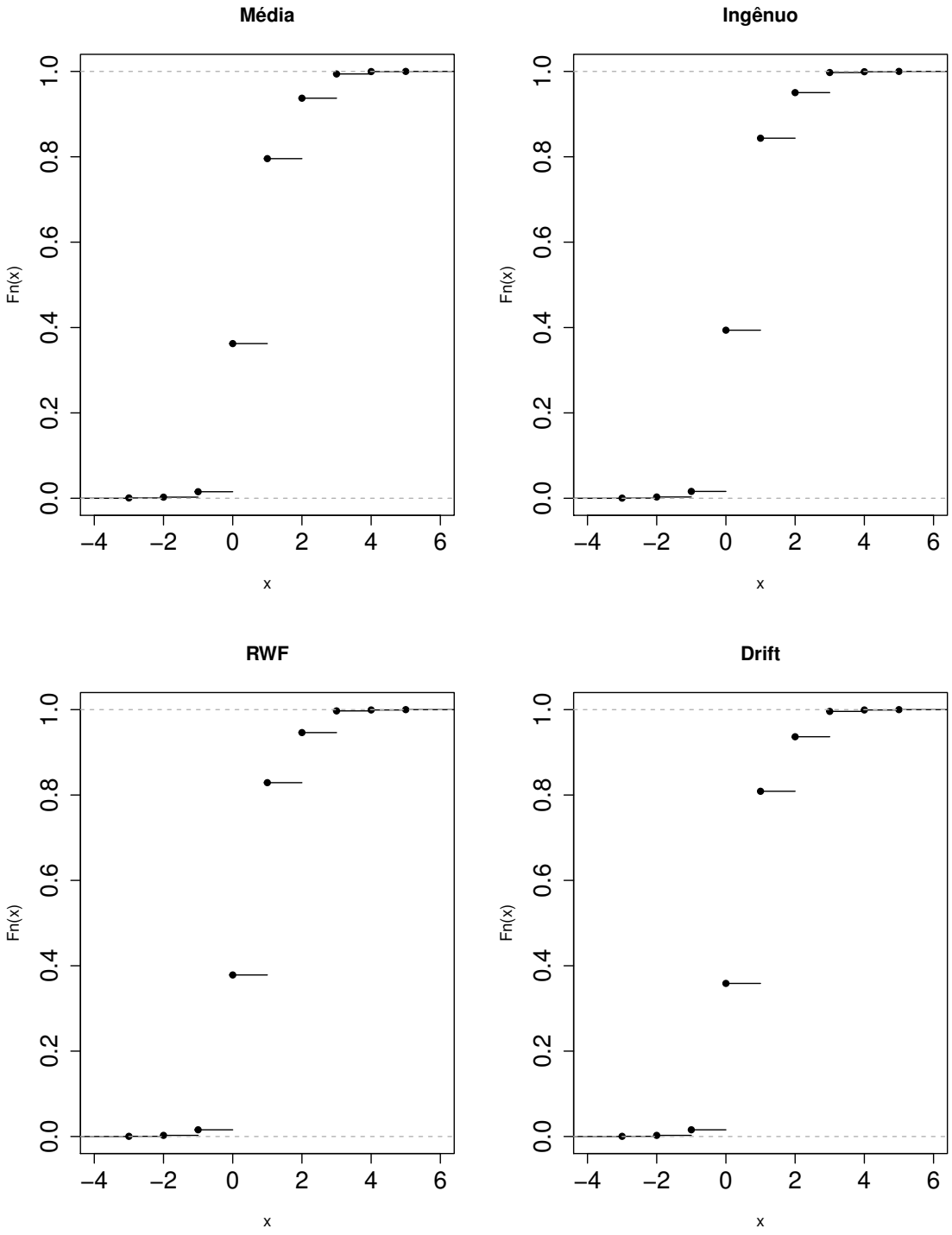

Figura 4.5: CDF dos erros de predição por preditor para a máquina com ID 1331690 do *cluster* do Google, utilizando a abordagem pessimista.

- TeraSort – essa é uma tarefa ordenação e possui um balanço entre processamento e acesso a disco.

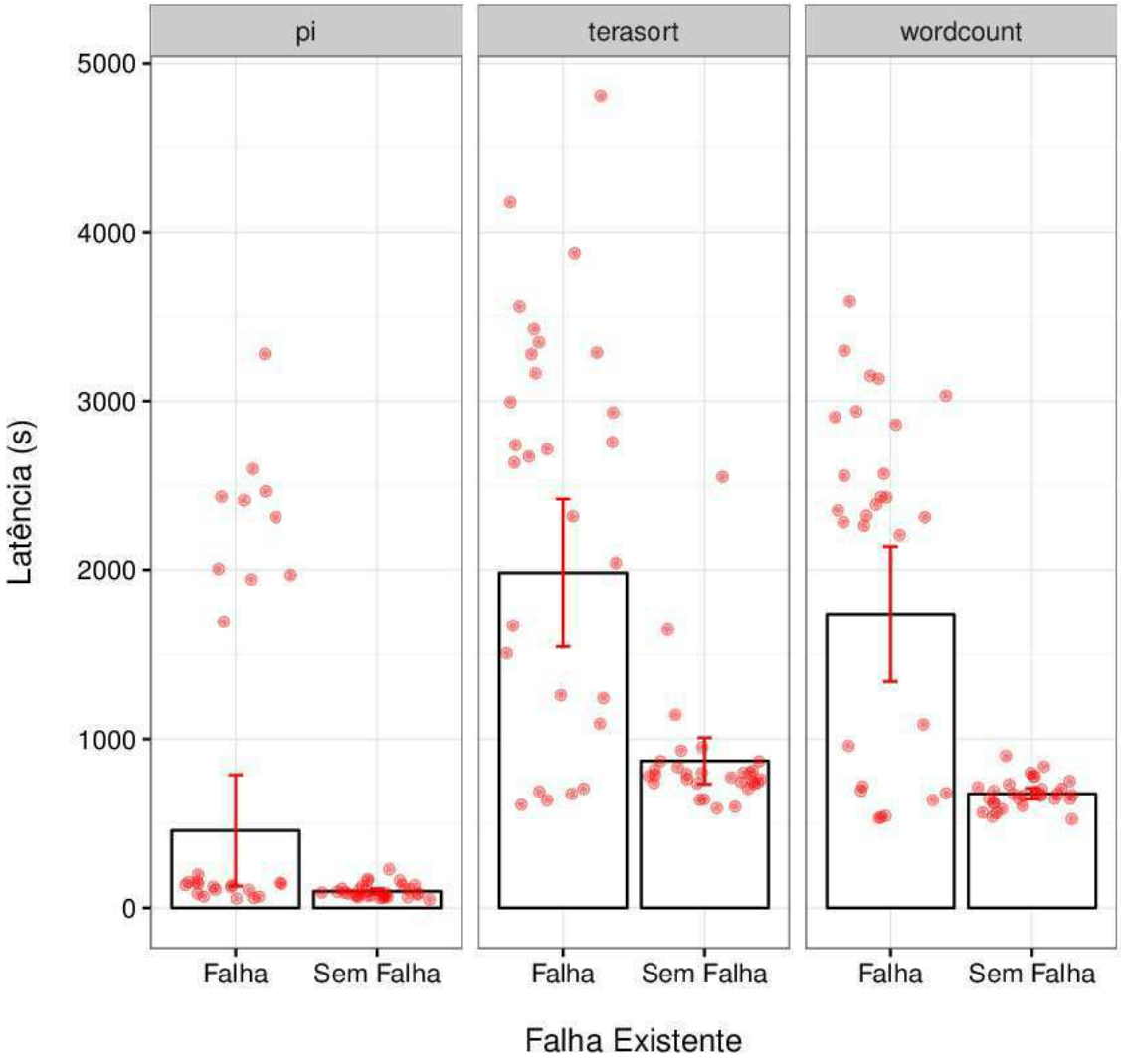

Figura 4.6: Tarefas Hadoop na presença de falha.

Buscamos também encontrar alguma tendência de crescimento ou decrescimento do quão prejudicial uma falha é para a completude da tarefa, ou se existe algum momento do processamento em que a falha é mais crítica. Essa informação poderia nos ajudar a identificar pontos cruciais onde a falha deveria ser evitada sempre. Na Figura 4.7 podemos ver que o momento em que a falha ocorre não influencia diretamente no tempo de completude da tarefa, apenas podemos afirmar que a falha em qualquer momento do processamento acarreta em um acréscimo no tempo de completude.

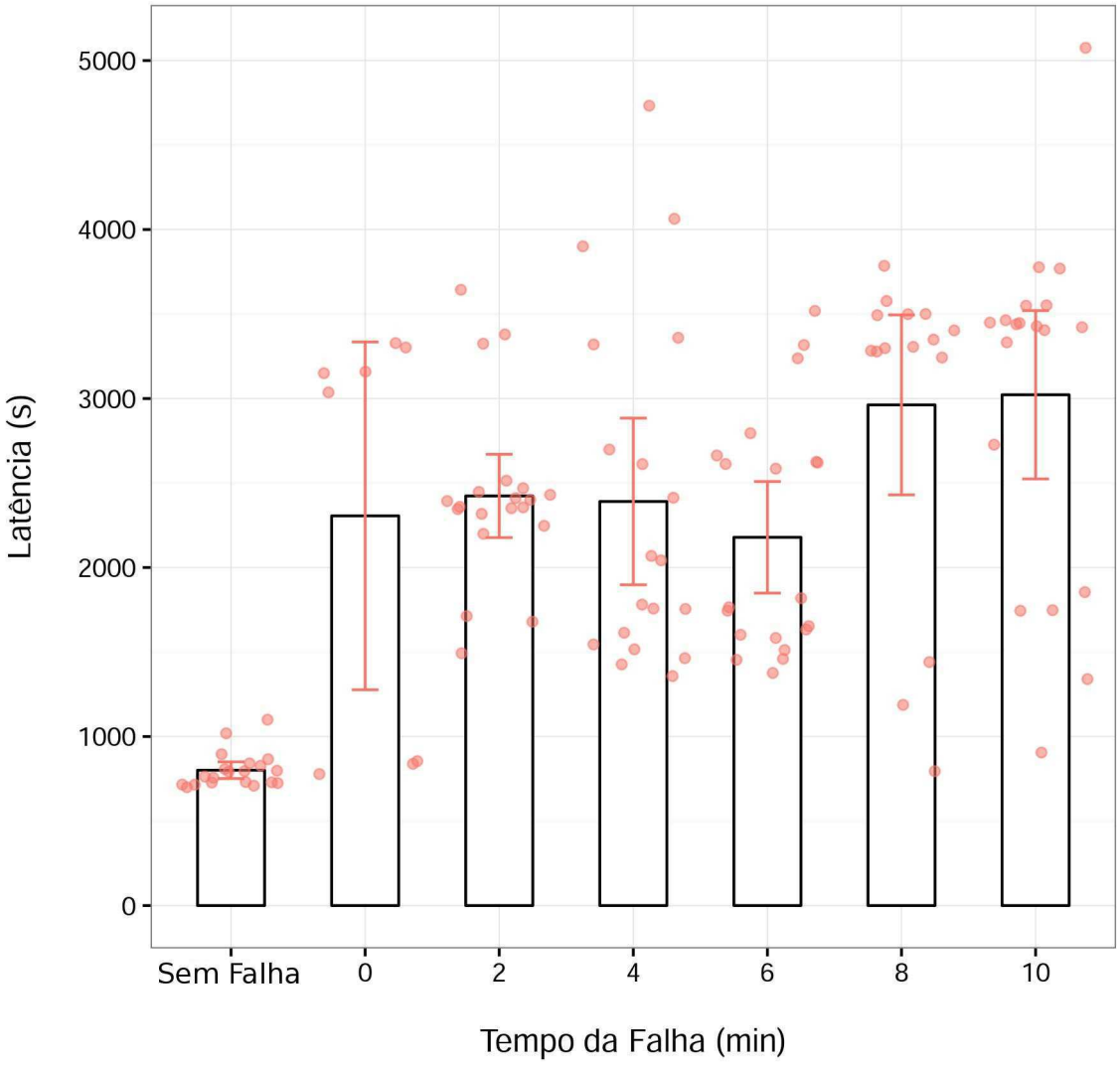

Figura 4.7: Tempo médio de completude de tarefas Hadoop na presença de falha em diferentes pontos da execução, incluindo a fase *Map* e *Reduce*.

4.1.4 Migração

Um conceito bastante utilizado neste trabalho foi a migração. Utilizamos migração para intervir no *cluster* Hadoop quando uma possível falha se apresenta. Entendemos que essa abordagem é apropriada pois quando utilizamos migração para evitar que uma máquina seja perdida não temos um aumento significativo no tempo de completude da tarefa. A Figura 4.8 nos mostra que os tempos de completudes das tarefas são similares nos cenários com e sem migração, o que nos dá uma segurança que na presença da falha não aumentaremos o tempo de completude de tarefa.

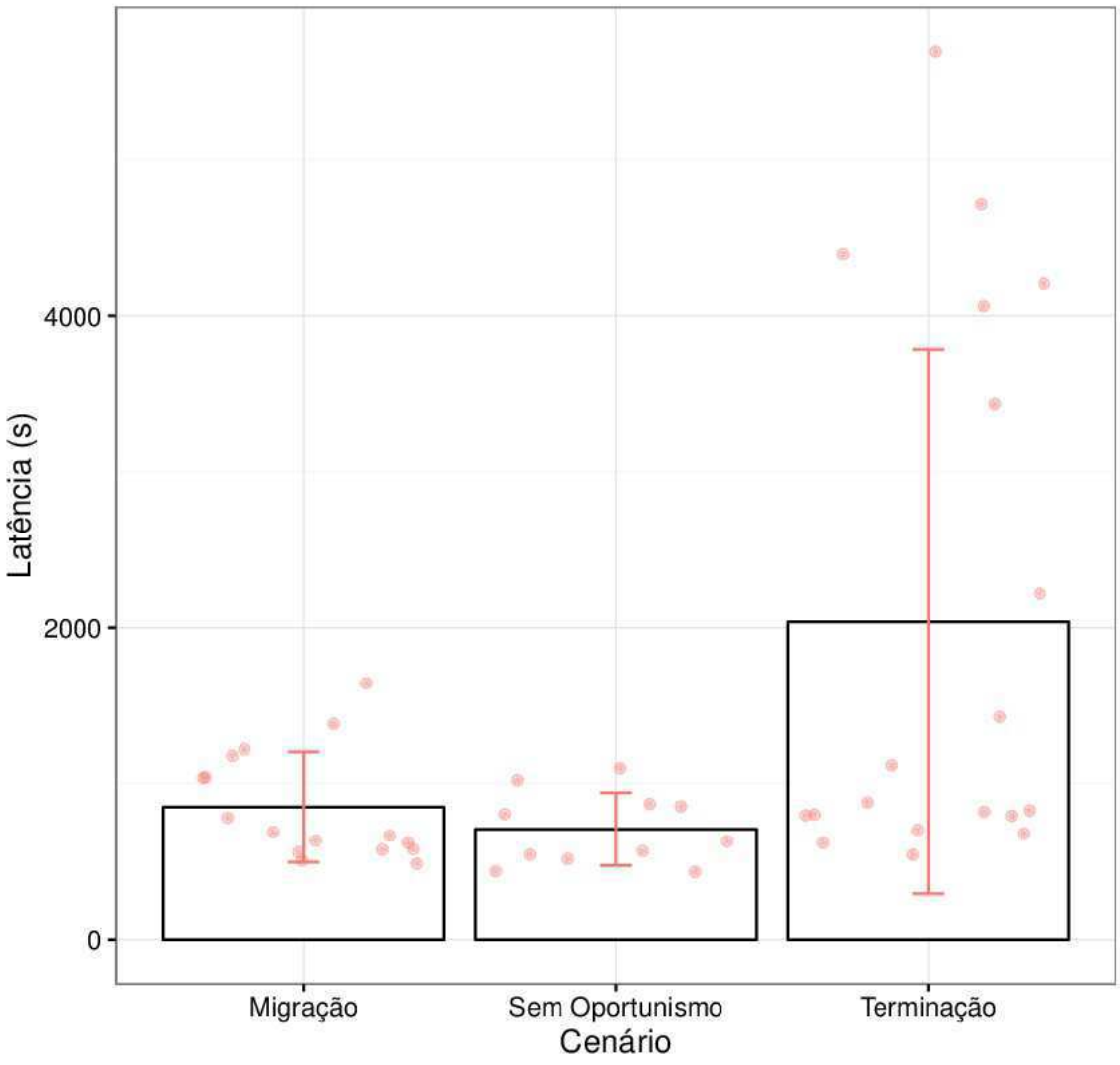

Figura 4.8: Tempo médio de execução de uma tarefa Hadoop quando uma instância é migrada, quando a execução ocorre sem oportunismo ou é bruscamente perdida.

4.2 Ambiente dos experimentos

Para os experimentos a seguir, dispomos de uma nuvem, controlada pelo OpenStack versão Kilo com o Sahara na mesma versão. Esta nuvem é composta de 4 servidores dedicados para criação de VMs, 1 servidor de controle, 1 servidor de rede e 3 servidores de armazenamento. Destes 4 servidores dedicados para computação, 2 foram dedicados para nossos experimentos. Esses servidores são HPE *ProLiant BL460c Gen8*, com dois processadores Intel Xeon E5-2630L, cada um com 6 núcleos operando a 2GHz, chegando a 24 CPUs, com *Hyper-threading*. Os servidores são conectados através de uma rede Gigabit e têm, somados, 100 GB de memória RAM e tem configurado NFS (*Network File System*) como sistema de arquivo. A utilização do NFS permite o compartilhamento dos dados entres as máquinas físicas e possibilita a utilização da migração viva.

4.3 Experimentos

Nós realizamos um experimento que visa verificar se a utilização de oportunismo realmente é capaz de reduzir o tempo de completude de uma tarefa Hadoop. Neste experimento executamos uma tarefa de contador de palavras com um arquivo de entrada de aproximadamente 13 GB. A tarefa foi executado com um *cluster* Hadoop no cenário sem instâncias oportunistas e no cenário com instâncias oportunistas. No cenário onde utilizamos oportunismo, forçamos falha em uma das máquinas do *cluster* utilizando o *lookbusy* e aumentando a utilização de CPU da máquina física. O objetivo de forçar a falha é mostrarmos que a utilização de migração é apropriada para evitar o aumento do tempo de terminação. Na execução sem oportunismo utilizamos 4 instâncias (3 *workers* e 1 *master*), com oportunismo esse valor cresce para aproximadamente 6 instâncias (5 *workers* e 1 *master*), onde *worker* são as máquinas do *cluster* Hadoop que executam trabalho e *master* apenas coordena os *workers*. É importante notar que, na definição do *cluster* no Sahara, o tipo de máquina virtual usado para o *master* é diferente do tipo usado para os *workers*, então o *master* será hospedado em uma máquina oportunista.

A Figura 4.9 mostra que a terminação de uma instância é o pior caso que podemos esperar. Podemos ver também que existe uma redução no tempo de processamento quando

utilizamos oportunismo em relação a quando não utilizamos, isso fica mais claro na Figura 4.10, que apresenta os mesmos resultados da Figura 4.9, porém com a coluna de terminação removida. Fizemos uma análise estatística utilizando o teste *T-student* para verificar que as duas amostras não tem média semelhante e obtivemos um p-valor de $4.464 \times e^{-10}$ o que nos dá uma confiança de que o oportunismo, mesmo no cenário onde utilizamos apenas 2 instâncias oportunistas, já oferece um ganho real.

Na Figura 4.10 também observamos o resultado da execução quando utilizamos migração para evitar a perda de uma VM. Como podemos ver, o tempo de completude da tarefa utilizando migração é inferior ao caso onde não utilizamos oportunismo. Realizamos o mesmo teste estatístico e obtivemos um p-valor de $1.057 \times e{-}09$, provando estatisticamente que mesmo com migração o desempenho do processamento ainda é melhor. Observando a Figura 4.10, vemos o tempo da execução com oportunismo sem falhas e com oportunismo quando realizamos migração são bem próximos. Executamos novamente o teste T com esses dois cenários e temos um p-valor de 0.757, não podemos então rejeitar a hipótese que as duas amostras são iguais. Com isso podemos afirmar que o uso de oportunismo é capaz de melhorar o tempo de processamento de uma tarefa Hadoop e que no caso onde detectamos uma chance de falha, a utilização da migração evitará o acréscimo no tempo de processamento, tornando o uso de oportunismo transparente para o usuário.

Buscando uma visão mais real do que aconteceria em uma nuvem de produção, executamos um experimento na nuvem de produção do LSD que contém uma versão mais recente do OpenStack, *Mitaka*, do que a que executamos os experimentos anteriores. Nessa nuvem o armazenamento distribuído é feito utilizando Ceph[1] e não NFS. O Ceph é amplamente utilizado no mercado hoje, presente em cerca de 40% das nuvens OpenStack em produção no mundo [12] e a utilização dele tornaria os resultados mais próximos dos valores encontrados no mercado.

Neste experimento, utilizamos *cluster* Hadoop com oportunismo e sem oportunismo visando mostrar em um ambiente real o ganho do oportunismo para o processamento de dados. Os *clusters* com oportunismo possuem 10 máquinas trabalhando no processamento e nos *cluster* sem oportunismo temos 3. O tamanho do arquivo de entrada é de aproximadamente 5 GB e este está armazenado no *Swift*, e não no HDFS do Hadoop, situação mais comu-

[1] http://ceph.com/

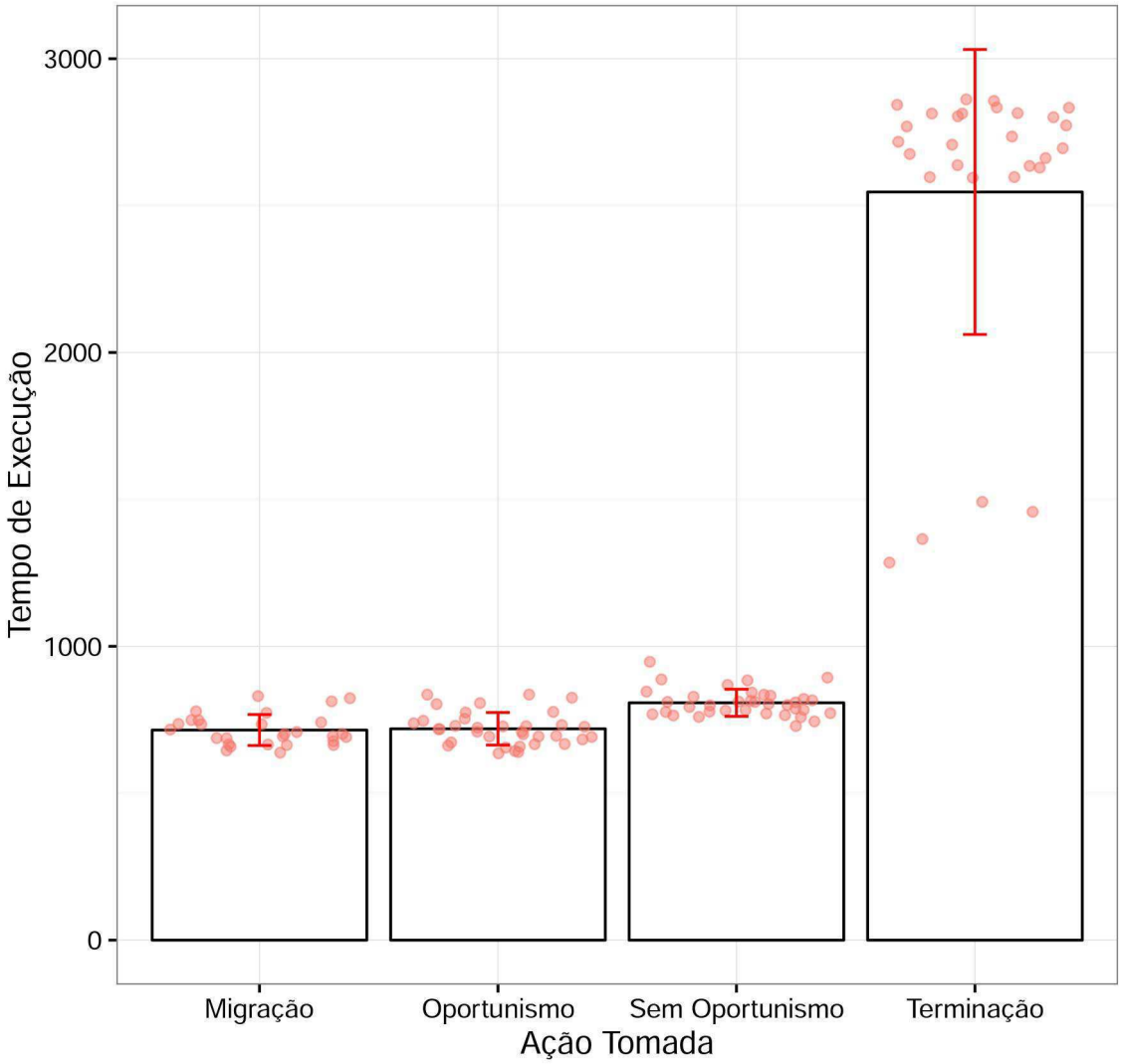

Figura 4.9: Tarefas Hadoop executadas com oportunismo com presença de falha matando a instância e migrando a instância, com oportunismo sem falha, sem oportunismo

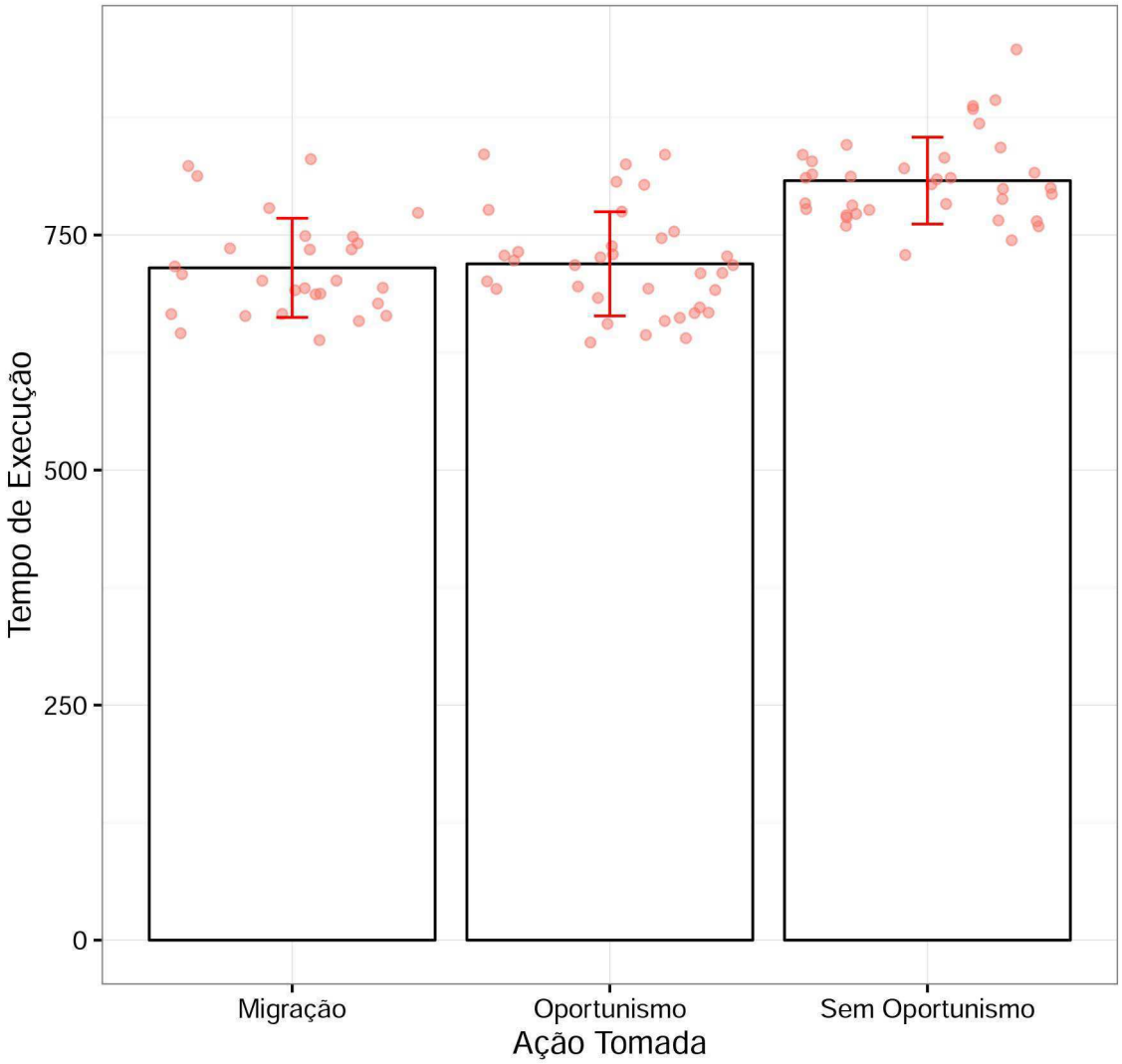

Figura 4.10: Tarefas Hadoop executadas com oportunismo com presença de falha migrando a instância, com oportunismo sem falha, sem oportunismo.

mente encontrada em nuvens computacionais com serviço de processamento de dados. Foi utilizado um arquivo de 5GB diferentemente do experimento anterior, devido a utilização de *Ceph* como armazenamento distribuído e algumas dificuldades encontradas na comunicação do mesmo com o Sahara para arquivos maiores que 5GB. Na Figura 4.11 podemos observar o desempenho das execuções e é visível o ganho que temos ao utilizarmos oportunismo. Temos em média quase metade do tempo de processamento quando utilizamos oportunismo e que, apesar do ganho ser sub-linear, é bastante significativo.

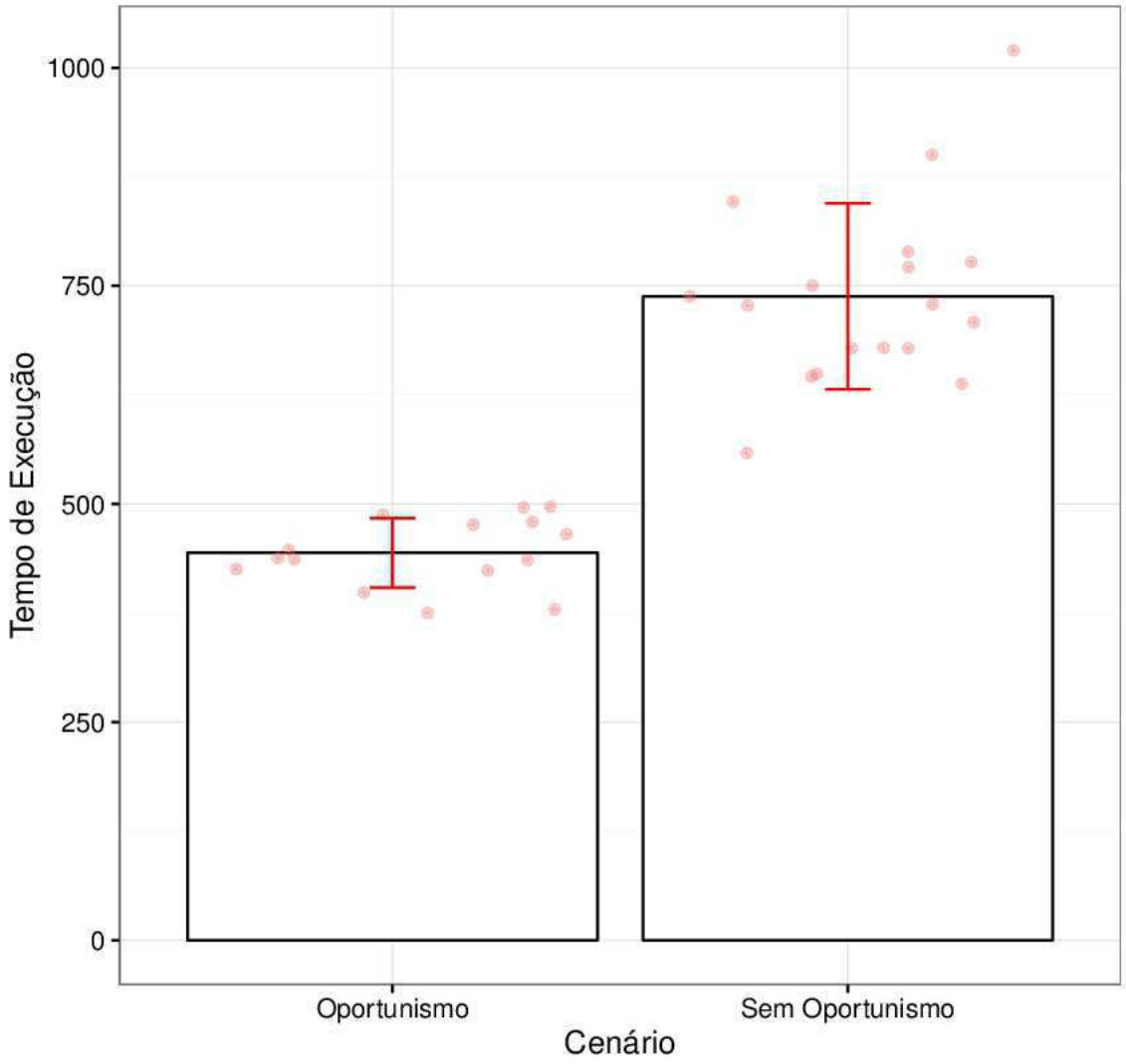

Figura 4.11: Tarefas Hadoop executadas com oportunismo e sem oportunismo em uma nuvem de produção do LSD.

Uma outra análise que fizemos foi verificar se o uso de instâncias oportunistas e o au-

mento da utilização de recursos teria um impacto em máquinas virtuais já existentes no sistema causando degradação na qualidade de serviço oferecida pelo provedor de nuvem. Durante o experimento demonstrado na Figura 4.9 e Figura 4.10 criamos duas máquinas virtuais, uma para cada servidor da nuvem, e executamos em cada um, *benchmarks* de CPU e de memória RAM. A Figura 4.12 mostra do lado esquerdo o resultado das execuções do *benchmark* de CPU separados por cenário: o cenário onde não utilizamos oportunismo; o cenário com oportunismo sem a presença de falhas e por fim o cenário com oportunismo na presença de falhas. Esse último se divide em dois cenários dependendo da ação tomada. O primeiro cenário é quando na presença da falha a VM é terminada e o segundo cenário é a migração da VM evitando a perda da mesma.

Do lado direito vemos a mesma organização para as execuções do *benchmark* de memória RAM. Podemos observar um aumento no tempo da execução do *benchmark* de CPU e memória RAM quando usamos oportunismo, porém esse aumento é bem pequeno. Vemos um aumento no cenário onde a instância oportunista é destruída. Esse aumento é esperado pois existe uma sobrecarga do próprio Hadoop para se reorganizar após uma falha e redistribuir o processamento que necessita ser refeito.

Na Figura 4.13 vemos o efeito do processamento Hadoop com instâncias oportunistas na presença de falha nas máquinas virtuais que executam os *benchmarks* de CPU e memória RAM. A linha tracejada vertical marca o tempo em que ocorreu um pico de carga de CPU o que inicia a migração ou destrói a máquina virtual, na primeira figura o pico acontece na fase *Map* e na segunda acontece na fase *Reduce*. No eixo horizontal temos cada execução do *benchmark* e vemos que o pico em si não tem um grande impacto no aumento do tempo dos *benchmarks*, porém vemos que a fase *Reduce* traz um aumento no tempo do *benchmarks*, devido a característica do Hadoop onde essa fase exige mais da CPU e da memória RAM.

Para expandir um pouco mais a avaliação da solução executamos o mesmo experimento que executamos com o contador de palavras para a tarefa do tipo *Terasort*, pois essa tarefa apresenta características um pouco diferentes do contador de palavras. Podemos ver na Figura 4.14 que o comportamento é bastante similar ao experimento executado com o contador de palavras. O pior caso se mostra quando uma instância é perdida, podendo ter um acréscimo de mais de 100% no tempo médio de execução, dependendo do tamanho da tarefa. Nos casos com oportunismo e oportunismo utilizando migração, quando existe uma

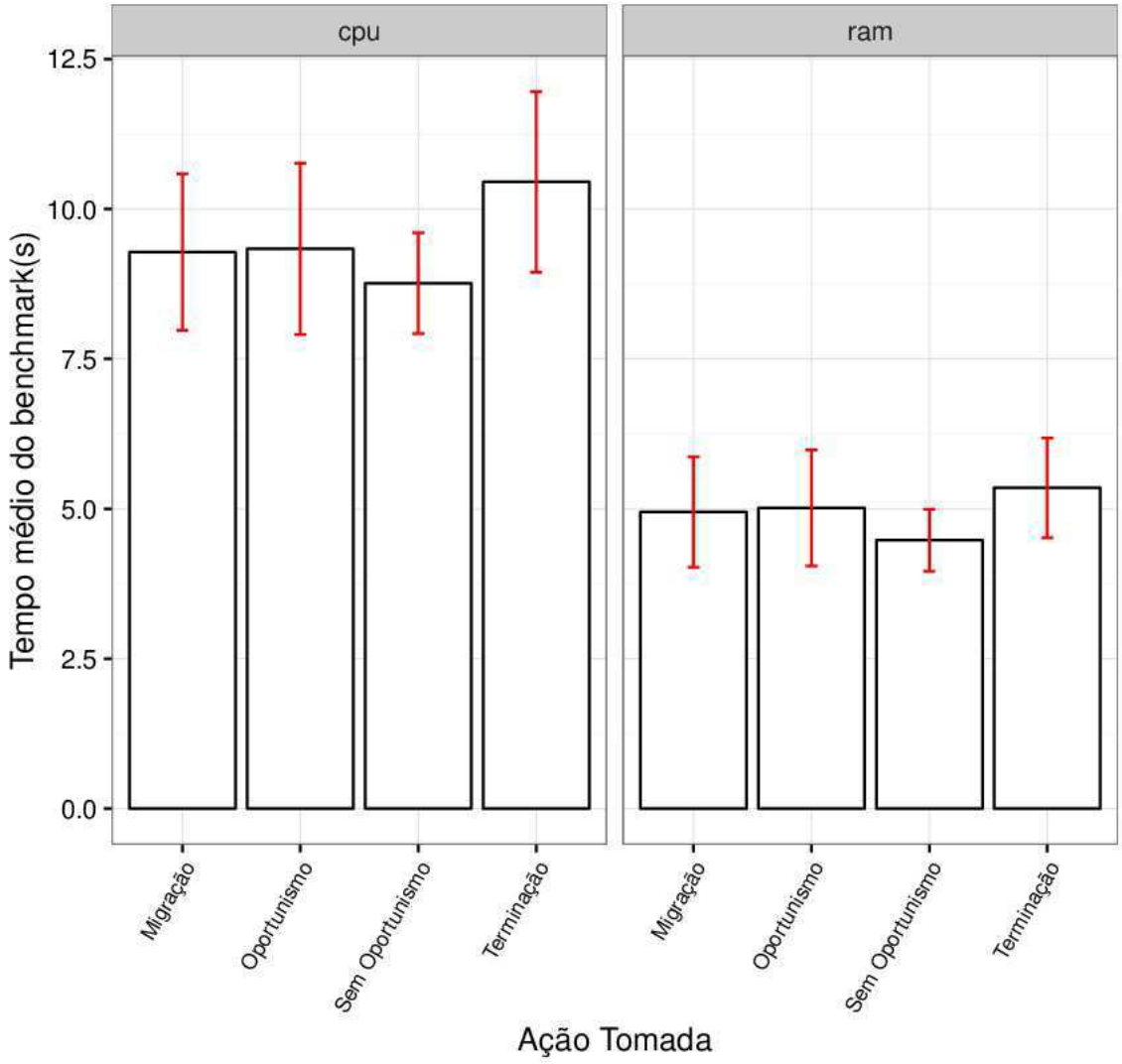

Figura 4.12: Tempo médio do *benchmark* de CPU e memória RAM medidos em máquinas virtuais em cada nó da nuvem, agrupados por ação tomada.

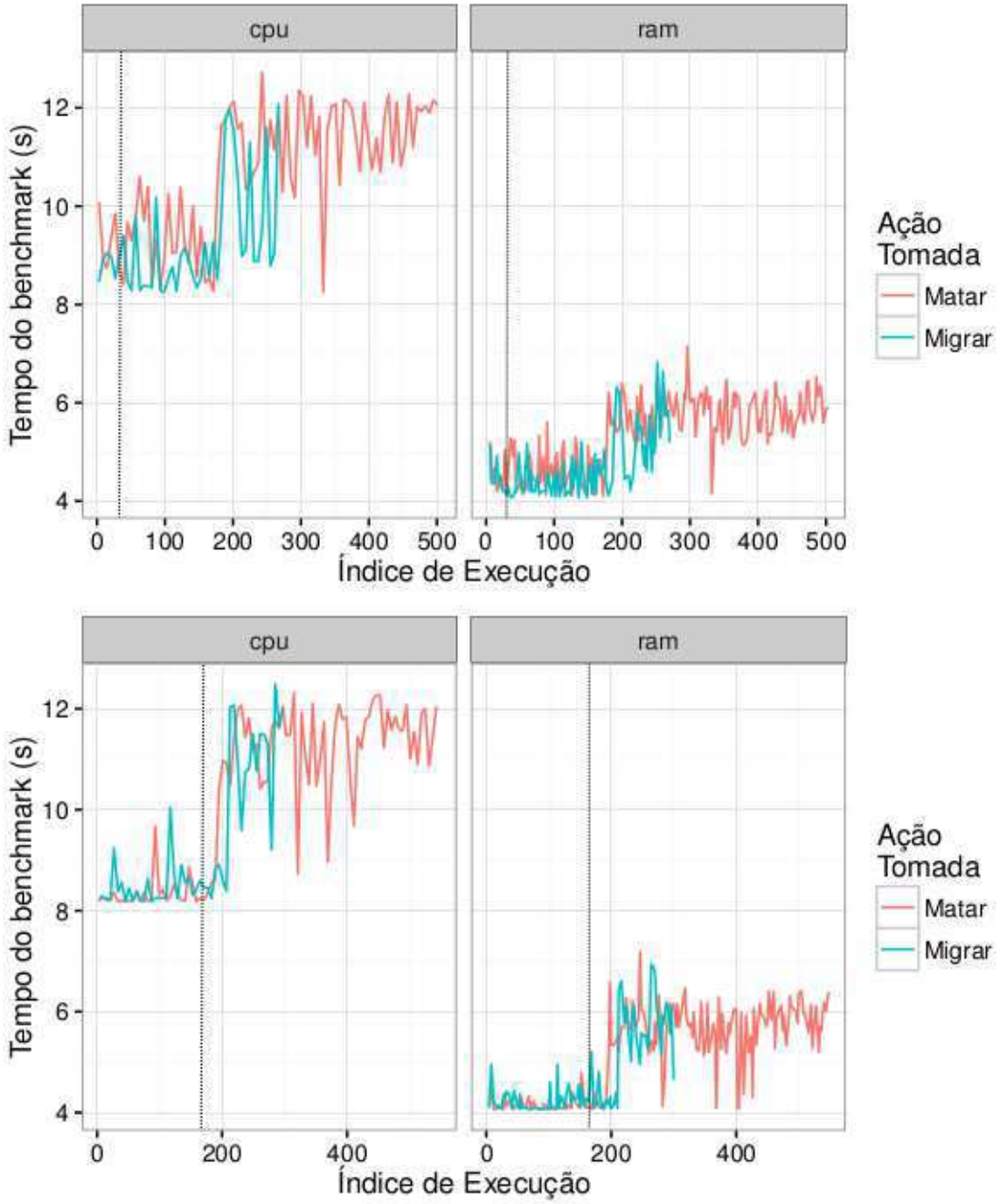

Figura 4.13: Tempo médio do *benchmark* de CPU e memória RAM medidos em máquinas virtuais em cada nó da nuvem, ao longo do tempo considerando o momento do pico de carga. Nos gráficos da parte superior da figura o pico ocorre na fase *Map*, enquanto nos gráficos da parte inferior da figura o pico ocorre na fase *Reduce*.

falha vemos um desempenho muito próximo ao caso sem instância oportunista. Isso pode ser devido a diversos fatores, por exemplo, neste experimento estávamos utilizando apenas 2 instâncias oportunistas, o que pode não mostrar um ganho muito grande. Outra explicação para estes valores próximos é o fato de, por padrão, o *Terasort* utiliza apenas um processo para o *Reduce*, ou seja, a parte do processamento que mais exige poder computacional não está paralelizada, não sendo possível perceber um ganho real com o acréscimo de poucas máquinas oportunistas. Apesar de não mostrar diminuição no tempo de processamento ao utilizar instância oportunista para essa tarefa, nós temos evidências dos experimentos anteriores, de que o ganho pode vir a acontecer com alguns detalhes de configuração da tarefa e que, no pior caso, nossa solução estará permitindo a completude da tarefa no mesmo tempo do caso em que não utilizamos oportunismo.

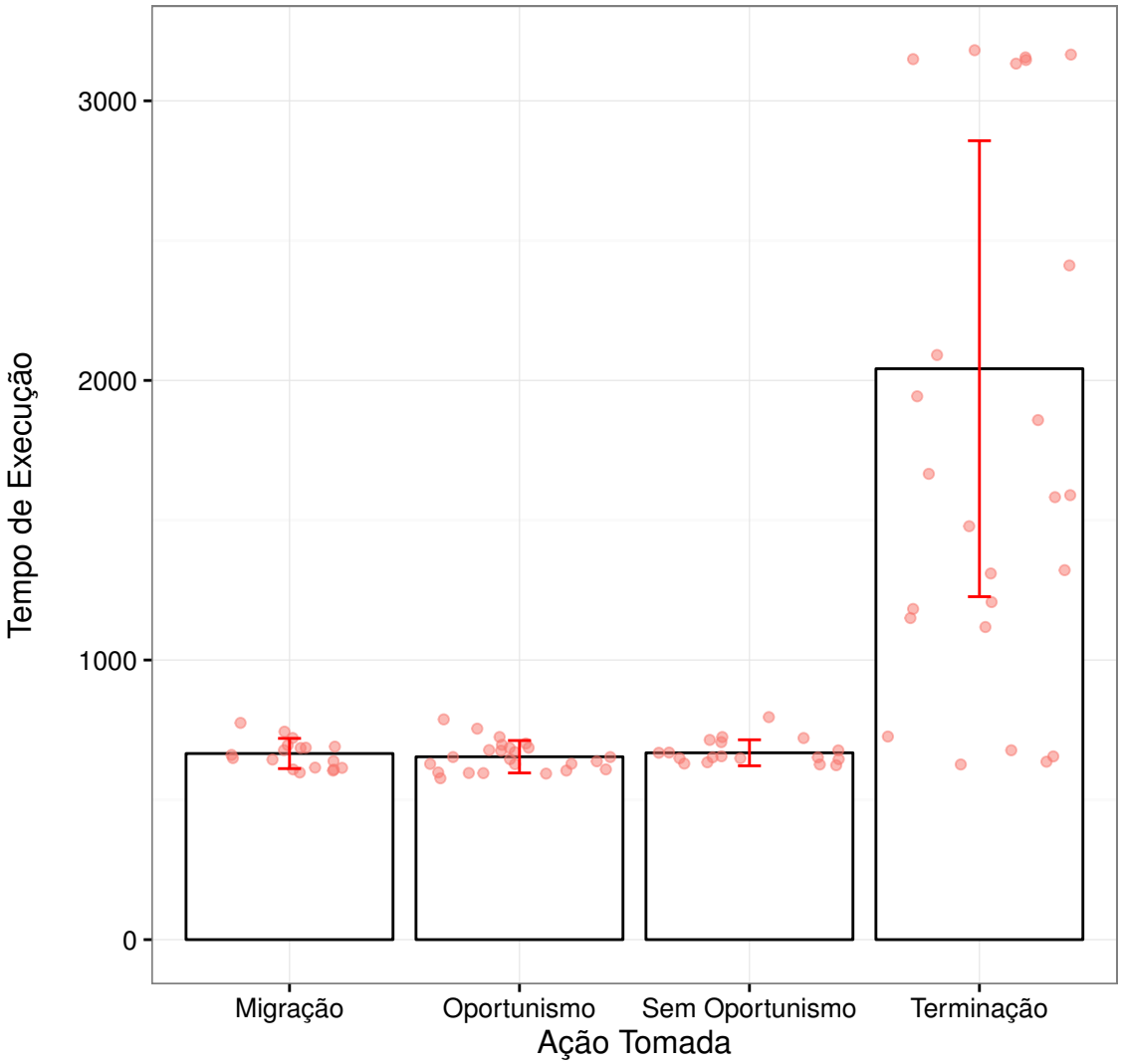

Figura 4.14: Tempo média de completude de uma tarefa Hadoop do tipo *Terasort* utilizando oportunismo nos cenários onde há falha e a instância é terminada, há falha e a instância é migrada, quando não há interferências e também sem oportunismo.

Capítulo 5

Conclusão

Neste capítulo, na Seção 5.1, sumarizamos a motivação do nosso trabalho, a solução proposta e falamos dos resultados obtidos. Já na Seção 5.2 fazemos um levantamento das limitações encontradas e possibilidades de trabalhos futuros.

5.1 Sumário

Motivado pelo problema de otimizar o processamento de dados com Hadoop na nuvem, buscamos neste trabalho propor uma nova técnica de executar tarefas Hadoop. Fazendo uma análise dos registros da nuvem privada do Google e também fundamentado na literatura, observamos que nuvens computacionais, públicas ou privadas, possuem uma taxa de utilização baixa, ficando em torno de 50% para CPU e memória RAM. Vimos também que essa baixa utilização acontece devido a recursos sob-demanda que são alocados ou reservados, mas não são de fato utilizados. Essa lacuna entre recursos reservados e recursos em uso abriu espaço para a criação de instâncias oportunistas. Instâncias oportunistas são voláteis e não são adequadas para todos os tipos de aplicação. O Hadoop, no entanto, por ser uma aplicação distribuída e que oferece tolerância a falhas, é uma aplicação que pode fazer uso desse modelo de instâncias pois a perda de uma dessas máquinas não culmina no término prematuro da aplicação.

Propomos então a utilização de instâncias oportunistas para aumentar o tamanho do *cluster* Hadoop e assim reduzir o tempo do processamento. Essa solução, apesar de parecer óbvia, nos mostrou que existe um risco de piorarmos o tempo de processamento da

tarefa Hadoop. Caso ocorra falha (perda de máquinas do *cluster*) o Hadoop leva um tempo para detectá-la e após a detecção precisa se reorganizar e reprocessar tudo que foi perdido pela máquina. Esse tempo de recuperação e reprocessamento aumenta em torno de 50% no tempo total de processamento para a maioria das tarefas Hadoop.

Considerando a utilização de oportunismo para otimizar processamento de dados com Hadoop na nuvem e que instâncias oportunistas, no seu modelo comum, podem ser prejudiciais para o processamento de dados, propomos a criação de um novo modelo de instâncias oportunistas, chamadas de instâncias oportunistas com qualidade de serviço. Essas instâncias são criadas baseadas nos mesmos recursos que as instâncias oportunistas normais, porém a quantidade de instâncias que são criadas é calculada com base em uma predição de utilização da nuvem. Ao ser solicitada a criação de um *cluster* pelo usuário, são utilizados dados históricos da nuvem e um preditor composto por 4 preditores comumente usados para a predição da utilização de máquinas físicas da nuvem em intervalos de 15 minutos, ajudando a garantir que as instâncias oportunistas não serão perdidas naquele período. Como a maioria das tarefas Hadoop tem em média tempo de duração de de 20 a 30 minutos [34], prevemos duas janelas de 15 minutos antes da criação das instâncias.

Mesmo utilizando instâncias oportunistas com qualidade de serviço, é evidente que o risco de perda de máquinas oportunistas ainda existe. Cargas inesperadas, erros do preditor e fatores externos podem culminar para um aumento inesperado de utilização e a necessidade de terminar uma instâncias oportunista. Como meio de contornar esse risco, propomos o uso de uma técnica conhecida no meio do gerenciamento de recursos, a migração de máquinas virtuais. Na nossa solução utilizamos migração viva pois o efeito dessa migração se mostrou estatisticamente transparente no nosso ambiente para o Hadoop, enquanto a migração comum, por causa da paralisação e desligamento da instância antes de migração, tem um efeito similar à terminação da instância.

Para validar nossa proposta realizamos alguns experimentos utilizando a nuvem privada do Laboratório de Sistemas Distribuídos. Como essa nuvem possui uma utilização bastante restrita, geramos uma carga sintética de CPU e memória RAM para cada servidor, com base nos dados do Google. Nesses experimentos executamos a tarefa contador de palavras com arquivos de entrada de 13 GB e variamos o comportamento da nuvem forçando a situações extremas e coletamos dados como tempo total da tarefa e impacto da carga extra em máquinas

virtuais já presentes.

Podemos ver nos resultados dos experimentos que instâncias oportunistas são capazes de otimizar o processamento de dados com Hadoop na nuvem, porém é necessário que seja evitada que a máquina virtual seja destruída. Para isso fizemos uma estimativa de recursos utilizáveis a partir dos resultados de um preditor de carga. Usamos também a migração, que se mostrou transparente para o Hadoop, não trazendo acréscimo ao tempo de processamento e mostrando um tempo 70% menor em relação ao cenário onde instâncias são perdidas e também observamos uma redução de 39,8% em relação ao tempo original de processamento.

Finalmente, avaliamos o impacto que o uso de oportunismo poderia causar em outras instâncias em execução na nuvem. O resultado observado foi que, com o uso de migração, a utilização instâncias oportunistas não afeta o desempenho de outras instâncias executando na mesma máquina física. Esses resultados evidenciam a existência de espaço para o Hadoop se desenvolver no ambiente de nuvem. Com novos conceitos criados e novos modelos de utilização recursos ainda podemos fazer muito para otimizar Hadoop, diminuindo o custo para o usuário e melhorando a eficiência da nuvem por meio de uma maior utilização de recursos.

5.2 Trabalhos futuros e limitações

Apesar de se mostrar favorável e dos resultados bem sucedidos, há algumas considerações que podem ser feitas com objetivo de tornar a solução mais robusta e de aplicação mais ampla.

A primeira limitação que encontramos é com relação ao preditor de carga. O preditor utilizado na nossa validação é bastante simples, e que não faz nenhuma análise profunda dos dados históricos. Com um preditor mais robusto podemos ter mais certeza da quantidade de recursos disponíveis e, consequentemente, usar valores menos conservadores.

Outra limitação encontrada foi a execução dos experimentos utilizando um tamanho único de entrada e também um tipo único de tarefa, contador de palavras. Apesar de termos testado o *Terasort* também, esse teste foi superficial e um aprofundamento em variações de tamanhos de entrada, tipos de tarefas e tamanhos de *clusters* pode nos trazer dados interessantes que não foram vistos até o momento. Esses experimentos não foram realizados

devido a limitações de *hardware* e escalonamento da nuvem privada do laboratório entre vários usuários, e seria útil fazê-los utilizando, principalmente, tarefas que consomem diferentes recursos da nuvem, como por exemplo, o *Pi Estimator* que é uma aplicação que faz uso majoritariamente de CPU.

Implementamos, em forma de prova de conceito, o componente proposto onde é feita a predição de recursos para criação de *clusters*, mais detalhes sobre essa implementação podem ser vistos no Apêndice A. Experimentamos com essa implementação com objetivo de mostrar a utilização do preditor para criação de *clusters* Hadoop. Esses experimentos foram bastante limitados pois apenas buscamos mostrar o desempenho do preditor, e temos como objetivos futuros levar em consideração todos os conceitos apresentados, o preditor, o conceito de migração e também queremos levar esses experimentos para uma nuvem real, onde teremos mais variações de utilização de recursos e podermos avaliar como se comporta a solução de uma forma mais completa. Observamos também na nossa implementação que a presença de duas requisições simultâneas serão tratadas da mesma maneira e isso pode acarretar em erros de predição. Queremos aprimorar nossa solução para identificar a quantidade de solicitações de criação de *cluster* e poder assim, repassar a informação correta de quantidade disponível de recurso para cada solicitação.

Bibliografia

[1] Amazon ec2. https://aws.amazon.com/pt/ec2/, dec 2015.

[2] Amazon ec2 spot instances. http://aws.amazon.com/ec2/purchasing-options/spot-instances/, feb 2015.

[3] Amazon emr. http://aws.amazon.com/elasticmapreduce/, feb 2015.

[4] Apache storm. http://storm.apache.org/, feb 2015.

[5] Openstack cloud computing software. http://www.openstack.org/, dec 2015.

[6] Welcome to apache™ hadoop®! http://hadoop.apache.org/, feb 2015.

[7] Amazon ec2 service level agreement. https://aws.amazon.com/ec2/sla/, 2016.

[8] Apache spark - lightning-fast cluster computing. http://spark.apache.org/, jun 2016.

[9] Google cluster data. https://github.com/google/cluster-data, jul 2016.

[10] lookbusy - a synthetic load generator. https://www.devin.com/lookbusy/, 2016.

[11] Openstack cloud computing software user stories. https://www.openstack.org/user-stories/, jun 2016.

[12] Openstack user survey. https://www.openstack.org/user-survey/survey-2016-q1/landing, 2016.

[13] Sysstat - performance monitoring tools for linux. http://sebastien.godard.pagesperso-orange.fr/, 2016.

[14] Raja Appuswamy, Christos Gkantsidis, Dushyanth Narayanan, Orion Hodson, and Antony Rowstron. Scale-up vs Scale-out for Hadoop: Time to rethink? In *ACM Symposium on Cloud Computing*, 2013.

[15] Luiz André Barroso and Urs Hölzle. The case for energy-proportional computing. *Computer*, 40(12):33–37, 2007.

[16] Salman A Baset, Long Wang, and Chunqiang Tang. Towards an understanding of oversubscription in cloud. 2012.

[17] Orna Agmon Ben-Yehuda, Muli Ben-Yehuda, Assaf Schuster, and Dan Tsafrir. The Rise of RaaS: The Resource-as-a-Service Cloud. *Analytical Chemistry*, 29(7):5A–5A, jul 1957.

[18] G E P Box and G M Jenkins. *Time Series Analysis: Forecasting and Control*, volume Third. 1994.

[19] David Breitgand, Zvi Dubitzky, Amir Epstein, Alex Glikson, and Inbar Shapira. SLA-aware Resource Over-Commit in an IaaS Cloud. In *8th International Conference on Network and Service Managementrk and Service Management*, pages 73–81, 2012.

[20] KP K.P. Burnham and D.R. DR Anderson. *Model Selection and Multimodel Inference: A Practical Information-Theoretic Approach (2nd ed)*, volume 172. 2002.

[21] Marcus Carvalho, Francisco Brasileiro, and John Wilkes. Long-term SLOs for reclaimed cloud computing resources.

[22] C Chatfield. *The Analysis of Time Series: An Introduction*, volume 59. 2004.

[23] Navraj Chohan, Claris Castillo, Mike Spreitzer, Malgorzata Steinder, Asser Tantawi, Chandra Krintz, and Santa Barbara. See Spot Run: Using Spot Instances for MapReduce Workflows. In *2nd USENIX Workshop on Hot Topics in Cloud Computing, HotCloud*, 2010.

[24] Antonio Corradi, Mario Fanelli, and Luca Foschini. VM consolidation: A real case based on OpenStack Cloud. *Future Generation Computer Systems*, 32:118–127, 2014.

[25] J Dean and S Ghemawat. MapReduce : Simplified Data Processing on Large Clusters. *Communications of the ACM*, 51:1–13, 2008.

[26] Dave Durkee. Why cloud computing will never be free, 2010.

[27] Fahimeh Farahnakian, Tapio Pahikkala, Pasi Liljeberg, and Juha Plosila. Energy aware consolidation algorithm based on K-nearest neighbor regression for cloud data centers. In *Proceedings - 2013 IEEE/ACM 6th International Conference on Utility and Cloud Computing, UCC 2013*, pages 256–259, 2013.

[28] Rahul Ghosh and Vijay K. Naik. Biting off Safely More than You Can Chew: Predictive analytics for resource over-commit in IaaS cloud. In *Proceedings - 2012 IEEE 5th International Conference on Cloud Computing, CLOUD 2012*, pages 25–32, 2012.

[29] Rachel Householder, Scott Arnold, and Robert Green. On Cloud-based Oversubscription. *International Journal of Engineering Trends and Technology*, 8(8):425–431, 2014.

[30] Nikolaus Huber, Marcel von Quast, Michael Hauck, and Samuel Kounev. Evaluating and Modeling Virtualization Performance Overhead for Cloud Environments. *Proceedings of the 1st International Conference on Cloud Computing and Services Science (CLOSER 2011)*, pages 563–573, 2011.

[31] Rob J. Hyndman and Yeasmin Khandakar. Automatic time series forecasting: The forecast package for R. *Journal Of Statistical Software*, 27:C3–C3, 2008.

[32] Sadeka Islam, Jacky Keung, Kevin Lee, and Anna Liu. Empirical prediction models for adaptive resource provisioning in the cloud. In *Future Generation Computer Systems*, volume 28, pages 155–162, 2012.

[33] Selvi Kadirvel, Jeffrey Ho, and José A. B. Fortes. Fault management in map-reduce through early detection of anomalous nodes. In *10th International Conference on Autonomic Computing, ICAC'13, San Jose, CA, USA, June 26-28, 2013*, pages 235–245, 2013.

[34] Soila Kavulya, Jiaqi Tany, Rajeev Gandhi, and Priya Narasimhan. An analysis of traces from a production MapReduce cluster. In *CCGrid 2010 - 10th IEEE/ACM International Conference on Cluster, Cloud, and Grid Computing*, pages 94–103, 2010.

[35] Steven Y Ko and Brian Cho. On Availability of Intermediate Data in Cloud Computations. *Solutions*, pages 6–6, 2009.

[36] Young Choon Lee, Albert Y. Zomaya, Young Choon, and Lee Albert. Energy efficient utilization of resources in cloud computing systems. *The Journal of Supercomputing*, 60:268–280, 2012.

[37] Chin-hung Li. Evaluating the Effectiveness of Memory Overcommit Techniques on KVM-based Hosting Platform. pages 622–626, 2012.

[38] Heshan Lin, Xiaosong Ma, Jeremy Archuleta, Wu-chun Feng, Mark Gardner, and Zhe Zhang. MOON: MapReduce On Opportunistic eNvironments. *Proc. of the HPDC - Intl. Symp. on High Performance Distributed Computing*, pages 95–106, 2010.

[39] Heshan Lin, Xiaosong Ma, and Wu chun Feng. Reliable MapReduce computing on opportunistic resources. *Cluster Computing*, 15:145–161, 2012.

[40] Huan Liu and Accenture Technology Labs. Cutting MapReduce Cost with Spot Market. In *USENIX Workshop on Hot Topics in Cloud Computing (HotCloud)*, 2011.

[41] Paul Marshall, Kate Keahey, and Tim Freeman. Improving utilization of infrastructure clouds. In *Proceedings - 11th IEEE/ACM International Symposium on Cluster, Cloud and Grid Computing, CCGrid 2011*, pages 205–214, 2011.

[42] Fabio Jorge Almeida Morais, Francisco Vilar Brasileiro, Raquel Vigolvino Lopes, Ricardo Araujo Santos, Wade Satterfield, and Leandro Rosa. Autoflex: Service Agnostic Auto-scaling Framework for IaaS Deployment Models. *2013 13th IEEE/ACM International Symposium on Cluster, Cloud, and Grid Computing*, pages 42–49, may 2013.

[43] George Pallis. Cloud computing: The new frontier of internet computing, 2010.

[44] Charles Reiss, Gregory R Ganger, Randy H Katz, Michael A Kozuch, and Alexey Tumanov. Heterogeneity and Dynamicity of Clouds at Scale: Google Trace Analysis. In *IEEE SOC Conference*, page 13, 2012.

[45] Nilabja Roy, Abhishek Dubey, and Aniruddha Gokhale. Efficient Autoscaling in the Cloud Using Predictive Models for Workload Forecasting. *2011 IEEE 4th International Conference on Cloud Computing*, pages 500–507, July 2011.

[46] Jonhnny Weslley Silva, Thiago Emmanuel Pereira, Francisco Brasileiro, Universidade Federal, and De Campina Grande. Computação intensiva em dados com MapReduce em Computação a ambientes oportunistas. In *SBRC 2010*, pages 279–292, 2010.

[47] Shekhar Srikantaiah, a Kansal, and F Zhao. Energy Aware Consolidation for Cloud Computing. *Scenario*, page 10, 2008.

[48] Dawei Sun, Guangyan Zhang, Songlin Yang, Weimin Zheng, Samee U. Khan, and Keqin Li. Re-Stream: Real-time and energy-efficient resource scheduling in big data stream computing environments. *Information Sciences*, mar 2015.

[49] Chandramohan A. Thekkath, John Wilkes, and Edward D. Lazowska. Techniques for File System Simulation. *Software - Practice and Experience*, 24:981–999, 1994.

[50] Fengguang Tian and Keke Chen. Towards optimal resource provisioning for Running MapReduce programs in public clouds. In *Proceedings - 2011 IEEE 4th International Conference on Cloud Computing, CLOUD 2011*, pages 155–162, 2011.

[51] Luis Tomás and Johan Tordsson. Improving cloud infrastructure utilization through overbooking. In *Proceedings of the 2013 ACM Cloud and Autonomic Computing Conference on - CAC '13*, page 1, New York, New York, USA, 2013. ACM Press.

[52] William Voorsluys and Rajkumar Buyya. Reliable provisioning of spot instances for compute-intensive applications. In *Proceedings - International Conference on Advanced Information Networking and Applications, AINA*, pages 542–549, 2012.

[53] Rafael Weingärtner, Gabriel Beims Bräscher, and Carlos Becker Westphall. Cloud resource management: A survey on forecasting and profiling models. *Journal of Network and Computer Applications*, 47:99–106, January 2015.

[54] Timothy Wood, Prashant Shenoy, Arun Venkataramani, and Mazin Yousif. Sandpiper: Black-box and gray-box resource management for virtual machines. *Computer Networks*, 53(17):2923–2938, 2009.

[55] Jiong Xie Jiong Xie, Shu Yin Shu Yin, Xiaojun Ruan Xiaojun Ruan, Zhiyang Ding Zhiyang Ding, Yun Tian Yun Tian, J. Majors, A. Manzanares, and Xiao Qin Xiao Qin. Improving MapReduce performance through data placement in heterogeneous Hadoop clusters. *Parallel & Distributed Processing, Workshops and Phd Forum (IPDPSW), 2010 IEEE International Symposium on*, 2010.

[56] Fei Xu, Fangming Liu, Linghui Liu, Hai Jin, Bo Li, and Baochun Li. iAware: Making Live Migration of Virtual Machines Interference-Aware in the Cloud. *IEEE Transactions on Computers*, 63(12):3012–3025, 2014.

[57] Kejiang Ye, Xiaohong Jiang, Ran Ma, and Fengxi Yan. VC-migration: Live migration of virtual clusters in the cloud. *Proceedings - IEEE/ACM International Workshop on Grid Computing*, pages 209–218, 2012.

[58] Qi Zhang, Lu Cheng, and Raouf Boutaba. Cloud computing: State-of-the-art and research challenges. *Journal of Internet Services and Applications*, 1(1):7–18, 2010.

[59] Han Zhao, Miao Pan, Xinxin Liu, Xiaolin Li, and Yuguang Fang. Optimal Resource Rental Planning for Elastic Applications in Cloud Market. *2012 IEEE 26th International Parallel and Distributed Processing Symposium*, pages 808–819, may 2012.

[60] Qin Zheng. Improving MapReduce fault tolerance in the cloud. In *Proceedings of the 2010 IEEE International Symposium on Parallel and Distributed Processing, Workshops and Phd Forum, IPDPSW 2010*, 2010.

F

Apêndice A

Implementação

Implementamos, como uma prova de conceito, uma primeira versão do componente que é responsável por se comunicar com o Sahara para criação. Este código foi disponibilizado [1] e está em uma versão funcional e experimentada. A Figura A.1 mostra a arquitetura da implementação e a comunicação entre os componentes.

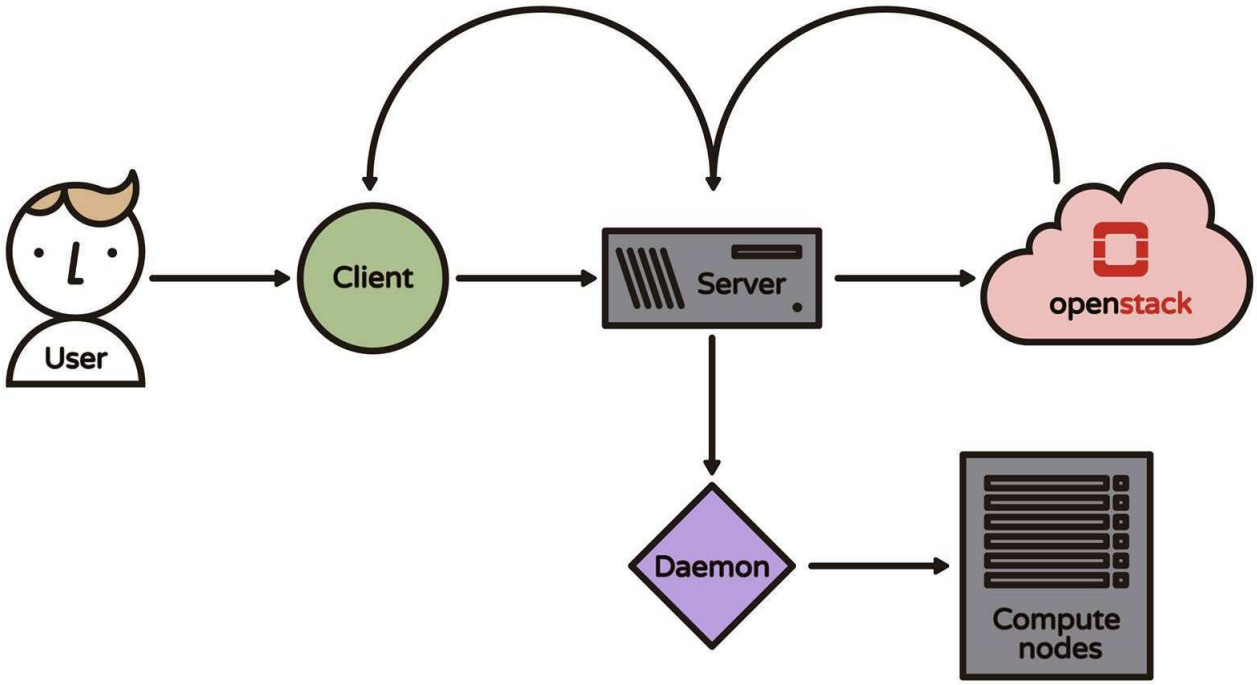

Figura A.1: Arquitetura da implementação da nossa solução.

[1] https://git.lsd.ufcg.edu.br/tellesnobrega/opportunistic-hadoop

A implementação foi dividida em 3 componentes, são eles:

- *Daemon* - responsável pela coleta de informações de utilização de CPU e memória RAM.
- *Server* - responsável pela parte de predição e comunicação com o Sahara.
- *Client* - responsável por receber o pedido do usuário e enviar para o *Server*.

O *Daemon* coleta as informações de utilização em intervalos de 5 minutos, e armazena em um arquivo de texto. Esse arquivo de texto é utilizado pelo preditor como entrada para predição da carga futura.

O fluxo de execução é:

1. Usuário solicita a criação de um *cluster* Hadoop passando o tamanho mínimo do *cluster* no *Client*;
2. O *Server* recebe o pedido do usuário, e busca o arquivo com histórico de utilização. Esse arquivo é passado para o preditor que retorna o número de máquinas disponíveis para criação de um *cluster*.
3. Ainda no *Server*, é solicitado a criação do *cluster* com o novo tamanho.

É importante notar que o Sahara e o OpenStack em geral não conhecem o conceito de instâncias oportunistas, então nessa implementação consideramos o seguinte cenário. Quando solicitado a criação de um *cluster*, é verificada a possibilidade de se utilizar oportunismo. Se for possível, o *cluster* é criado utilizando um novo usuário, chamado de *sahara-oi*, assim nós aplicamos o conceito de oportunismo, pois as cotas do usuário não são reduzidas.

Apêndice B

Resultados do preditor

Neste apêndice, mostramos os resultados dos preditores em diversas outras máquinas da nuvem do Google. Aqui serão mostrados resultados do preditor utilizando o valor previsto e também do preditor pessimista, que utiliza o valor mais alto no intervalo de confiança da predição. Também serão mostrados os gráficos do erros de predição para os cenários mostrados.

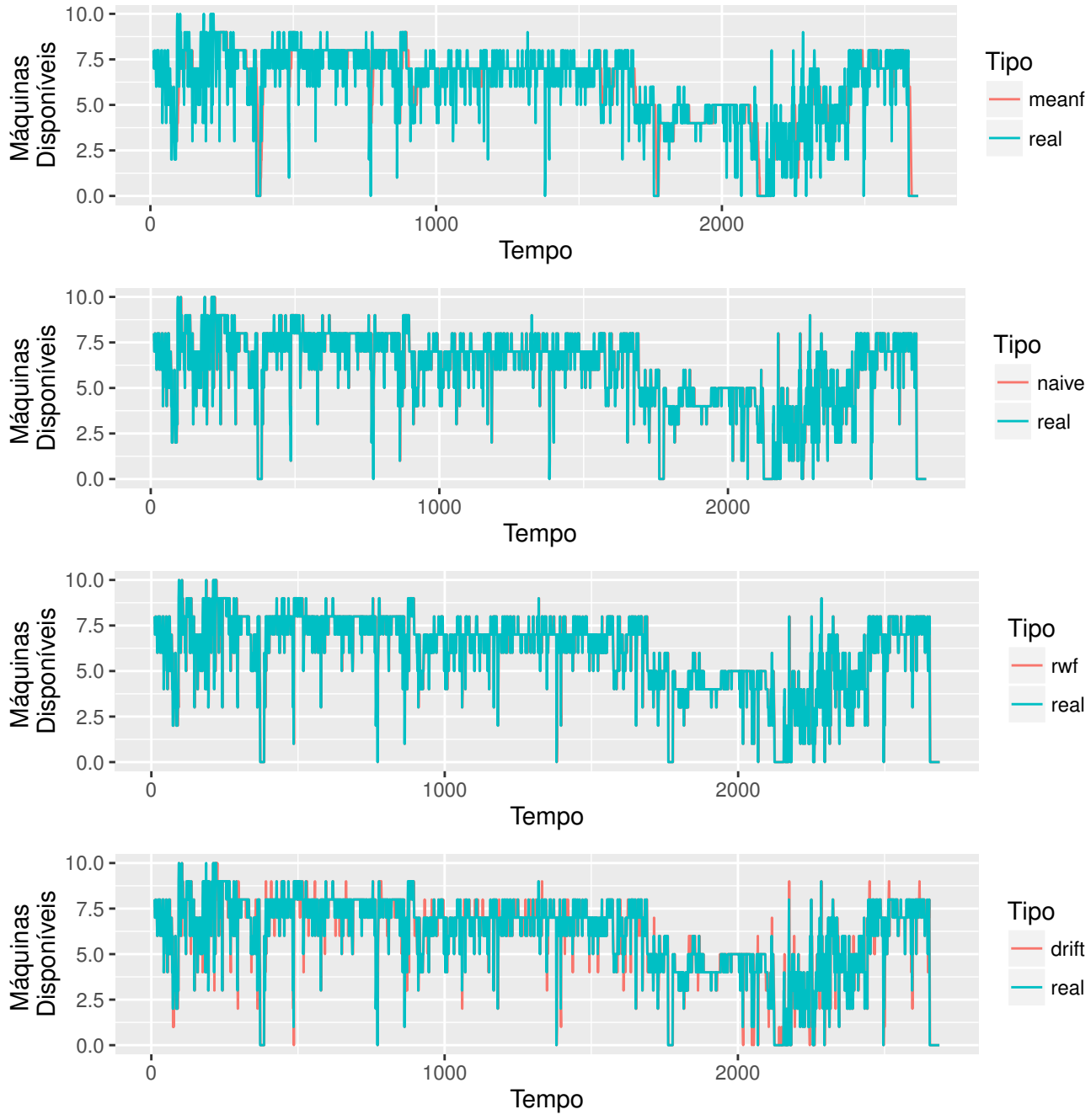

Figura B.1: Predição de recursos ociosos por preditor para máquina com ID 97959424 do *cluster* do Google com predição de 1 janela de 15 minutos no futuro.

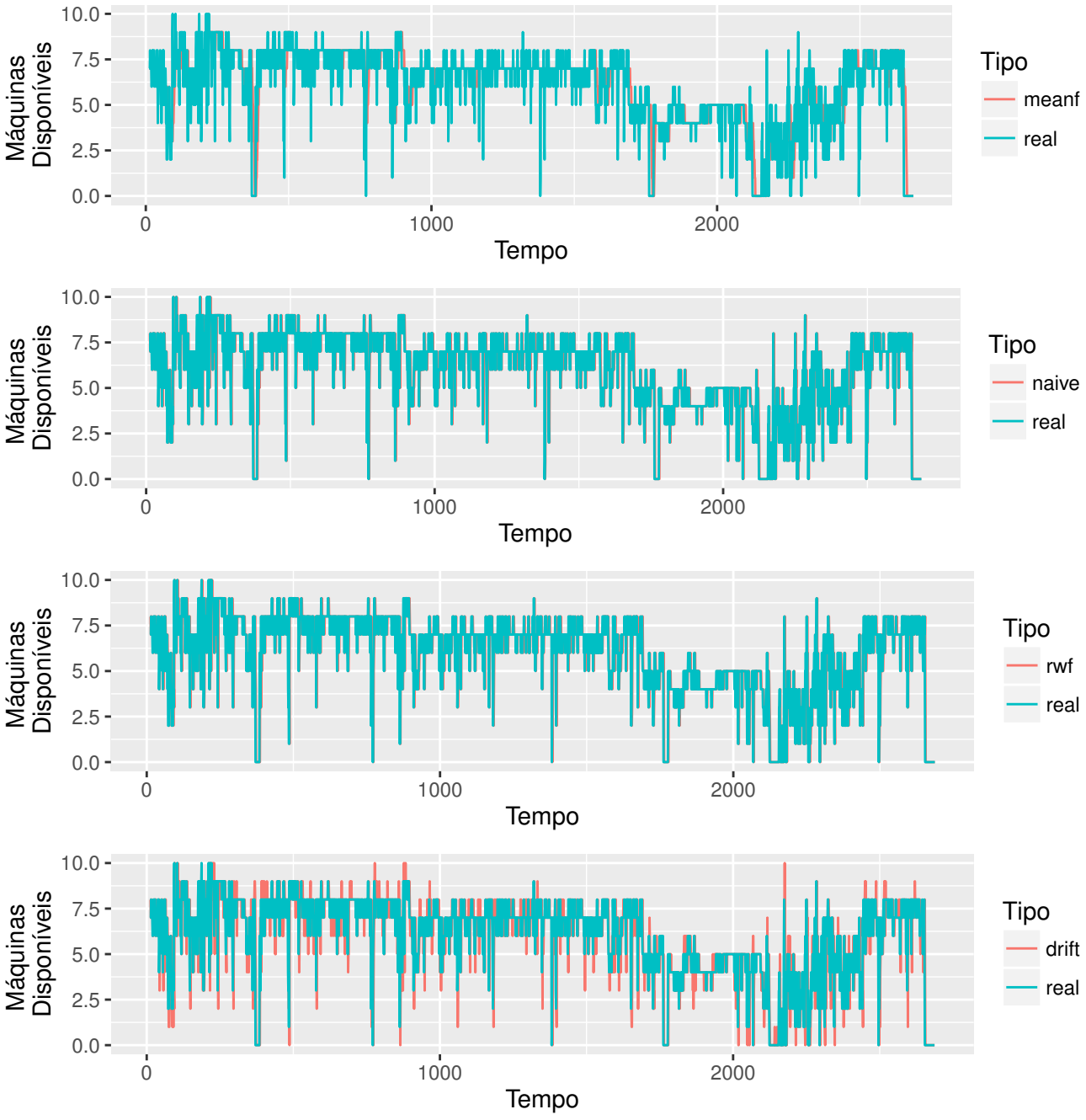

Figura B.2: Predição de recursos ociosos por preditor para máquina com ID 97959424 do *cluster* do Google com predição de 2 janela de 15 minutos no futuro.

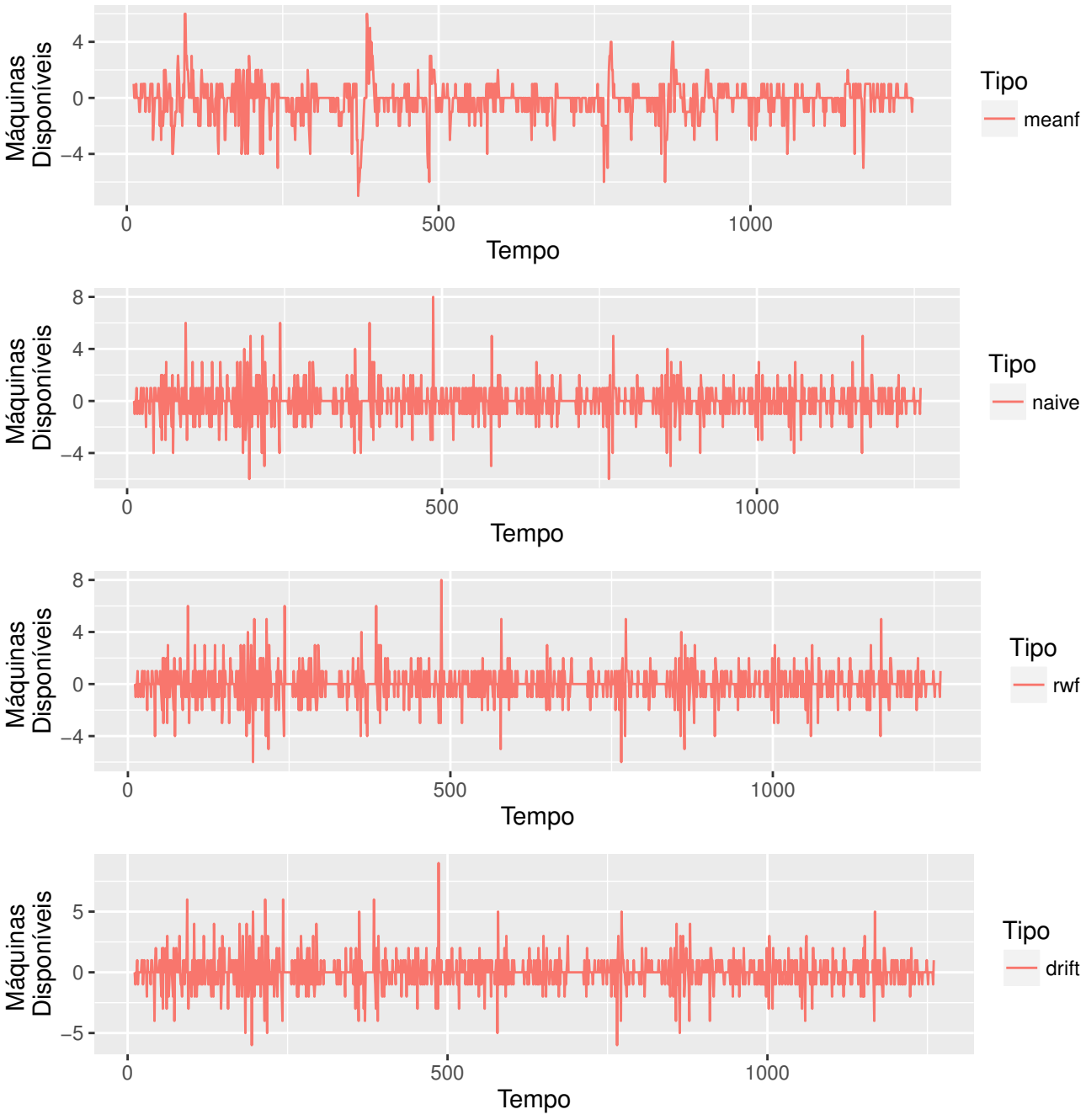

Figura B.3: Erros de predição por preditor para a máquina com ID 97959424 do *cluster* do Google com 1 janela de predição de 15 minutos no futuro.

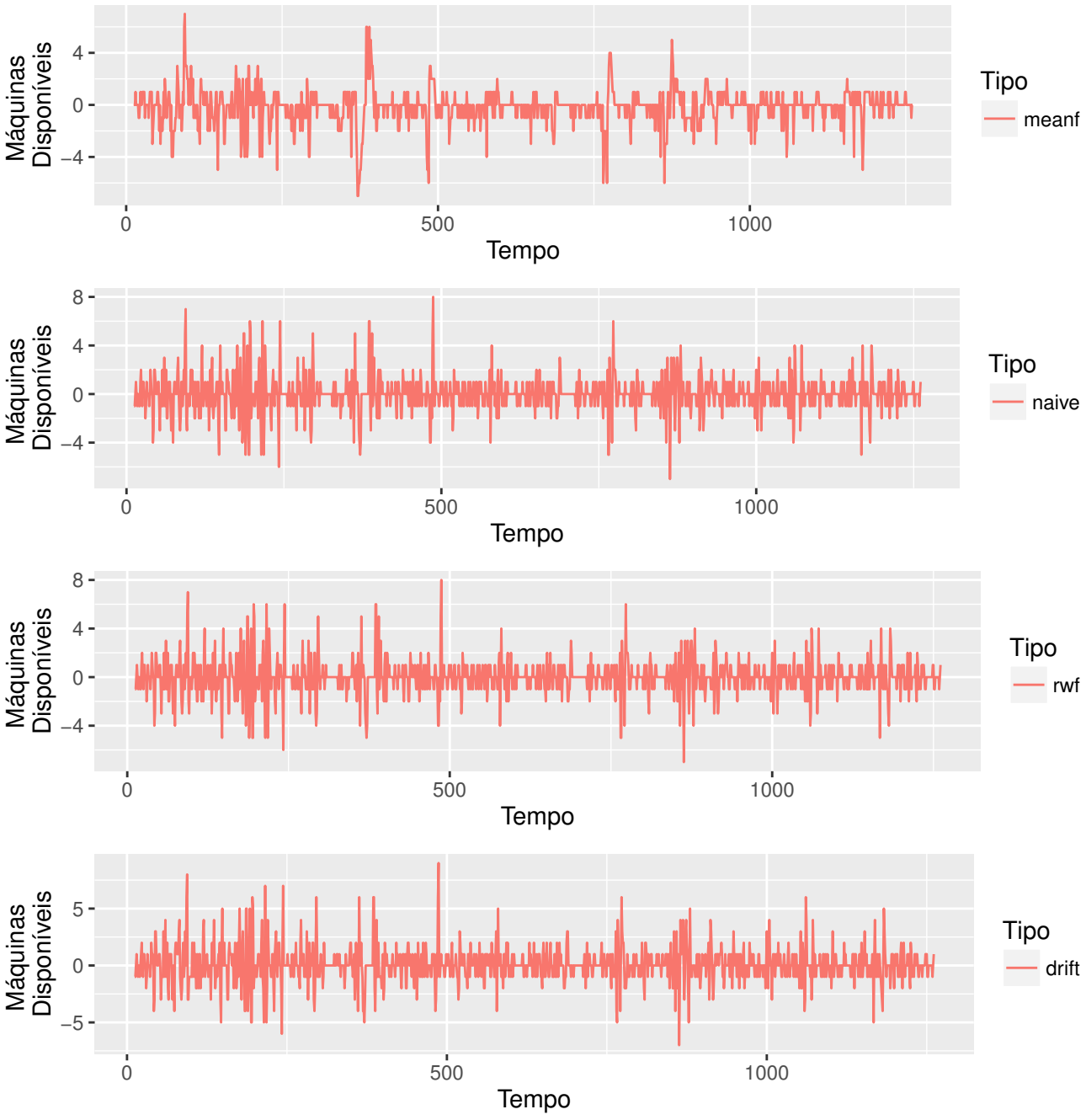

Figura B.4: Erros de predição por preditor para a máquina com ID 97959424 do *cluster* do Google com 2 janelas de predição de 15 minutos no futuro.

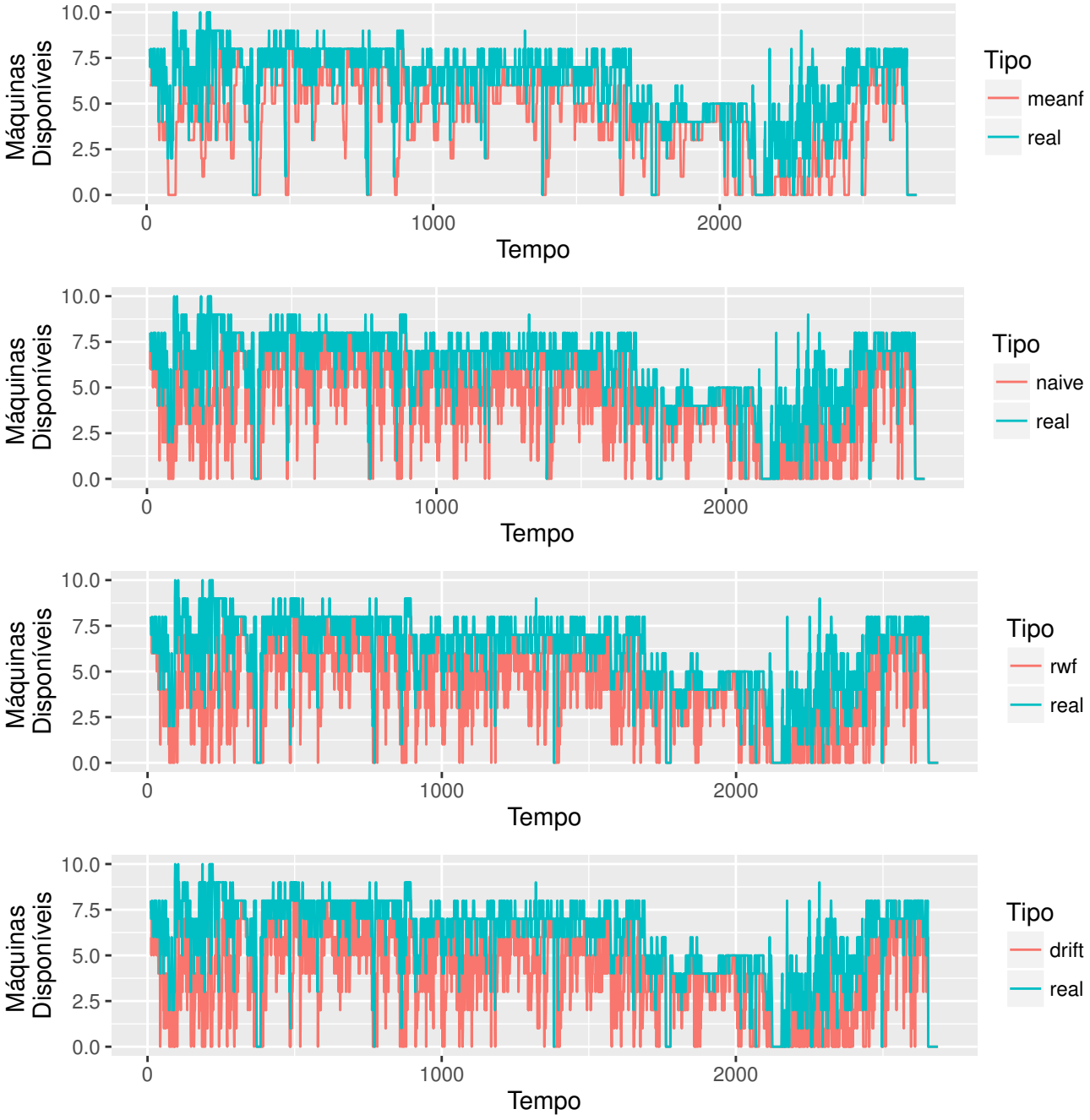

Figura B.5: Predição de recursos ociosos por preditor para máquina com ID 97959424 do *cluster* do Google com predição de 1 janela de 15 minutos no futuro, utilizando preditor pessimista.

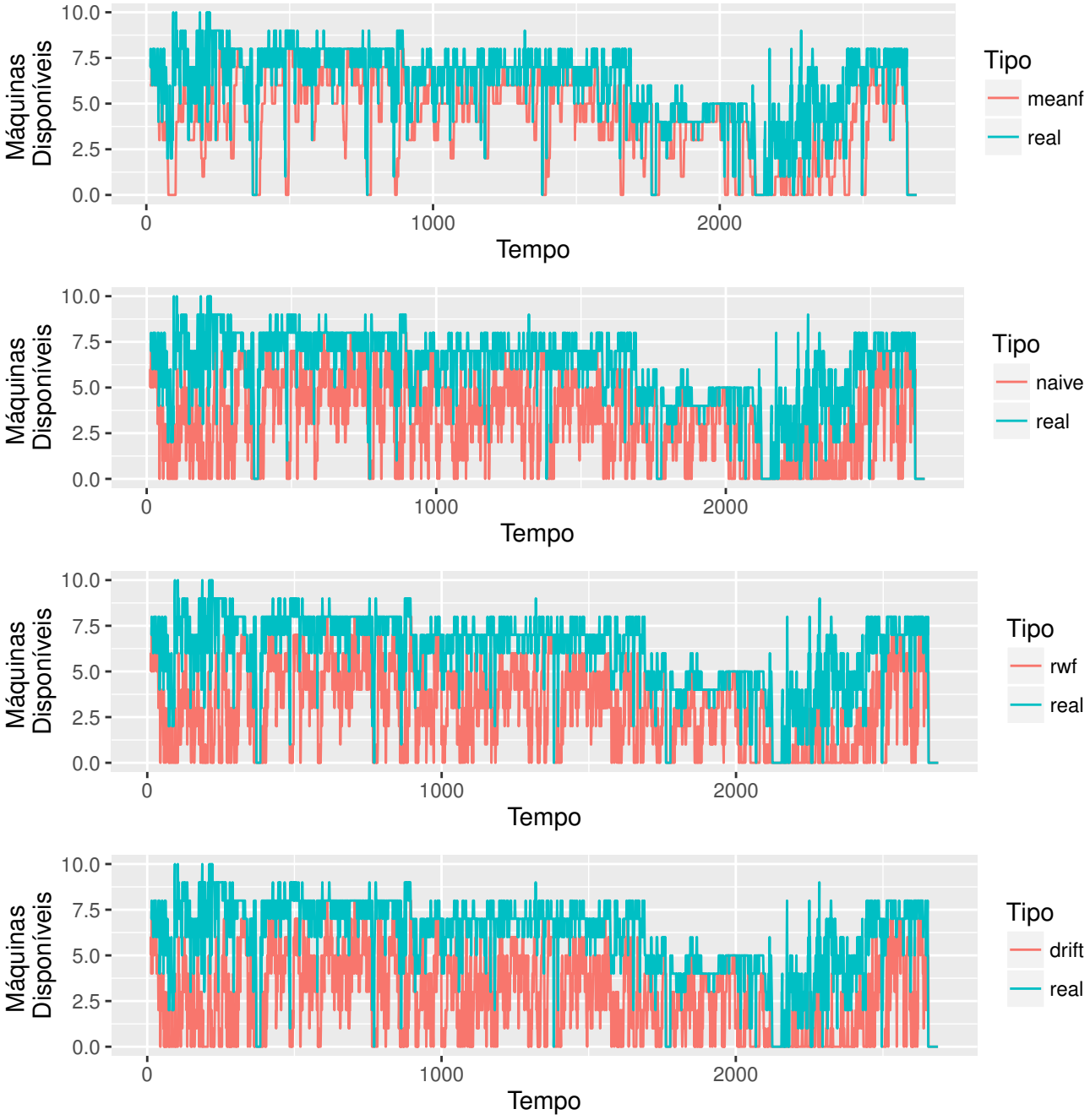

Figura B.6: Predição de recursos ociosos por preditor para máquina com ID 97959424 do *cluster* do Google com predição de 2 janela de 15 minutos no futuro, utilizando preditor pessimista.

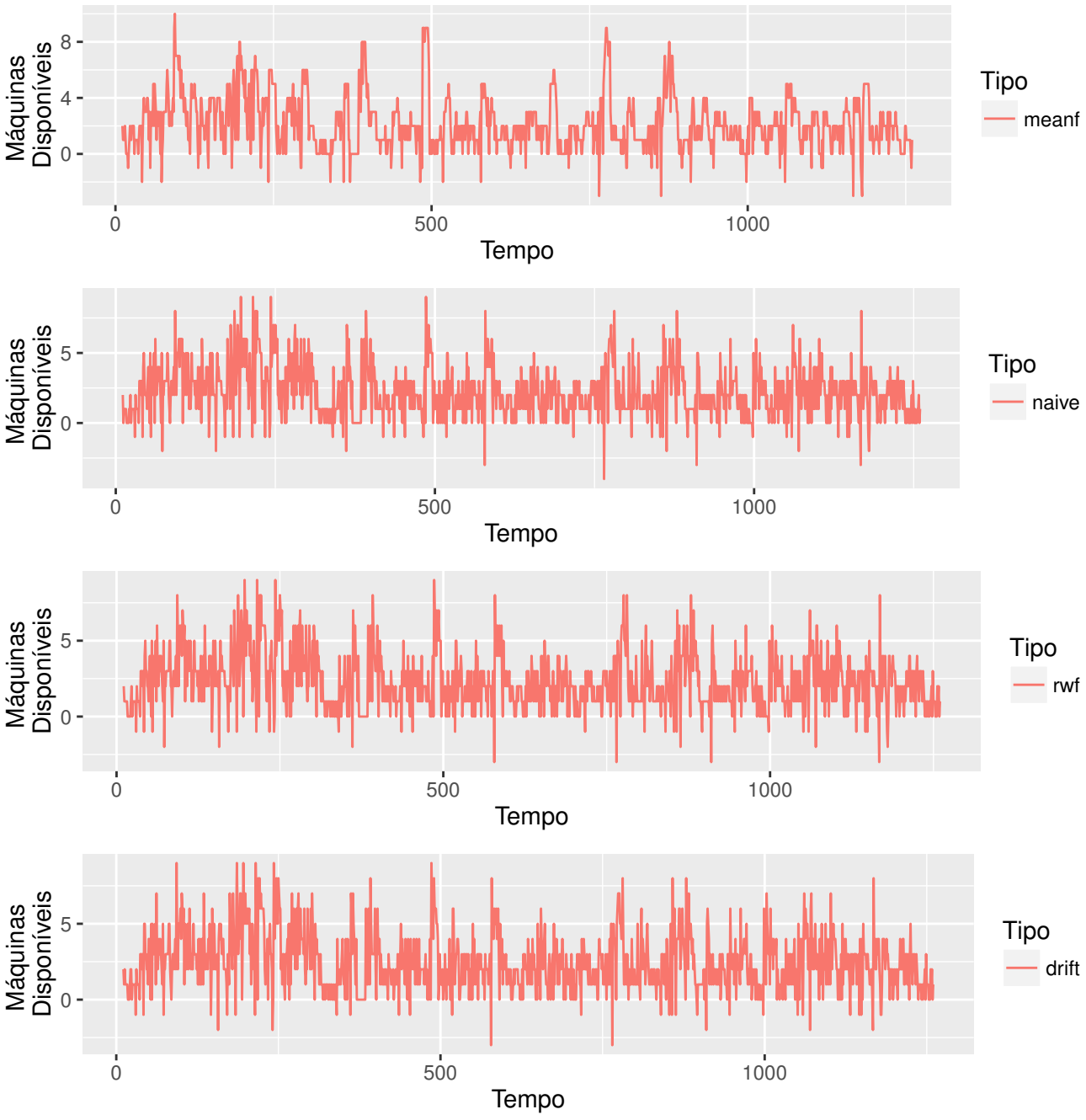

Figura B.7: Erros de predição por preditor para a máquina com ID 97959424 do *cluster* do Google com 1 janela de predição de 15 minutos no futuro, utilizando preditor pessimista.

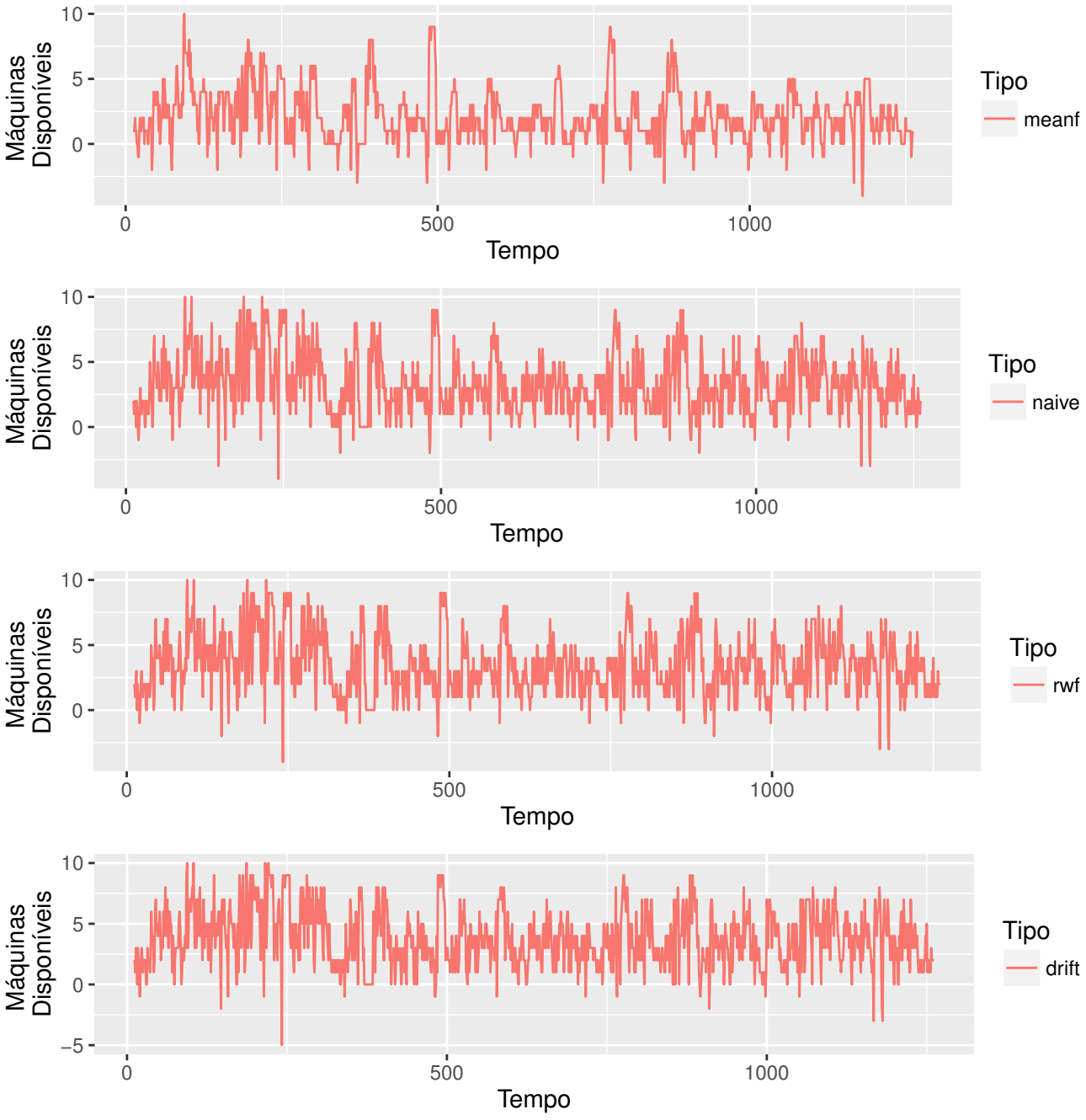

Figura B.8: Erros de predição por preditor para a máquina com ID 97959424 do *cluster* do Google com 2 janelas de predição de 15 minutos no futuro, utilizando preditor pessimista.

Apêndice C

Registros do Google

Seguem alguns registros de utilização do *cluster* do Google. Podemos ver aqui alguns registros de utilização de 1 dia e também da utilização agregada dos 29 dias registrados.

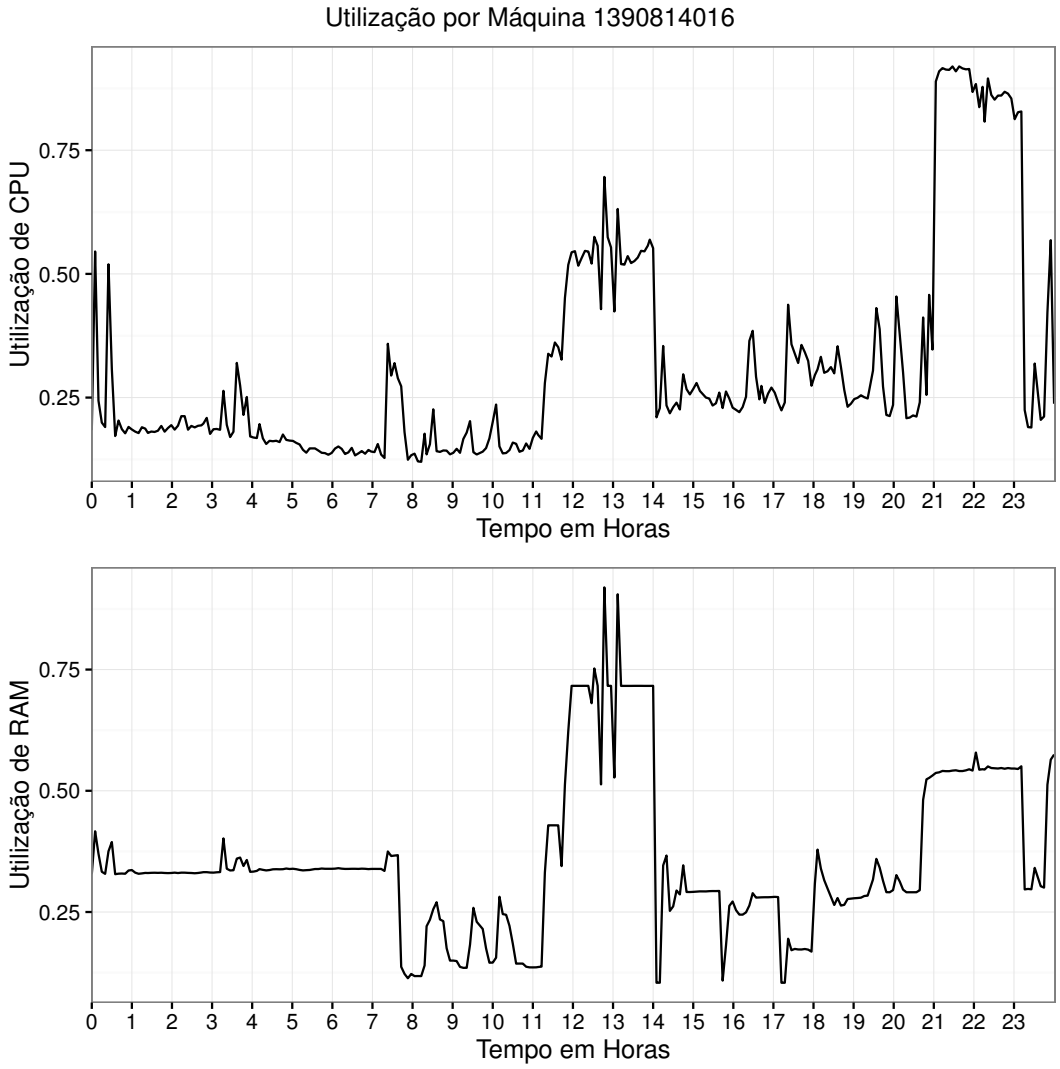

Figura C.1: Gráfico de utilização de CPU e memória RAM por hora, do primeiro dia do registro do Google para máquina com ID 1390814016.

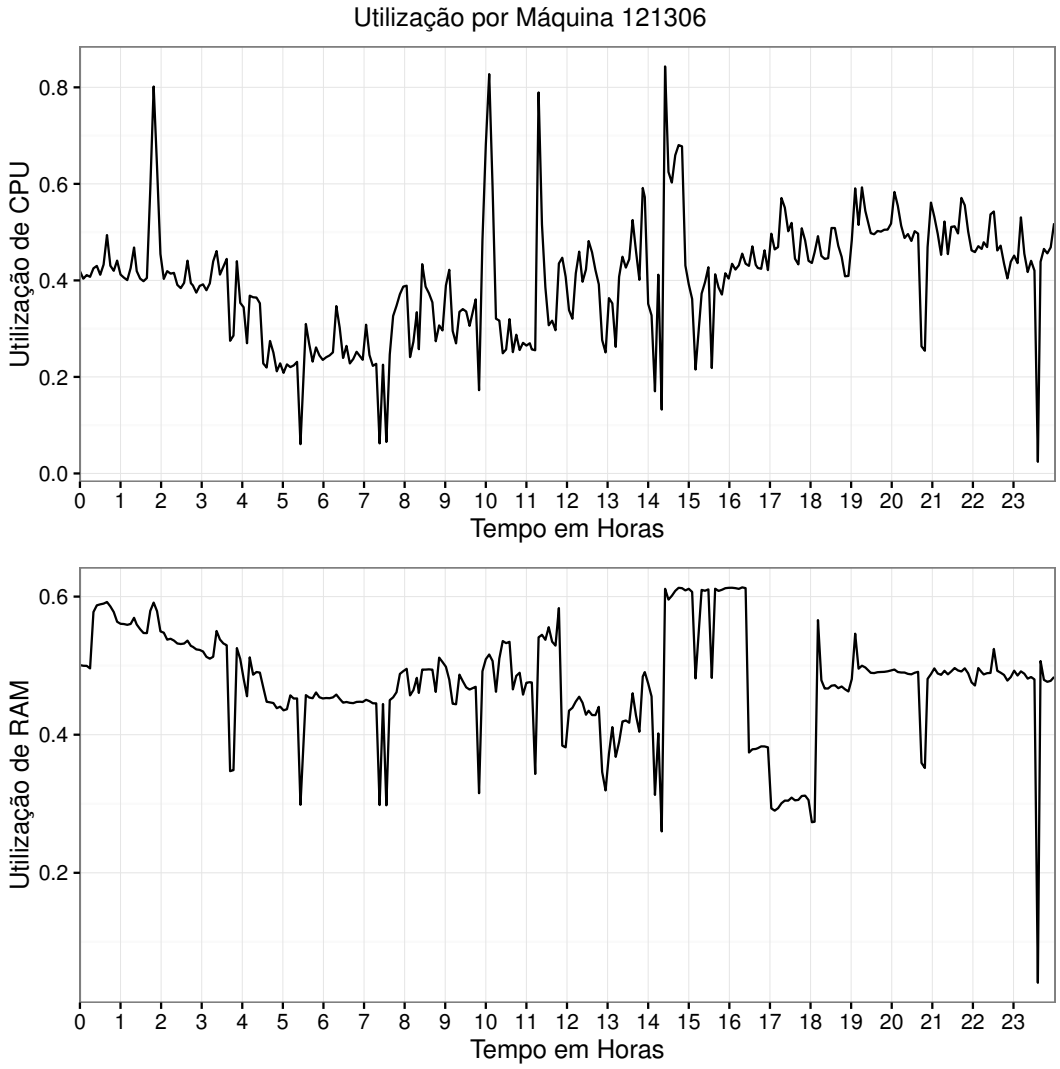

Figura C.2: Gráfico de utilização de CPU e memória RAM por hora, do primeiro dia do registro do Google para máquina com ID 121306.

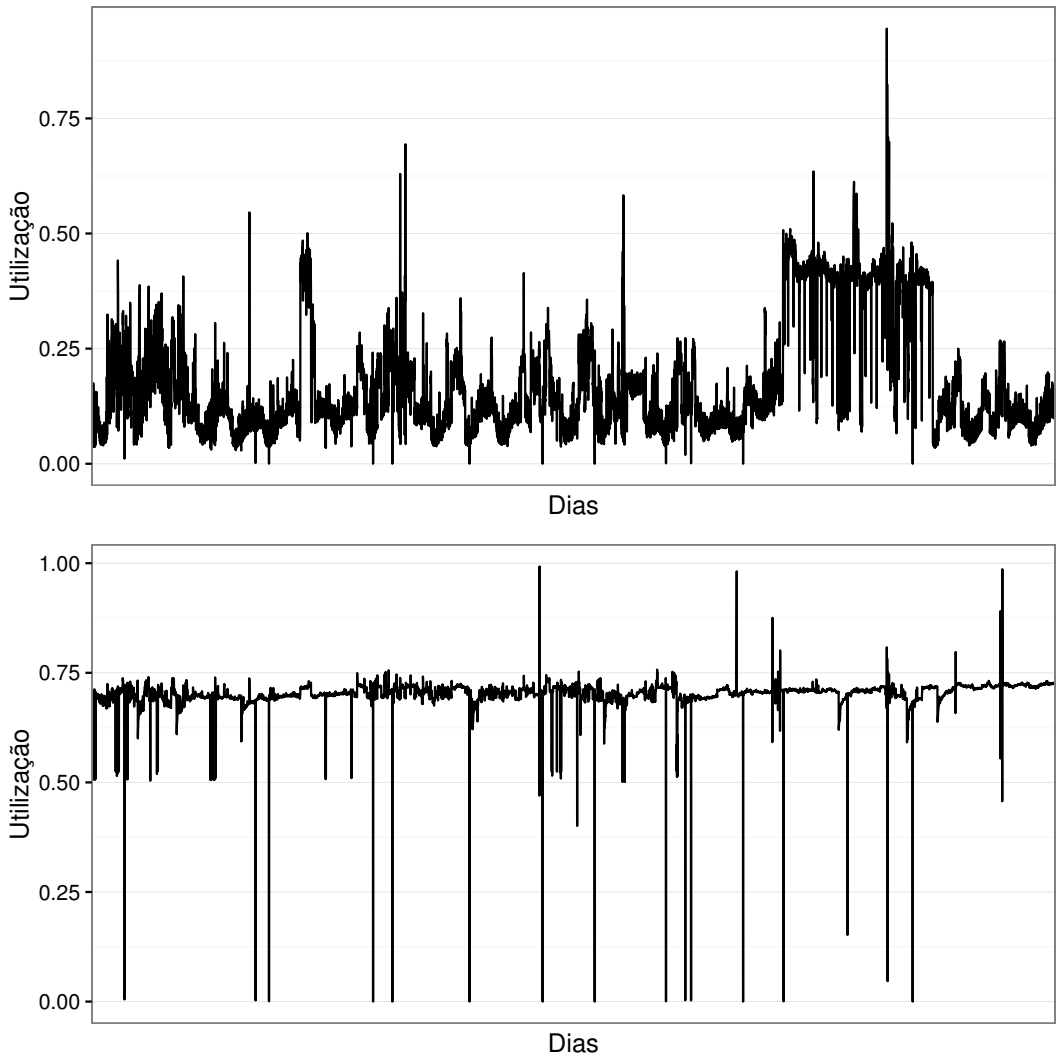

Figura C.3: Gráfico de utilização de CPU e memória RAM, dos 29 dias do registro do Google para máquina com ID 277433540.

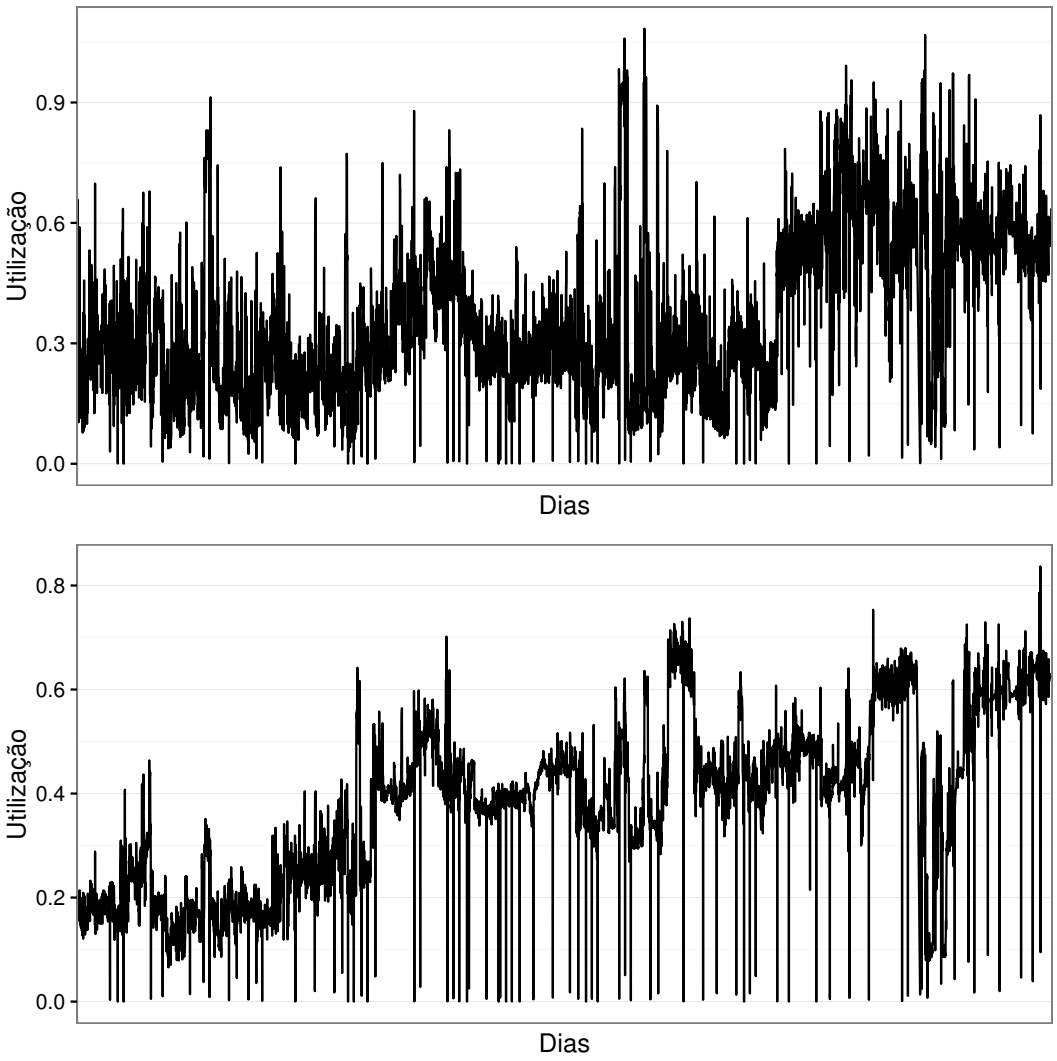

Figura C.4: Gráfico de utilização de CPU e memória RAM, dos 29 dias do registro do Google para máquina com ID 4302737021.