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

Otimizando a utilização de ambientes de nuvem *PaaS* usando uma abordagem preditiva

Ítalo Henrique Costa Truta

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)
Raquel Vigolvino Lopes (orientadora)

Campina Grande, Paraíba, Brasil

FICHA CATALOGRÁFICA ELABORADA PELA BIBLIOTECA CENTRAL DA UFCG

T874o Truta, Ítalo Henrique Costa.
Otimizando a utilização de ambientes de nuvem PaaS usando uma abordagem preditiva / Ítalo Henrique Costa Truta. – Campina Grande, 2016.
75 f. : il. color.

Dissertação (Mestrado em Ciência da Computação) – Universidade Federal de Campina Grande, Centro de Engenharia elétrica e Informática, 2016.
"Orientação: Prof. Dr. Andrey Elísio Monteiro Brito, Profa. Dra. Raquel Vigolvino Lopes".
Referências.

1. 1. Rede de Computadores. 2. Computação na Nuvem. 3. Processamento de Dados – Modelo PaaS. I. Brito, Andrey Elísio Monteiro. II. Lopes, Raquel Vigolvino. III. Universidade Federal de Campina Grande, Campina Grande (PB). IV. Título.

CDU 004.7(043)

DTIMIZANDO A UTILIZAÇÃO DE AMBIENTES DE NUVEM PaaS USANDO UMA ABORDAGEM PREDITIVA"

ITALO HENRIQUE COSTA TRUTA

DISSERTAÇÃO APROVADA EM 05/09/2016

ANDREY ELÍSIO MONTEIRO BRITO, Dr., UFCG
Orientador(a)

RAQUEL VIGOLVINO LOPES, D.Sc, UFCG
Orientador(a)

FRANCISCO VILAR BRASILEIRO, Ph.D, UFCG
Examinador(a)

KEIKO VERÔNICA ONO FONSECA, D.Sc., UTFPR
Examinador(a)

CAMPINA GRANDE - PB

Resumo

Seguindo a tendência de diversas áreas da computação, o processamento de *Big Data* também foi movido para a nuvem, devido à flexibilidade e facilidade que o modelo de computação na nuvem oferece, especialmente no modelo PaaS, onde os usuários podem executar suas aplicações *Big Data* de maneira mais fácil e barata. Centros computacionais de nuvem necessitam fazer um gerenciamento adequado de recursos para fazer melhor aproveitamento do aparato disponível, bem como minimizar custo operacional, já que grande parte dos recursos disponíveis passam a maior parte do tempo ociosos. Muito dessa ociosidade se deve ao sistema de cotas, que considera alocação estática de recursos, ao invés de observar se os recursos estão sendo, de fato, utilizados.

O trabalho apresentado nesta dissertação propõe uma solução para melhor gerenciamento de recursos em nuvens voltadas para processamento de dados com o modelo PaaS. Esse procedimento é feito por meio de abordagens preditivas tanto a nível operacional, estimando a carga dos servidores que compõem a nuvem em instantes futuros com base na utilização real no momento e em dados históricos, e a nível de aplicação, estimando a duração de tarefas de processamento em lote usando um agrupamento baseado em dados de aplicações previamente executadas. Com estes dados, a abordagem proposta é capaz de tomar decisões não-triviais na medida em que o usuário submete aplicações para execução, tais como acelerar a tarefa, caso haja recursos sobrando no momento, adiá-la, caso não haja recursos no momento mas tenha previsão de que haverá em um curto prazo, ou rejeitá-la, caso não haja recursos no momento da chegada, nem previsão de haver nas próximas janelas de execução.

Fazendo uso dessa abordagem, em comparação com o caso usual, controlado pelo sistema de cotas estáticas de recurso, obtivemos um acréscimo médio de mais de 10% dos recursos da nuvem, com um acréscimo no custo operacional de apenas 1%, considerando a não-proporcionalidade de energia, também avaliada nos nossos experimentos. Além disso, houve um aumento de 20% na taxa de requisições de tarefas processadas com sucesso, o que acarretou em um acréscimo no faturamento líquido entre 10 e 20%.

Abstract

Following the trend of many computing areas, Big Data processing has also been moved to the cloud, due to the flexibility and easiness the cloud model provides, specially in the PaaS model, where users can run their Big Data applications in an even easier and cheaper way. Cloud computing data centers need to have proper resource management in order to make best usage of available capabilities and minimize operational cost, as a large portion of the computational resources is idle most of the time. This idleness is mostly caused by the quota management system, which only considers static resource allocation, instead of looking whether the resources are indeed used.

In this work, we propose a solution to better manage the resources in PaaS cloud environments, focused on data processing. This management is made through predictive approaches, both in operational level, forecasting the workload of cloud servers in next time windows (based on the current utilization and historical data), and at application level, estimating the makespan of batch data processing applications with a clustering algorithm based on previously executed jobs characteristics. With those data, the proposed solution is able to take a set of non-trivial decisions, such as accelerating the job if more resources than requested are available, postponing the job when resources are only available in next time windows, or rejecting it, when there are not enough resources at the moment, neither in next windows.

With this approach, when compared to the usual case, regulated by static resource quotas, we obtained a 10% increase of average CPU and RAM utilization across the cloud, with an operation cost increase of only 1%, considering the non-proportionality of power consuming, also observed in our experiments. Besides that, the system also showed a 20% increase in the average successfully processed jobs, occasioning a profit increment between 10% and 20%.

Agradecimentos

Agradeço primeiramente a minha família, em especial aos meus pais, Clemilton e Livya, meu irmão Hiago, minha tia Tânia e a Gilmar, por todo o suporte que me deram durante essa jornada acadêmica, e à minha namorada Silmara por ter me aguentado também durante diversos momentos difíceis dessa jornada.

Toda minha gratidão também a Andrey, meu orientador durante toda essa jornada, além do importante papel da co-orientadora Raquel, especialmente no início. Obrigado pelos diversos ensinamentos, direcionamentos e repreendidas durante esse tempo, que serviram tanto para meu crescimento profissional quando pessoal.

Meu muito obrigado ao pessoal do Laboratório de Sistemas Distribuídos por formarem esse excelente ambiente de trabalho, que além de ter pessoas para ajudar imensamente na parte técnica, também proporcionam raros momentos de descontração. Agradeço também o pessoal da manutenção na pessoa de Guilherme, pelas incontáveis horas de pronto-socorro 24/7.

Finalmente, agradeço aos amigos, que mesmo não envolvidos cientificamente, contribuem para grandes momentos durante essa jornada.

Conteúdo

Lista de Símbolos

API - *Application Programming Interface*

AWS - *Amazon Web Services*

CLI - *Command Line Interface*

DPaaS - *Data Processing as a Service*

EC2 - *Elastic Cloud Computing*

EDP - *Elastic Data Processing*

EMR - *Elastic MapReduce*

HPE - *Hewlett-Packard Enterprise*

IaaS - *Infrastructure as a Service*

KNN - *K-Nearest Neighboors*

kWh - *Kilo Watt-Hora*

PaaS - *Platform as a Service*

SaaS - *Software as a Service*

SLA - *Service Level Agreement*

SLO - *Service Level Objective*

VCPU - *Virtual CPU*

VM - *Virtual Machine*

Lista de Figuras

Lista de Tabelas

Capítulo 1

Introdução

Nesse capítulo, introduzimos o trabalho, dando uma contextualização do problema, bem como sumarizando as contribuições aqui planejadas.

1.1 Contextualização

1.1.1 Computação na nuvem

A computação na nuvem popularizou-se nos últimos anos por ser uma alternativa mais eficiente, fácil e barata para obtenção de recursos computacionais sob demanda. Ambientes de computação na nuvem, ou apenas nuvens, são classificados geralmente em três tipos: (i) nuvens públicas, disponibilizadas para o público geral, onde o usuário paga de acordo com a utilização, (ii) nuvens privadas, que em geral, atendem internamente as demandas de determinadas organizações, tais como empresas e universidades, e por último, (iii) nuvens híbridas, que têm propósito similar ao da nuvem privada, mas angariam recursos de nuvens públicas. Cada um desses ambientes pode prover diferentes tipos de serviços, que se diferenciam de acordo com o nível de responsabilidades entre provedor e usuário. Em um nível mais baixo de responsabilidades, há o modelo de infraestrutura como um serviço (IaaS, do inglês *Infrastructure as a Service*), onde o usuário requisita recursos de processamento, rede e armazenamento na forma de máquinas virtuais (VM, do inglês *Virtual Machine*), e o provedor de nuvem é responsável por administrar a infraestrutura dessas máquinas, cuidando de virtualização, rede e armazenamento. Do outro extremo, em um nível mais alto de responsa-

bilidades do provedor, está o modelo de *software* como serviço (SaaS, do inglês *Software as a Service*), onde o cliente é o usuário final de um sistema completo, sendo responsabilidade do provedor de nuvem manipular desde a infraestrutura de *hardware* até os requisitos de aplicação e dados. Diversas aplicações *web* usam esse modelo, tais como clientes de *e-mail* e sistemas do tipo CRM, que gerenciam contato com o cliente.

Em uma posição intermediária a estes dois tipos, há o modelo de plataforma como serviço (PaaS, do inglês *Platform as a Service*), onde o provedor de nuvem é responsável pela plataforma na qual o usuário executa suas aplicações e manipula seus próprios dados. O usuário pode requisitar diversos tipos de plataformas pré-configuradas, como um ambiente para desenvolvimento pronto para linguagem Python, um servidor *web*, ou ferramentas de processamento de dados.

1.1.2 Processamento de dados na nuvem

Com a massificação da internet, o número de dispositivos geradores de dados e o crescente número de usuários, a quantidade de dados disponíveis cresce em um ritmo cada vez mais acelerado. Há uma infinidade de aplicações que geram altos volumes de dados, tais como redes sociais, *sites* de *e-commerce* e portais de notícias. Esses dados, porém, encontram-se originalmente em uma forma não estruturada, dificultando seu processamento. Devido ao grande volume que esses dados podem alcançar, chegando a atingir petabytes de informação, é inviável processá-los de forma tradicional. Dentro deste contexto, surgiu *Big Data*, um conjunto de tecnologias voltadas para o processamento de grandes massas de dados, de forma mais eficiente, facilitando a obtenção de informações não triviais a partir destas bases. Dentro de *Big Data*, há várias ferramentas para diferentes tipos de processamentos de grandes massas de dados, como por exemplo, uma categoria que engloba ferramentas que executam o processamento dos dados em tempo real, onde as informações são processadas de forma imediata, à medida que chegam, enquanto uma outra grande categoria é a de processamento em lote (do inglês *batch processing*), na qual grandes massas de dados são acumuladas para serem processadas em determinado momento.

Uma das ferramentas mais consolidadas no mercado para processamento de *Big Data* é o Apache Hadoop [6]. O Hadoop é um arcabouço de código aberto que permite, fazendo uso do paradigma MapReduce [29], executar aplicações em grandes volumes de dados. No

MapReduce, as aplicações podem ser executadas de maneira distribuída em um *cluster* de máquinas, de modo a escalar até centenas ou milhares de máquinas. Neste paradigma, o processamento se divide em duas funções principais: o *map*, que gera um conjunto intermediário de dados que serão consumidos de forma agrupada pela função *reduce*, que, por sua vez, faz o agregamento destes dados e produz o resultado final.

O Hadoop, por meio das funções *map* e *reduce* trata uma tarefa como um conjunto de subtarefas, dividindo grandes conjuntos de dados em várias partições, tal que cada unidade de processamento lida com uma partição do conjunto, em paralelo com as demais unidades. A partir desse modelo, Hadoop implementa replicação e tolerância a falhas a nível de aplicação, removendo do usuário essa responsabilidade. Ademais, atividades como distribuição das tarefas, gerência dos nós e tolerância a falhas são nativamente oferecidas pelo Hadoop e funcionam de uma forma automatizada, cabendo ao usuário apenas escrever as aplicações MapReduce. No projeto Hadoop principal, mantido pela *Apache Foundation*, os principais componentes são:

- HDFS (do inglês *Hadoop Distributed File System*): sistema de arquivos distribuído que provê alta vazão e disponibilidade para aplicações de processamento de dados;
- Hadoop YARN: módulo responsável pelo gerenciamento de recursos e tarefas no *cluster*;
- Hadoop MapReduce: implementação do paradigma MapReduce, módulo responsável pelo processamento dos dados.

Seguindo a tendência de diversas aplicações, o processamento de dados também foi movido para a nuvem, beneficiando-se da facilidade e flexibilidade de requisitar recursos que o modelo de nuvem oferece. No modelo tradicional de criar um *cluster* de máquinas para execução de tarefas Hadoop, o usuário precisa alocar a quantidade de máquinas físicas desejada, configurar armazenamento e rede entre elas para, então, instalar cada componente do Hadoop em todas estas máquinas e executar o processamento. Em computação na nuvem, especialmente no modelo de PaaS, esse processo é muito simplificado. Há serviços de PaaS voltados especificamente para processamento de dados, tais como o Amazon Elastic MapReduce [2] (EMR), no qual o usuário especifica o quanto de processamento deseja, *e.g.*, número de VMs, número de VCPUs (do inglês *Virtual CPUs*) e quantidade de RAM,

enquanto o provedor é responsável pela configuração de forma automatizada da plataforma para processamento de ferramentas como o Hadoop, para então ceder os recursos para o usuário. Nesse exemplo, tendo o ambiente configurado, cabe ao usuário apenas submeter seus dados e aplicações Hadoop. O processamento Hadoop na nuvem se beneficia também da elasticidade que a nuvem oferece, uma vez que em soluções como o Amazon EMR, é possível redimensionar o *cluster* Hadoop de forma simples e automatizada, enquanto no modelo tradicional, seria necessário reexecutar manualmente todas as etapas de configuração para cada cada unidade de processamento que venha a ser adicionada.

Esse tipo de plataforma é empregado tanto por usuários sob demanda que criam um *cluster* para executar uma única tarefa, quanto por sistemas automatizados, a exemplo dos que seguem a arquitetura *lambda* [11; 45], que pode ser resumida em um sistema que responde requisições em tempo real de forma mais simples, por exemplo, com base em um modelo pré-computado, e que periodicamente executa uma tarefa *batch* para refinamento dos modelos que serão usados no processamento das requisições em tempo real. Um sistema com essa arquitetura se beneficia da facilidade introduzida pelo modelo de PaaS, uma vez que seria muito custoso manter um *cluster* de máquinas físicas exclusivamente para processamentos pontuais. Em ambos os casos, com PaaS, os recursos (VMs) são normalmente voláteis, podendo ser apagados e recriados com poucos comandos, em questão de minutos.

1.1.3 OpenStack e Sahara

Com o crescimento da demanda por computação na nuvem, diversas tecnologias surgiram com operadores visando a fazer melhor uso de seus recursos, e construir seus próprios ambientes de nuvem. Nesse cenário, o OpenStack [5], um sistema operacional de código aberto para gerenciamento de nuvens públicas ou privadas, se destacou. Entre suas principais vantagens, o OpenStack conta com uma comunidade grande e ativa, além de uma arquitetura modular, baseada em um conjunto pequeno de serviços fundamentais, mas com uma grande quantidade de outros serviços opcionais que podem ser adicionados conforme a necessidade da nuvem.

Os principais componentes são os responsáveis por prover computação, identidade, rede e imagens de sistemas operacionais, chamados, respectivamente de Nova, Keystone, Neutron e Glance. Além disso, há outros serviços que podem ser considerados principais, a

exemplo do Horizon, que provê interface gráfica ao OpenStack, sendo ponto de interação da maior parte dos usuários finais, Cinder, o componente de armazenamento em blocos, e o Swift, para armazenamento de objetos. Além de expostos na interface gráfica do Horizon, os componentes OpenStack também disponibilizam acesso às suas funcionalidades por meio de APIs REST, bibliotecas Python e clientes de linha de comando.

Dentre os componentes não-essenciais, o OpenStack possui um componente voltado para processamento de dados, seguindo o modelo de PaaS, chamado de Sahara. Com propósito similar ao Amazon EMR, o Sahara facilita a criação de *clusters* configurados para diversos arcabouços de processamento, tais como Spark [7], Storm [3] e Hadoop. Além de possibilitar criar *clusters*, o Sahara possui uma interface para submissão e monitoramento de tarefas nestes arcabouços. A arquitetura em alto nível do Sahara é ilustrada na Figura 1.1.

Quando a API REST do Sahara recebe uma requisição para iniciar o processo de criação de um *cluster* de VMs para processamento de dados, o mecanismo de provisionamento do Sahara delega para o Nova, através de sua API REST, a criação de cada uma destas VMs. Antes de criar as VMs, o Nova também é responsável pelo gerenciamento de cotas de seus recursos para determinados usuários ou projetos. Atualmente, o OpenStack realiza o gerenciamento de cotas dividido de acordo com as responsabilidades de cada serviço. A título de exemplo, o Glance, componente do OpenStack que gerencia imagens de sistemas operacionais, controla a cota de imagens que podem ser criadas por usuários, enquanto o Neutron, serviço de gerenciamento de redes, coordena a criação de seus respectivos recursos, como redes e endereços IP públicos. O Nova, por sua vez, é responsável por cotas de chaves de acesso, grupos de segurança e recursos individuais de VMs, que envolvem número total de CPUs virtuais (VCPUs), instâncias e total de RAM a ser usado. Esse sistema de cotas funciona independente do *hardware*, atuando como uma camada a nível de aplicação para controlar alocação de recursos de forma estática. Um caso comum de compartilhamento de cota acontece quando o administrador da nuvem cria um projeto, adiciona usuários a este projeto e define cotas de recursos, tais como número de instâncias, VCPUs, RAM, volumes e imagens. Dessa maneira, no momento que um usuário membro desse projeto requisita a criação de um *cluster* de 5 VMs, totalizando 5 VCPUs e 10 GB de RAM, se um dos três atributos não for satisfeito, ou seja, não haver cota livre suficiente no projeto, o Nova rejeitará a criação deste *cluster*. Caso haja cota, o Nova cria as máquinas virtuais e esta cota é

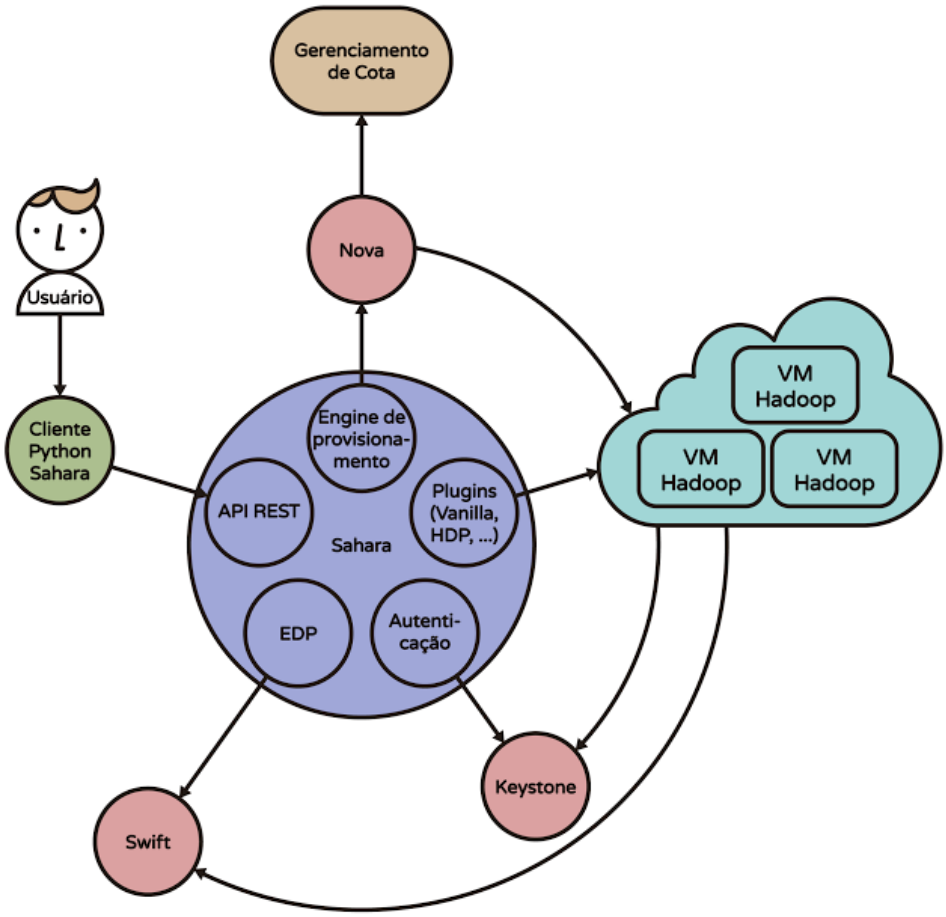


Figura 1.1: Arquitetura do Sahara, o componente de processamento de dados do OpenStack

descontada do total livre, ficando alocada para aqueles recursos.

Uma vez criadas as VMs, o próximo passo do Sahara é configurar este *cluster*, que se dá através de *plugins* de arcabouços. Atualmente, o Sahara conta com *plugins* para criação de plataformas de Apache Hadoop, Storm, Spark, além do Hortonworks Hadoop, uma versão modificada do Apache Hadoop. Cada um desses *plugins* contém o passo-a-passo necessário para, a partir de VMs recém-criadas, criar um ambiente funcional para cada um desses arcabouços, inicializando cada um de seus componentes. Finalmente, com o *cluster* configurado, entra em ação o componente EDP (do inglês, *Elastic Data Processing*), que armazena informações de aplicações e seus dados a serem executadas, para executar em um *cluster* apropriado.

O Sahara foi utilizado nesta pesquisa para criação de ambientes de processamento de tarefas Hadoop, bem como servirá de base para implementação da solução proposta.

1.2 Problema investigado

Com a crescente demanda por computação na nuvem, cresce também a necessidade de manter centros computacionais cada vez maiores, capazes de satisfazer essa demanda. Entretanto, a utilização desse grande volume de recursos é subótima. Estudos mostram que grandes *data centers* tem utilização média entre 30 e 50% [40; 21], considerando métricas como uso total de CPU e RAM. Com isso, há uma grande vertente de estudos que buscam aumentar esta utilização média, fazendo uso de técnicas como consolidação de máquinas virtuais, que tem objetivo de minimizar o número de máquinas físicas em funcionamento [30; 22; 36], e técnicas de melhor escalonamento de recursos, a nível operacional, como controle dinâmico de frequência e voltagem (DVFS, do inglês *Dynamic voltage and frequency scaling*) [39; 25], e a nível de aplicação, como consolidação de tarefas [35]. Estes estudos têm muitas interseções com a área de computação verde (do inglês, *green computing*), que tem como maior objetivo reduzir a emissão de gás carbônico na atmosfera, buscando um menor consumo de energia em grandes centros computacionais [25; 48; 43; 38]. Além do fator ambiental, essa redução é importante também do ponto de vista econômico, uma vez que o consumo de energia é responsável por, em média, 50% [33] dos custos operacionais totais de um *data center*, custos esses que podem chegar a cifras

milionárias no caso de grandes *data centers*.

Um fator forte considerado na maioria destes estudos é a não-proporcionalidade do consumo de energia em relação ao uso de recurso. O consumo de energia oriundo de um servidor cresce sub-linearmente à medida que a utilização de seus recursos, como CPU e RAM cresce. Um exemplo deste cenário é que um servidor ligado, porém ocioso, tem um consumo equivalente a mais de metade da energia do mesmo estando sobrecarregado, com CPU e RAM sendo utilizadas próximas de sua totalidade. Por conseguinte, foi observado que servidores possuem uma zona ótima para execução, na qual conseguem obter o maior índice de eficiência energética, ou seja, executam mais tarefas consumindo proporcionalmente menos energia.

Há, ainda, uma série de estudos com objetivo de otimizar aspectos da nuvem analisando seu comportamento histórico, observando características que demonstrem padrões de utilização. A título de exemplo, nuvens voltadas para infraestrutura de desenvolvimento e suítes de teste de integração contínua em geral têm normalmente uma maior utilização durante o horário comercial, já que a maior parte de seus usuários está ativa, desenvolvendo e submetendo mudanças, passando o período da madrugada em estado ocioso. Uma vez conhecido o padrão desta nuvem, há uma série de ações que o administrador pode tomar para fazer melhor uso dos recursos, tais como: (i) Desligar ou manter em estado de baixo consumo de energia as máquinas mais ociosas; (ii) Programar execuções em lote de clientes com arquitetura *lambda*; (iii) Executar tarefas menos prioritárias mas que demandam mais processamento, e; (iv) Realizar rotinas de auditoria, manutenção e outras análises. Para tal, são aplicadas diversas técnicas que buscam prever o quão ocupada a nuvem estará em um momento futuro, a partir de dados históricos, como regressão, agrupamento, aprendizado de máquina, entre outras. Estas abordagens servem tanto para obter uma janela de previsão a curto prazo, como para obter padrões de sazonalidade na carga da nuvem a médio prazo. Ambas informações podem ser úteis na tomada de decisão do administrador, dependendo de qual ação ele pretende executar.

1.3 Contribuições e relevância

Este trabalho tem como objetivo abordar o problema do desperdício de recursos em *data centers*, investigando a diminuição da quantidade de recursos disponíveis, porém ociosos, atingindo um cenário mais eficiente de utilização de recursos da nuvem. Para tanto, é proposto um sistema aplicado em nuvens do tipo PaaS voltadas para processamento de dados. Este sistema faz uso de abordagens preditivas, tanto a nível de aplicação, prevendo a duração de tarefas a serem executadas de acordo com as respectivas características, quanto a nível operacional, com um preditor de carga da nuvem a curto prazo, que considera uma janela histórica. Combinadas essas técnicas, o arcabouço analisa tarefas submetidas para a nuvem, examina o estado atual e em curto prazo da nuvem para, então, fazer o escalonamento desta tarefa. O escalonador, então, pode tomar decisões não-triviais, como: (i) Adiar a tarefa, caso a mesma chegue em um momento que a nuvem esteja carregada, mas haja recursos disponíveis em curto prazo; (ii) Antecipar a tarefa, caso haja recursos disponíveis no momento e predição de que continue havendo; (iii) Rejeitar a tarefa, caso não haja recursos disponíveis no presente momento e na próxima janela de predição.

Entre os benefícios oriundos desse conjunto de decisões tomadas pelo escalonador, podemos listar: (i) Melhora da utilização média dos recursos da nuvem; (ii) Diminuição do número de tarefas rejeitadas; (iii) Melhora na eficiência energética da nuvem, fazendo-a operar em um nível mais eficiente; (iv) Diminuição dos custos de provisionamento do usuário, identificando cenários que aumentar o número de recursos pode não ser benéfico. Esses objetivos foram atingidos com pouca ou nenhuma degradação de performance em alguns cenários.

1.4 Metodologia

A partir de levantamento bibliográfico do estado da arte, que serviu de base para investigações sobre otimização de ambientes de processamento de dados, foi produzida uma versão do arcabouço aqui proposto unindo ideias de trabalhos que consideram a previsão de carga de nuvem e outros trabalhos baseados em predição de tarefas de processamento de dados. Em seguida, o arcabouço foi implementado para validação, fazendo adição de um

subcomponente ao fluxo do OpenStack Sahara, que será detalhado nos próximos capítulos. A implementação foi feita usando a linguagem Python [18], para comunicação com o OpenStack e fluxos gerais da aplicação, além da linguagem R [15], para os procedimentos estatísticos, que envolvem as predições de carga e das durações de tarefas.

Após a implementação, o sistema foi instanciado e configurado em uma nuvem experimental, utilizando configurações comuns em soluções existentes no mercado e na literatura. Foi realizada também injeção de carga no ambiente utilizando registros públicos de *clusters* voltados para processamento de dados do Google, a fim de criar um ambiente com carga real. Nesta nuvem, foram executadas diferentes tarefas de processamento com variadas configurações, para comparar o arcabouço proposto com um cenário simplista de execução de tarefas MapReduce controlado pelo sistema padrão de cotas. Essa comparação foi feita coletando, durante a execução de cada um dos cenários, métricas como: (i) Utilização de CPU e memória dos servidores que compõem a nuvem; (ii) Percentual de tarefas aceitas para processamento; (iii) Tempo de execução de tarefas; e (iv) Consumo de energia da nuvem.

1.5 Organização do documento

Este documento é organizado da seguinte forma:

- Capítulo 2 mostra os trabalhos relacionados na literatura, e como outros autores resolveram problemas similares ao nosso;
- No Capítulo 3 é detalhada a abordagem aqui proposta, onde mostramos em detalhes cada um dos preditores usados e o algoritmo principal da solução;
- Já no Capítulo 4, detalhamos os experimentos executados para montar um ambiente para a instanciação da solução, bem como os resultados obtidos nesta pesquisa;
- Finalmente, no Capitulo 5, concluímos o trabalho fornecendo um sumário dos resultados, e listamos as limitações e trabalhos futuros.

Capítulo 2

Trabalhos relacionados e estado da arte

Nesse capítulo, listamos os trabalhos relacionados, primeiro detalhando os que tratam de gerenciamento de recursos, e em seguida, trabalhos que fazem uso de técnicas de predição de recursos em ambientes de nuvem. Em seguida, é dado um breve sumário de como este trabalho se diferencia dos demais.

2.1 Gerenciamento de recursos

Há várias técnicas utilizadas para melhorar a utilização de ambientes de computação na nuvem. Uma abordagem bastante comum para isso é o uso de instâncias oportunistas, que são instâncias criadas em recursos alocados, porém não utilizados, e que possuem menor prioridade em relação às demais, podendo ser removidas a qualquer momento, caso os recursos utilizados por elas venham a ser requisitados pelas instâncias de alta prioridade, devido a alterações na utilização dos servidores. Um exemplo desse tipo de instância no mercado é a Instância Spot [1] do Amazon EC2, que tem preços mais baixos do que as instâncias tradicionais do EC2, porém, tem uma qualidade de serviço reduzida, já que podem ser perdidas em determinado momento da execução. O preço desse tipo de instância é calculado dinamicamente ao longo do tempo, de acordo com a quantidade de recursos ociosos, e o usuário faz uma proposta de que preço em relação ao das VMs tradicionais está disposto a pagar para mantê-las. Uma vez que esse preço excede a proposta do usuário, as instâncias são removidas. Chohan [26] *et al.* executoram experimentos para investigar como o uso de instâncias oportunistas poderia melhorar a performance de tarefas Hadoop. Eles mostr-

ram que é possível ter um acréscimo de desempenho quando não acontecem falhas, ou seja, quando a instância não é perdida durante a execução. Contudo, a ocorrência de uma falha torna o processamento muito mais lento e, consequentemente, mais caro. Chohan propôs uma função de custo a fim de otimizar a proposta do usuário garantindo uma redução do custo com ganho de desempenho em processamento de tarefas.

No trabalho de Voorsluys [46] e Buyya, os autores propõem uma estratégia de provisionamento de instâncias tipo Spot para tarefas MapReduce que inclui técnicas de tolerância a falhas, como migração e duplicação de tarefas com objetivo de manter as instâncias disponíveis até o fim do processamento. Com o uso dessa técnica, os autores mostraram que é possível ter eficácia no processamento de dados com instâncias Spot, respeitando regras de qualidade de serviço e reduzindo custo apesar da eminência de perda de VMs. Já Carvalho *et al.* [24] introduzem uma nova classe de instâncias chamada de *Economy*, que são mais baratas que instâncias sob-demanda, porém um pouco mais caras que as instâncias oportunistas. Assim como as instâncias Spot, as Economy também são criadas utilizando recursos ociosos e, fazem uso de preditores de carga para a nuvem. Eles definem um tipo de instância oportunista com um SLO (do inglês *Service Level Objective*, acordos de nível de serviço, definidos em contrato) de disponibilidade de acordo com a carga estimada para as próximas janelas, que consideram semanas de dados. Os autores concluíram que esse tipo de instância pode ser benéfico tanto para o usuário, que tem uma oportunidade intermediária de acelerar seu processamento a melhor custo com mais garantia de serviço, como para o provedor de nuvem, onde mostram um cenário que há um acréscimo de lucro de cerca de 60%.

Outra técnica muito utilizada é o *overcommit* ou *overbooking*, que considera que os recursos estão ociosos na maior parte do tempo e, por isso, aloca mais recursos do que há de fato disponíveis. A título de exemplo, o OpenStack usa por padrão um *overcommit* de 150% para memória RAM e 1600% para CPUs [13]. Entre os estudos dessa categoria, destacamos o de Ghosh *et al.* [31], que faz um estudo analítico dos riscos associados ao *overcommit*, e como esse uso pode ser benéfico, desde que usado de maneira cuidadosa, fazendo uso de monitores, preditores e outros tipos de análise estatística. Ainda nessa área, Tomás *et al.* mostram que um bom uso de *overcommit* pode melhorar a utilização dos recursos da nuvem. Com base em algumas características observadas, como os picos e variações de carga, eles propõem algoritmos de escalonamento, usando técnicas de modelagem, monito-

ramento e predição para aumentar o uso dos recursos, sem ultrapassar, de fato, um limiar que consideram não ser recomendável de ser mantido. Como resultado, há um grande aumento na utilização de recursos, observada em uma avaliação de desempenho feita com múltiplas cargas de trabalho.

Há ainda uma área de estudos, chamada de *Green Computing*, que tem objetivo de otimizar a utilização de recursos visando a alcançar uma maior eficiência energética, seja ela em termos financeiros ou reduzindo a emissão de gás carbônico na atmosfera. Um destes trabalhos é o de Farahnakian *et al.,* [30], criadores de uma técnica de escalonamento de VMs chamada de *Ant Colony System*, que monitora os recursos por meio de agentes locais e globais, buscando aumentar a utilização dos recursos por meio de consolidação de máquinas virtuais, ou seja, movendo VMs de servidores ociosos e desligando-os a fim de se evitar servidores ligados, mas com utilização muito baixa. Essa moção de VMs entre servidores é feita via *live migration*, técnica na qual a máquina virtual pode ser movida sem precisar ser desligada. O uso do algoritmo *Ant Colony System* proposto por Farahnakian consegue diminuir o número de máquinas em baixa utilização, colocando-as em estado de baixo consumo de energia, e diminuindo o consumo geral da nuvem, sem degradar atributos de qualidade de serviço como desempenho e disponibilidade. Outro exemplo de otimização de recursos com foco em eficiência energética é o trabalho de Hsu [35] *et al.*, que usa consolidação de tarefas para encontrar um balanço entre utilização e consumo de energia. Isso é feito estabelecendo um limiar de uso de CPU e criando *clusters* virtuais, que representam diversos tipos de aplicação, utilizando um modelo robusto para estimar o custo de energia para cada aplicação. A solução proposta por Hsu *et al.* apresentou uma redução de 17% no consumo de energia em relação a *benchmarks* externos, sem perder qualidade de serviço.

2.2 Predição em nuvens computacionais

Em um nicho mais específico de escalonamento e otimização por meio de predições, há o PRESS (do inglês *PRedictive Elastic reSource Scaling*) [49]), onde Gong e demais conseguem, com um método não-intrusivo, prever de forma *online* demandas de recursos a longo e curto prazo, usando transformadas de Fourier e cadeias de Markov, respectivamente, obtendo erro de subestimação próximo de zero, e conseguindo reduzir o desperdício de recursos

e violações de SLO. Ainda nessa linha de pesquisa, Roy *et al.* [41] executam *autoscaling* a partir de modelos preditivos de carga da nuvem com técnicas de autorregressão e média móvel. Mais uma vez, os autores conseguem prever a utilização da nuvem com boa acurácia e otimizar sua utilização sem violar termos de QoS.

Já entre os estudos na literatura relacionados à predição do tempo de tarefas Hadoop, há o de Kavulya *et al.* [28], que analisa *logs* públicos de tarefas MapReduce em um *cluster* de produção do *Yahoo!* para entender características de cargas de trabalho e descobrir que pontos mais impactam o desempenho dessas tarefas. Nesse estudo, os autores identificaram a necessidade de um melhor escalonamento de tarefas para um melhor uso dos recursos e para impedir grandes tarefas de tomarem uma grande parte do *cluster*. Conseguem, assim, estimar o tempo de execução das tarefas com erro de 26%, utilizando um método de regressão linear. Ganapathi e Patterson [19] também modelaram a carga da nuvem visando a melhorar decisões de *design* e escalonamento. Eles desenvolveram um arcabouço estatístico baseado em uma técnica de análise de correlação, chamada KCAA (do inglês *Kernel Canonical Correlation Analysis* [20]), obtendo acurácia próxima de 80% na predição de duração de tarefas Hadoop fazendo correlação de características da pré-execução e métricas de desempenho da pós-execução. No trabalho de Tian e Chen [44], os autores estudam em mais detalhes o processamento MapReduce e montam uma função de predição baseada em regressão linear, que considera parâmetros como tamanho da entrada e recursos disponíveis, buscando minimizar o custo de execução de cada tarefa e melhorar a alocação de recursos. Finalmente, Carrera e Geyer [23] propõem uma forma de prever o tempo de duração de tarefas MapReduce na nuvem a partir de modelagem. Nosso trabalho se difere destes por fazermos uma abordagem baseada na semelhança entre tarefas a serem executadas e tarefas passadas, o que é adequado para um provedor público ou privado, baseado, por exemplo, na arquitetura *lambda*, que com frequência processa tarefas dos mesmos usuários repetidamente e que têm um histórico de informações sobre tarefas executadas, sendo a maior alteração entre elas o tamanho da entrada.

2.3 Considerações finais

Neste capítulo foi dada uma visão global do cenário atual da busca por otimizar a utilização de recursos em ambientes de computação na nuvem. O trabalho aqui proposto se baseia em vários conceitos apresentados nesses trabalhos, já que consideramos a premissa de ociosidade de recursos em nuvem que é motivação das instâncias oportunistas, além de buscarmos um nível seguro de *overcommit* por meio de monitoramento e previsões de carga da nuvem. Por fim, fazemos uso também de predição de tempo de tarefas Hadoop, o que nos faz ter um melhor controle do uso de recursos, tanto a nível operacional, como a nível de aplicação.

Nosso trabalho se difere por buscar prever tanto a utilização da nuvem, como a duração da aplicação, no caso, tarefas Hadoop, já que consideramos uma nuvem do tipo PaaS com esse fim. Essa abordagem nos dá uma granularidade que possibilita um controle eficiente dos recursos. Adicionalmente, consideramos também eficiência energética dos servidores, procurando executar em uma zona mais eficiente do que acontece em uma nuvem ociosa. Nenhum trabalho que tenhamos conhecimento considera ao mesmo tempo essas três frentes.

No próximo capítulo será descrito o método proposto nessa pesquisa. Detalharemos cada um dos preditores utilizados, bem como o algoritmo que toma as decisões de escalonamento.

Capítulo 3

Abordagem

Este capítulo detalha a abordagem utilizada neste trabalho, que envolve a arquitetura da solução proposta e os métodos preditivos considerados, que executam tanto a nível operacional, observando a utilização dos recursos da nuvem, como a nível de aplicação, analisando as tarefas Hadoop que foram e serão executadas. Por fim, são mostrados os cenários possíveis nas tomadas de decisão de acordo com as saídas destes preditores.

3.1 Arquitetura da solução proposta

Como mostrado no Capítulo 1, o OpenStack possui um componente de processamento de dados como um serviço (DPaaS, do inglês, *Data Processing as a Service*), chamado Sahara, tendo sua arquitetura exibida na Figura 1.1. O fluxo normal de uso de um ambiente pré-configurado do Sahara é: (i) Usuário requisita criação de um *cluster*; (ii) Sahara invoca o Nova para criar um *cluster* de VMs, de acordo com a disponibilidade de cota no projeto; (iii) Sahara configura o *cluster*, de acordo com o *plugin* escolhido (Hadoop, Spark, Storm); (iv) Usuário submete aplicação e dados, pela interface do Sahara; (v) Usuário executa tarefa no *cluster*.

Além disso, já foi mencionado que o processo de verificar a disponibilidade de recursos para criação de um *cluster* no OpenStack é controlado pelo sistema de cotas, principalmente do Nova. Esse sistema de cotas, entretanto, considera apenas valores estáticos alocados para os usuários. Esta abordagem costuma ser ineficiente no que se refere a aproveitamento de recursos, uma vez que alocados os recursos, estes não poderão ser alocados para outros

usuários, mesmo que não estejam sendo de fato utilizados. Essa limitação é especialmente relevante quando consideramos recursos sujeitos a alta demanda como CPU e RAM. Nesse cenário, é comum haver máquinas físicas repletas de VMs, que encontram-se ociosas ou com baixa utilização na maior parte do tempo, causando um uso subótimo do poder computacional disponível na nuvem.

Este trabalho propõe uma solução mais eficaz, que, fazendo uso da volatilidade intrínseca do modelo de PaaS, considera a utilização real de recursos, em detrimento do sistema de análise estática de cotas. Para tanto, esta solução faz uso de predição de utilização de recursos a nível operacional, estimando o quão utilizadas as máquinas físicas que compõem a nuvem estarão em uma curta janela de tempo, além de utilizar predição a nível de aplicação, buscando estimar quanto tempo uma tarefa levará para ser executada. Estas informações possibilitam fazer um melhor escalonamento, tendo uma visão mais precisa sobre o que está sendo utilizado na nuvem no momento, assumindo também o próximo estado da nuvem. A arquitetura da solução proposta é exibida em detalhes na Figura 3.1.

Antes da chamada ao Sahara, que criaria o *cluster* de acordo com a necessidade do usuário, há um novo componente, responsável por prever o estado da nuvem, e tomar as devidas decisões. Esse novo componente, proposto nesta pesquisa, é dividido em três subcomponentes. São eles:

- Preditor de carga da nuvem, que considera dados históricos de utilização de CPU e memória RAM das máquinas físicas que compõem a nuvem, executando previsões de carga a respeito das próximas janelas de execução. Detalhes desse componente serão apresentados na Seção 3.2;

- Preditor de tempo de execução de tarefas, que faz uma estimativa, a partir de agrupamento de tarefas semelhantes anteriormente executadas, da duração que determinada tarefa levará para ser executada. Detalhes desse componente serão apresentados na Seção 3.3;

- Avaliador, que, com base nos dados dos dois preditores acima, toma a decisão a respeito da tarefa, que pode ser: executa normalmente no momento, rejeita, adia, ou acelera a execução. Detalhes desse componente serão apresentados na Seção 3.4.

As próximas seções detalham cada um desses três componentes.

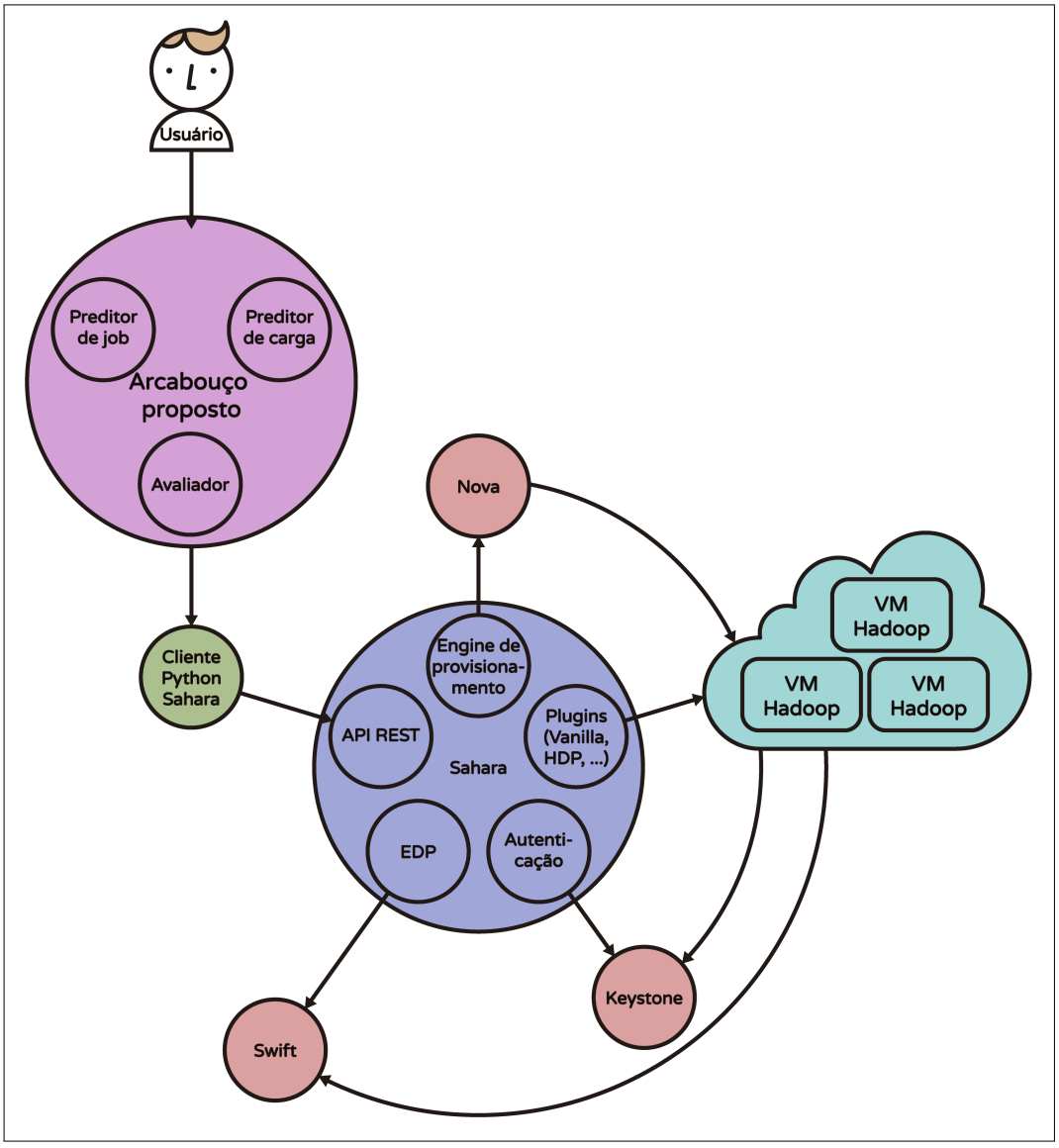


Figura 3.1: Sistema proposto. Adição de um componente para análise da nuvem, antes de se comunicar com o Sahara para criação de *clusters* e executar tarefas. O sistema de cotas do Nova é desconsiderado nesse novo processo.

3.2 Predição de carga da nuvem

Nesta seção, detalhamos o processo utilizado para estimar a carga da nuvem em um momento futuro do tempo, a partir de um histórico de curto prazo de utilização de CPU e RAM de servidores da nuvem.

Predição de carga é uma técnica vastamente utilizada para auxiliar no gerenciamento de recursos em computação na nuvem, seja ela executada a curto prazo, prevendo o estado nos próximos minutos como no trabalho de Carvalho *et al.* [24] e Roy *et al.* [41], ou uma abordagem a mais longo prazo, detectando um padrão de dias ou até semanas [47]. Para tal, os preditores disponíveis na literatura e no mercado variam em sua complexidade, indo desde os mais triviais, como um repetidor (que apenas considera que o valor da janela atual vai se manter na próxima janela), até preditores que combinam técnicas mais avançadas como transformadas de Fourier [42] e cadeias de Markov [49].

Este trabalho não tem como objetivo criar preditores de carga mais eficientes ou robustos. E, sim, mostrar que mesmo com preditores simples é possível obter resultados satisfatórios e uma melhora no gerenciamento de recursos da nuvem. Nosso processo considera janelas de predição, de 5 minutos, cada uma representando um ponto coletado, com histórico de uma hora, para prever a utilização na próxima janela, totalizando assim 12 janelas em cada histórico a ser avaliado. Com base nos dados de utilização de CPU e memória RAM coletados na última hora, são usados simultaneamente, para então definir um consenso entre o número de VMs previsto, os seguintes preditores:

- *Naive*: Valor futuro é igual ao último valor observado. Preditor simples e muito usado como *benchmarking* em comparação com preditores mais complexos;
- Média móvel: Predição dos valores futuros é igual à média dos valores do histórico de predição;
- *Drift*: Considera a variação entre o início e o fim da janela de previsão. Segue a tendência de crescimento ou redução da janela. Esse método faz uma extrapolação linear, usando como base o primeiro e o último ponto da janela para estimar o próximo ponto no futuro;
- RWF: Do inglês *Random Walk Forecast*, similar ao *Drift*, mas escolhe a tendência

de crescimento ou queda de forma aleatória. Aqui, o fator de alteração é calculado da mesma forma do *Drift*, porém, a decisão sobre se esse valor vai ser acrescido ou diminuído é feita de forma aleatória. Esse preditor é muito usado para estimar variáveis com padrões econômicos que oscilam muito, como taxas de câmbio e de bolsas de valores.

Uma vez obtidos os valores percentuais de utilização de CPU e RAM estimados pelos preditores para as próximas janelas, esses valores são usados para determinar a quantidade de máquinas virtuais disponíveis nessa janela. O cálculo desse número de máquinas é feito usando a relação entre a capacidade total e a estimada livre dos servidores e o tamanho das máquinas virtuais a serem alocadas. A título de exemplo, consideremos máquinas virtuais com $2\ VCPUs$ e $3\ GB$ de RAM, e um servidor com total de $8\ CPUs$ e $16\ GB$ de RAM. Em um cenário que o preditor retorna estimativas de 60% e 50% para CPU e RAM, respectivamente, haverá aproximadamente $3\ CPUs$ e $8\ GB$ de RAM disponíveis. Isso significa que, nessa janela, será possível criar 1 VM, já que a quantidade de CPUs disponíveis impossibilita a alocação de mais uma, embora haja RAM disponível. O limite é dado pelo menor número de VMs dentre os dois recursos, ou seja, a RAM pode eventualmente ser o limitador da quantidade de VMs disponíveis.

No momento da chegada de cada tarefa na nuvem, são calculadas as quantidades de VMs disponíveis de acordo com o resultado retornado por cada um dos preditores, e, em caso de divergência, o veredito da predição é dado utilizando a moda do número de VMs de cada preditor.

3.3 Predição de tempo de tarefas Hadoop

Nesta seção, detalhamos nosso processo para estimar a duração de tarefas Hadoop. Tarefas Hadoop, em geral, tem poucos elementos não-determinísticos, o que as torna possíveis de ter sua duração estimada com acurácia satisfatória. O conhecimento da duração de uma tarefa permite que o provedor de nuvem faça um melhor gerenciamento da infraestrutura física, podendo controlar melhor as variações de utilização a curto prazo.

Entre os modos para predição de tarefas na literatura, há trabalhos usando regressão linear [28; 44] e modelagem [19; 23]. Nesta pesquisa, fazemos uso de agrupamento, utilizando

um algoritmo bastante comum, chamado KNN (do inglês *K-Nearest Neighboors*, K Vizinhos mais Próximos).

O KNN consiste de um algoritmo que, dados um conjunto de fatores que compõem um objeto, e um parâmetro K, monta um espaço e identifica os K elementos mais próximos, de acordo com uma determinada métrica de distância. A partir destes K vizinhos mais próximos, é tomada a decisão de como classificar o novo elemento. Essa decisão pode ser feita por sorteio ou moda, em variáveis qualitativas, ou através da média, quando inferindo variáveis quantitativas. A Figura 3.2 mostra uma execução simples do KNN em um plano bidimensional, usando distância euclidiana para classificar os novos elementos, na qual uma variação do parâmetro K implica em uma mudança na classificação.

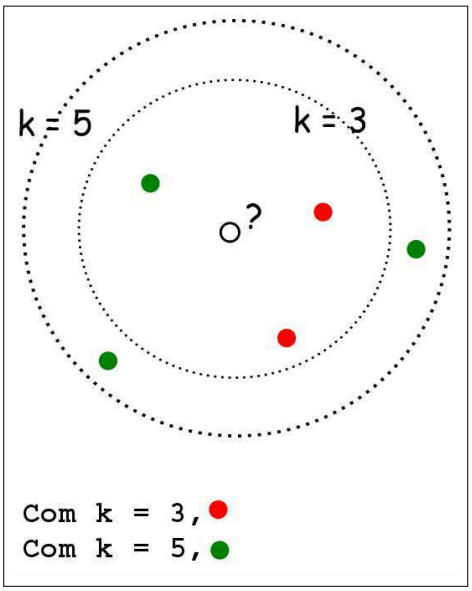


Figura 3.2: Exemplo simples de KNN usando moda para tomada de classificação de uma variável qualitativa. O elemento *O* pode ser categorizado como vermelho, utilizando *k=3*, ou verde, com *k=5*. *Fonte: http://quipu-strands.blogspot.com.br/2014/08/knn-classifier-in-one-line-of-python.html*

Utilizamos este algoritmo para prever o tempo de execução de uma tarefa Hadoop recém chegada na nuvem, a partir de uma base de dados de tarefas executadas anteriormente, variando parâmetros da tarefa. Os parâmetros usados no KNN foram:

1. Tamanho do *cluster*: Variando entre 3 e 10 nós trabalhadores (do inglês, *workers*);

2. Tipo da tarefa: Foram consideradas 3 tipos de tarefas básicas de *benchmarking* do Hadoop. São elas *Word Count*, *Tera Sort* e *Pi Estimator*. Aqui, o algoritmo recebe o nome da tarefa para agrupamento;

3. Tamanho da entrada: Com três níveis (pequeno, médio e grande) para cada tipo de tarefa, e, finalmente;

4. Número de *reducers* a executar, variando de acordo com o tamanho do *cluster*.

Esse modelo funciona especialmente bem em ambientes onde a tarefa precisa ser executada frequentemente, recalculando um modelo, como é comum na arquitetura *lambda*, por exemplo. Neste tipo de arquitetura, uma mesma aplicação é executada periodicamente, alterando apenas os dados de entrada, e é esperado que o modelo consiga uma aproximação razoável do tempo de execução com base no tamanho da entrada.

A escolha dos tipos de tarefas acima é significativa, devido ao fato de que elas usam diferentes tipos de recursos da nuvem, tornando o ambiente mais próximo do real, ilustrando variados tipos de cargas de trabalho que podem ser executadas sobre Hadoop. As aplicações MapReduce aqui utilizadas estão disponíveis no pacote de exemplos do Hadoop [9]. O primeiro deles, o *WordCount*, é o exemplo clássico de job MapReduce. A entrada consiste em um arquivo de texto e a saída é um arquivo com o número de ocorrências de cada palavra na entrada. A função *map* separa as linhas do texto em palavras e o resultado é agregado localmente com uma função intermediária, chamada de *combiner*. O *reduce* soma os valores nos conjuntos agregados, gerando a quantidade total de ocorrências de cada palavra. Esta aplicação é utilizada como um *benchmark* de aplicações que realizam operações simples (filtragem, conversão ou agregação simples) em cima de um grande volume de dados, fazendo uso principalmente de E/S.

O segundo tipo de tarefa é o *TeraSort*, que é executado em duas etapas, sendo a primeira chamada de *TeraGen*, na qual é criado um arquivo de texto com uma dada quantidade de linhas, cada uma com caracteres aleatórios, e em seguida, retorna este arquivo com as linhas ordenadas alfabeticamente. Em contraste com o *WordCount*, esta tarefa combina a leitura de dados com um uso intenso de memória e CPU.

Finalmente, há o *Pi Estimator*, que estima o valor de *Pi* usando o método quasi-Monte Carlo, que gera pontos em posições aleatórias dentro de um círculo inscrito em um quadrado

unitário, onde a probabilidade de um ponto estar inscrito no círculo é proporcional às áreas relativas do círculo e do quadrado. Esta tarefa recebe como entrada parâmetros que definem o quão preciso o valor será estimado. Neste trabalho os parâmetros passados foram X e 5000, ou seja, serão executados X *maps* e para cada *map* há cinco mil amostras (pontos gerados). Além disso, o número de *reducers* é fixo em 1 pelo próprio algoritmo. Esta aplicação faz uso majoritariamente de CPU.

A Tabela 3.1 ilustra os valores de entrada de cada tarefa, incluindo os valores de X do *Pi Estimator*.

Tabela 3.1: Níveis e fatores variados na formação da base de dados de tarefas Hadoop usada no KNN.

Tipo de tarefa	Unidade de medida	Pequena	Média	Grande
Word Count	Gigabytes (GB)	2	3	5
Pi Estimator	Número de *maps*	400	1000	1600
Tera Sort	Número de linhas	35000000	60000000	75000000

Tendo executado estas tarefas repetidas vezes e medido seus tempos de execução, o próximo passo foi validar esta base no KNN. Detalhes e resultados da avaliação desta base encontram-se na Seção 4.1.4.

A execução da aplicação, entretanto, é apenas a segunda parte do fluxo Hadoop. Em um ambiente PaaS, onde os recursos são naturalmente voláteis, os *clusters* utilizados para execução são criados imediatamente antes do processo e eliminados ao fim deste processo. Embora esses *clusters* sejam criados muito mais rapidamente do que seriam manualmente configurados por humanos, esse processo ainda leva alguns minutos, e não pode ser desconsiderado no momento de escalonamento de tarefas Hadoop. Sendo assim, podemos considerar a criação do *cluster* Hadoop como a primeira parte da execução de uma tarefa. Nesta solução, estimamos também o tempo de criação de *clusters*, fazendo uso de um método mais simples do que a estimativa de carga e tarefas, de acordo com a quantidade de VMs *workers* neste *cluster*.

Assim, estimamos o tempo total de execução de uma tarefa Hadoop como sendo:

$$T_{total} = T_{cluster} + T_{tarefa} \tag{3.1}$$

A Seção a seguir explica como são usados esses dados providos pelos preditores de carga da nuvem e de tempo de tarefas Hadoop.

3.4 Avaliador

Ao longo desta seção, consideramos um modelo de serviço na nuvem onde o usuário escolhe: (i) Número mínimo de *workers* no *cluster* para executar; (ii) Executável da sua tarefa, e; (iii) Dados de entrada, sendo o provedor responsável pelos demais quesitos, como parâmetros de rede e imagens do sistema operacional. Aqui descrevemos o algoritmo usado no avaliador, para, a partir das informações retornadas pelos preditores de carga e tempo de tarefa, tomar a decisão sobre a tarefa.

Estas decisões podem ser: (i) Adiar a tarefa, caso a mesma chegue em um momento que a nuvem esteja carregada, mas haja recursos disponíveis em curto prazo; (ii) Acelerar a tarefa, caso haja recursos disponíveis no momento e previsão de que continue havendo; (iii) Rejeitar a tarefa, caso não haja recursos disponíveis no presente momento e na próxima janela de predição, e finalmente; (iv) Executar a tarefa normalmente, com a quantidade de recursos solicitada.

As subseções a seguir detalham este processo.

3.4.1 Janelamento

As decisões de escalonamento são feitas baseadas em janelamento. É recomendável que o tamanho da janela seja igual ao intervalo de coleta de dados de utilização considerados pelo preditor de carga, a fim de obter uma granularidade consistente com as predições. A partir do valor estimado, em segundos, para completar a tarefa, contando tempo de criação do *cluster* e de execução da tarefa em si, encontramos a quantidade de janelas necessária usando a Equação 3.2.

$$N_{janelas} = \lceil \frac{T_{total}}{T_{janela}} \rceil \tag{3.2}$$

Onde $N_{janelas}$ é o número de janelas necessárias para executar a tarefa, T_{total} é o tempo total de execução, em segundos, considerando criação do *cluster* e execução da tarefa em ri, e T_{janela} é o tamanho da janela, em segundos. Vale observar que, a fim de evitar subestimar o

tempo de execução, o número de janelas é o teto dessa divisão, dando um pequeno fator de conservadorismo.

Assim sendo, no momento que o avaliador recebe uma nova tarefa, é feita uma requisição para o preditor de tarefa a fim obter o tempo estimado de executar esta tarefa de acordo com os atributos selecionados (aplicação, tamanho de entrada, tamanho do *cluster* e número de *reducers*) para então converter esse tempo em janelas. A título de exemplo, uma tarefa que teve sua execução calculada em 1600 segundos, é estimada em 6 janelas de 300 segundos (5 minutos), usando o teto.

Tendo a quantidade de janelas necessárias para executar determinada tarefa, o próximo passo é consultar o preditor de carga para verificar a disponibilidade de máquinas virtuais nas próximas janelas. Então, é feita uma requisição ao preditor de carga da quantidade de máquinas disponíveis no momento atual até as próximas $2 * N_{janelas}$, já que adiamos em, no máximo, $N_{janelas}$ para executar em, no máximo, mais $N_{janelas}$, evitando degradar a expectativa do usuário em mais que o dobro do tempo esperado.

3.4.2 Processo de avaliação

Tendo esses dados, o avaliador agora vai analisar as informações e tomar a devida decisão: adiar, acelerar, executar normalmente ou rejeitar. A seguir, nas próximas subseções, descrevemos em que cenário cada uma destas decisões é tomada.

Para tanto, consideremos:

- $min_{cluster}$ o número mínimo de *workers* no *cluster* que o usuário selecionou para executar a tarefa;
- $N_{janelas}$ o número de janelas necessárias para executar a tarefa com $min_{cluster}$ *workers*;
- $L = \left(N_0, N_1, ..., N_{2*N_{janelas}}\right)$, onde N_0 é o número de VMs disponíveis no atual momento, sem predição, N_1 representa o número de VMs na primeira janela de predição, e assim sucessivamente;
- $max_{cluster}$ o número máximo de *workers* que o *cluster* pode ter após o escalonamento. Esse número deve ser um tamanho de *cluster* grande arbitrário, escolhido de acordo com a capacidade da nuvem.

- $t_{cluster}$ o tamanho do *cluster* no qual a aplicação vai, de fato, ser executada;
- J_{espera} o número de janelas que a tarefa vai esperar para ser executada. Usada no caso de adiamento.

3.4.2.1 Aceleramento

O aceleramento acontece quando a requisição da tarefa é recebida em um momento que o preditor de carga detecta que haverá mais máquinas disponíveis nas próximas $N_{janelas}$ do que $min_{cluster}$.

Nesse caso, o tamanho de *cluster* a ser escolhido para executar a tarefa será o menor valor dentre $max_{cluster}$ e o menor valor entre $N_1 ... N_{N_{janelas}}$. Seja min_L o menor elemento da sequência $L[1 : N_{janelas}]$, temos:

$$t_{cluster} = min(min_L, max_{cluster}) \tag{3.3}$$

A Figura 3.3 ilustra esse caso.

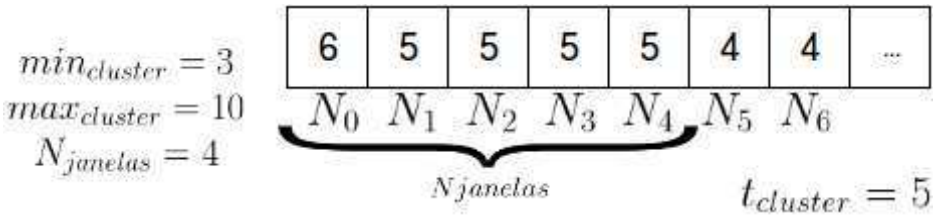


Figura 3.3: Exemplo de uma janela de previsão que resulta em um aceleramento.

No exemplo da Figura 3.3, o tamanho de *cluster* com o qual será executada a tarefa é de 5 VMs, já que o usuário definiu um $min_{cluster}$ de 3 máquinas e o preditor detectou mais máquinas disponíveis para executar uma tarefa que levaria $N_{janelas}$ janelas com a configuração mínima. Assim sendo, podemos afirmar que é seguro alocar 5 VMs para essa tarefa, considerando que esse aumento de VMs diminui o tempo de completá-la. Na Seção 4.1.4 executamos experimentos para observar em que níveis vale a pena aumentar o tamanho do *cluster*, observando a linearidade do tempo das tarefas conforme cresce o número de *workers*.

3.4.2.2 Adiamento

Ao contrário do aceleramento, o adiamento de tarefa acontece quando, no momento da requisição, não há $min_{cluster}$ máquinas disponíveis, mas o preditor estima que entre N_1 e $N_{janelas}$ haverá uma quantidade de máquinas maior. Com isso, a tarefa será postergada para um momento próximo com maior quantidade de VMs disponíveis, que execute em um tempo menor do que executaria com $min_{cluster}$ máquinas. Uma tarefa será adiada se:

$$\exists\, a, 1 \leq a \leq N_{janelas} \mid N_a > min_{cluster} \tag{3.4}$$

Neste cenário, há mais uma variável que deve ser considerada, que chamaremos de J_{espera}. Essa variável representa o número de janelas que a tarefa vai esperar até ser iniciada sua execução. Esse número corresponde ao índice de a na lista L, que satisfaz a condição mostrada na Equação 3.4. Vale ressaltar aqui, que verificamos a disponibilidade em apenas uma janela no futuro, uma vez que os preditores apresentam tendência de queda ou crescimento uniforme, sem variação do sinal ao longo das janelas de predição.

A Figura 3.4 mostra um caso onde ocorre adiamento da tarefa.

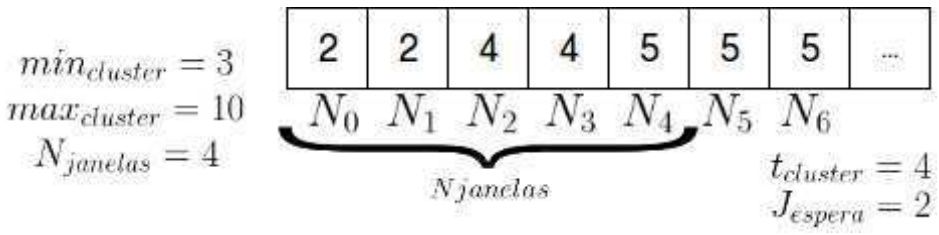


Figura 3.4: Exemplo de uma janela de previsão que resulta em um adiamento de tarefa.

Aqui, a tarefa chegou em um momento N_0 que não há recursos mínimos disponíveis, porém, o preditor detectou que em 2 janelas haverá mais recursos do que o mínimo necessário. Temos aqui um cenário em que a tarefa será executada 2 janelas depois, com um *cluster* com um *worker* a mais do que o mínimo previsto. Apesar da tarefa executar com um número maior de nós, esse processo não garante que o tempo inicial previsto da tarefa não será afetado, uma vez que o tempo esperando pelos recursos de J_{espera} janelas pode ser menor que o ganho de tempo aumentando o tamanho do *cluster*. O que garantimos aqui é

que a tarefa do usuário não será rejeitada, embora não haja recursos disponíveis no momento, além de manter a utilização da nuvem alta em eventual vale de utilização no futuro.

3.4.2.3 Rejeição

O próximo passo é avaliar se há, de alguma forma, recursos disponíveis para essa execução. Caso não haja, a tarefa será rejeitada. Em um sistema de nuvem normal, essa rejeição seria dada pelo sistema de cotas. Aqui, esse processo é mais complexo, e tende a ocorrer com menos frequência, uma vez que são considerados os recursos de fato utilizados, e não apenas alocados das VMs.

A rejeição acontece quando não há recursos disponíveis no momento da execução, nem há previsão de haver nas próximas $N_{janelas}$ janelas. Formalizando, temos uma tarefa rejeitada se:

$$\nexists\, a, 1 \leq a \leq N_{janelas} \mid N_a \geq min_{cluster} \tag{3.5}$$

A Figura 3.5 exemplifica esse cenário. Aqui, vemos um cenário onde não há máquinas disponíveis no atual momento, e o cenário tende a piorar nas próximas janelas. Assim, a tarefa é rejeitada.

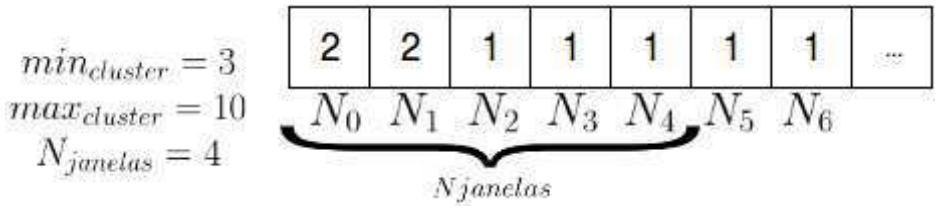
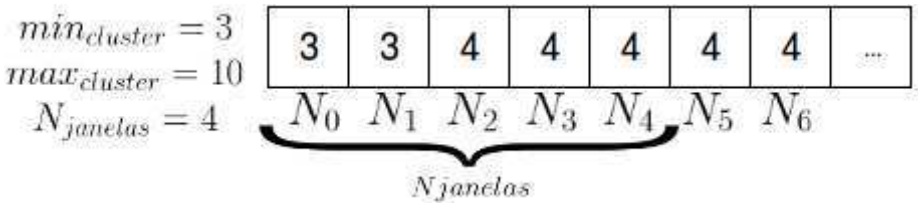


Figura 3.5: Exemplos de janelas de previsão que resultam em rejeição da tarefa.

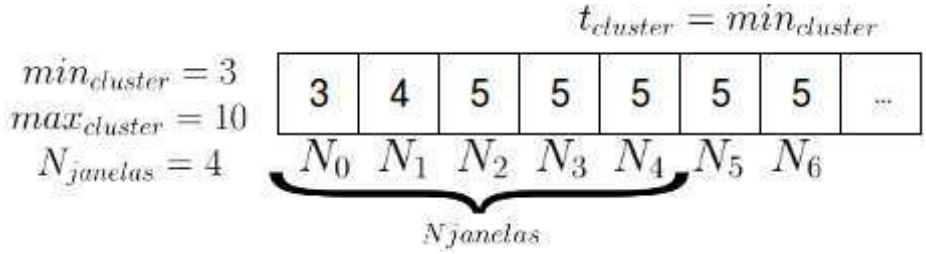
3.4.2.4 Execução normal

Finalmente, caso não haja nenhuma ação de aceleramento ou adiamento para ser tomada com a tarefa, mas não haja falta de recursos suficiente para impedir seu processamento, voltamos ao cenário mais trivial, que é a execução normal, que acontece sem adiamento, no momento da chegada da tarefa, e sem aumentar o tamanho do *cluster*, ou seja, usando apenas $min_{cluster}$ *workers*. Este cenário ocorre quando o número de máquinas disponível no momento é igual ao necessário e não há previsão de crescimento suficiente nas próximas

janelas. A tarefa será executada no fluxo normal, que aconteceria com frequência nos casos de sistemas controladas por cota, onde quando a aplicação não é rejeitada, é executada com o número requisitado pelo usuário. A Figura 3.6 mostra dois casos onde nenhuma ação não-trivial é tomada no escalonamento.



(a) Um caso de execução normal



(b) Um segundo exemplo de execução normal

Figura 3.6: Exemplos de janelas de previsão que resultam em uma execução normal.

Em ambos os casos, o fluxo segue normalmente, pois, de acordo com os dados retornados pelo preditor de tempos de tarefas, não vale a pena esperar duas janelas (Figura 3.6a), ou uma (Figura 3.6b), supondo que os ganhos de ter essa pequena quantidade a mais de recursos não compensam. Na Seção 4.1.4 há mais detalhes sobre esses ganhos de desempenho conforme o tamanho do *cluster* aumenta;

Em suma, a abordagem consiste em quatro análises: adiantar a execução, caso haja mais recursos livres que o requisitado; adiar a execução, caso haja os recursos necessários em um curto prazo, mas não no momento, ou rejeitá-la, caso os recursos não estejam disponíveis nem no momento, nem no curto prazo. Por fim, caso nenhum dos três cenários anteriores seja avaliado com sucesso, a execução segue normalmente, apenas com a quantidade mínima de recursos solicitada.

O próximo capítulo detalha a instanciação dessa solução bem como os resultados obtidos.

Capítulo 4

Avaliação e Resultados

Neste capítulo, descrevemos o ambiente criado para instanciação da ferramenta, com objetivo de validar a solução aqui proposta em um ambiente real, partindo dos pré-experimentos para validação de algumas premissas do sistema, e em seguida apresentamos e discutimos os resultados obtidos. Os experimentos aqui executados têm como objetivo simular um ambiente mais próximo do real e validar a solução proposta.

4.1 Pré-experimentos

Aqui, mostramos como foi validado o ambiente utilizado nos experimentos desta pesquisa. Essa validação envolve medições de consumo de energia em servidores da nuvem utilizada, injeção de carga a partir de dados públicos de nuvens da Google, além da avaliação de predição de carga e da predição de tempo de tarefas Hadoop. As próximas subseções detalham cada um destes pré-experimentos.

4.1.1 Validando injeção de carga

Para criar um ambiente mais próximo de uma nuvem real nos experimentos, foi gerada uma carga sintética, baseada em registros (*traces*) públicos de utilização de máquinas do *data center* da Google [4]. Esses *traces* contêm informações de utilização de CPU e RAM de um *cluster* de 12.500 máquinas da Google ao longo de um mês. Reiss *et al.* [40] mostraram que essas utilizações são em média de 50% e 60%, para os casos de CPU e RAM respectiva-

mente.

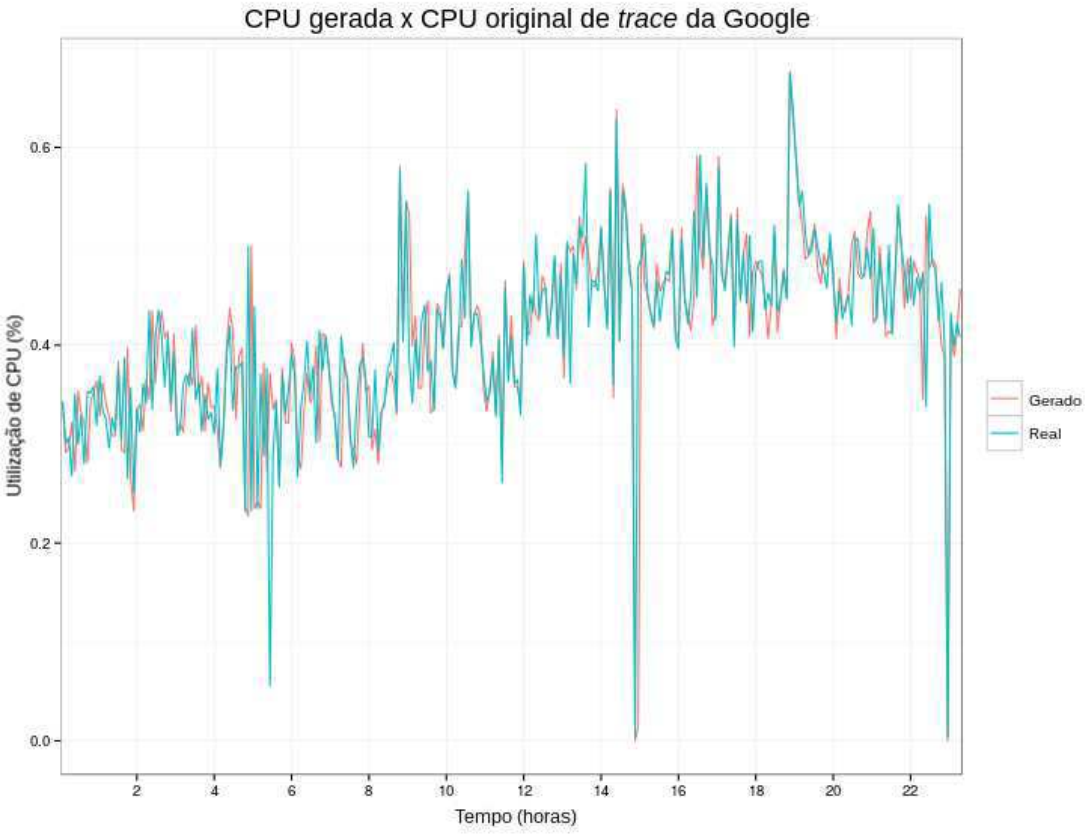


Figura 4.1: Comparação entre carga gerada pelo *lookbusy* e carga real, coletada de dados de *data center* do Google. Dados praticamente se sobrepõem.

Com base nos dados de utilização de máquinas individuais do *cluster* da Google, geramos uma carga de CPU sintética nos servidores da aplicação, usando a ferramenta *lookbusy* [12]. O *lookbusy* é um *script* Python simples que pode gerar carga de CPU, RAM e I/O em máquinas, com volumes definidos pelo usuário. 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* [16], a utilização real de CPU da máquina. Assim como acontece nos *traces* originais da Google, a carga foi dividida em período de 5 minutos, bem como a coleta.

Aqui, tivemos um erro médio próximo de 1% considerando a diferença entre a CPU real e a CPU gerada pelo *lookbusy* A Figura 4.1 ilustra o resultado deste experimento. Pode-

mos observar que a geração de carga consegue ser muito fiel à carga original, criando um comportamento similar ao de máquinas de *data centers* de produção reais.

Essa injeção de carga será utilizada nos experimentos principais na Seção 4.2.

4.1.2 Proporcionalidade de energia

Como mencionado antes, servidores computacionais em geral não têm consumo de energia proporcional à utilização de seus recursos. Executamos este experimento para ilustrar o quanto a utilização de recursos (CPU, no nosso caso) do servidor interfere no consumo de energia. Para tanto, dispomos nesse experimento de um servidor HPE *ProLiant BL460c Gen8*, com dois processadores Intel Xeon E5-2630L, cada um com 6 núcleos operando a 2GHz. Neste servidor, geramos uma carga de CPU sintética de 0 a 100% de utilização total, incrementando 5% a cada 5 segundos, e depois retornando no mesmo ritmo a 0%. Esta geração de carga é feita, assim como na Subseção 4.1.1, utilizando o *lookbusy*. Um recurso particular dessa linha de servidores da HPE se refere ao modo de economia de energia dos servidores, podendo o administrador da infraestrutura escolher entre o modo de desempenho máximo, modo de economia máxima, ou um intermediário modo dinâmico, que faz ajustes visando a um equilíbrio entre eficiência energética e desempenho dos servidores.

Para coleta de dados de potência do servidor, foi utilizado o OneView [10], um *software* da HPE para gerenciamento de infraestruturas de TI em empresas. O OneView oferece uma interface gráfica e uma API REST para monitoramento de *status* de saúde dos servidores, além de diversas métricas de utilização como consumo de energia e temperatura. Além disso, o OneView é capaz de operar diretamente em parâmetros da BIOS dos servidores, facilitando a configuração.

A Figura 4.2 mostra o resultado deste experimento. Na parte superior, temos a utilização de CPU gerada, enquanto na parte inferior, podemos observar a evolução do consumo de energia conforme varia a CPU. Observamos que o consumo de energia do servidor bom baixa utilização (no começo e no fim do gráfico) varia entre 85 e 90 W (Watts), enquanto o servidor com carga máxima consome entre 145 e 150 W. Em outras palavras, um servidor ocioso consome cerca de 60% da energia de um servidor sendo utilizado ao máximo, o que reforça a ideia de que devemos evitar ter servidores pouco utilizados, procurando manter sua utilização em níveis altos. Já na Figura 4.3, vemos o quanto a taxa consumo de energia em relação à

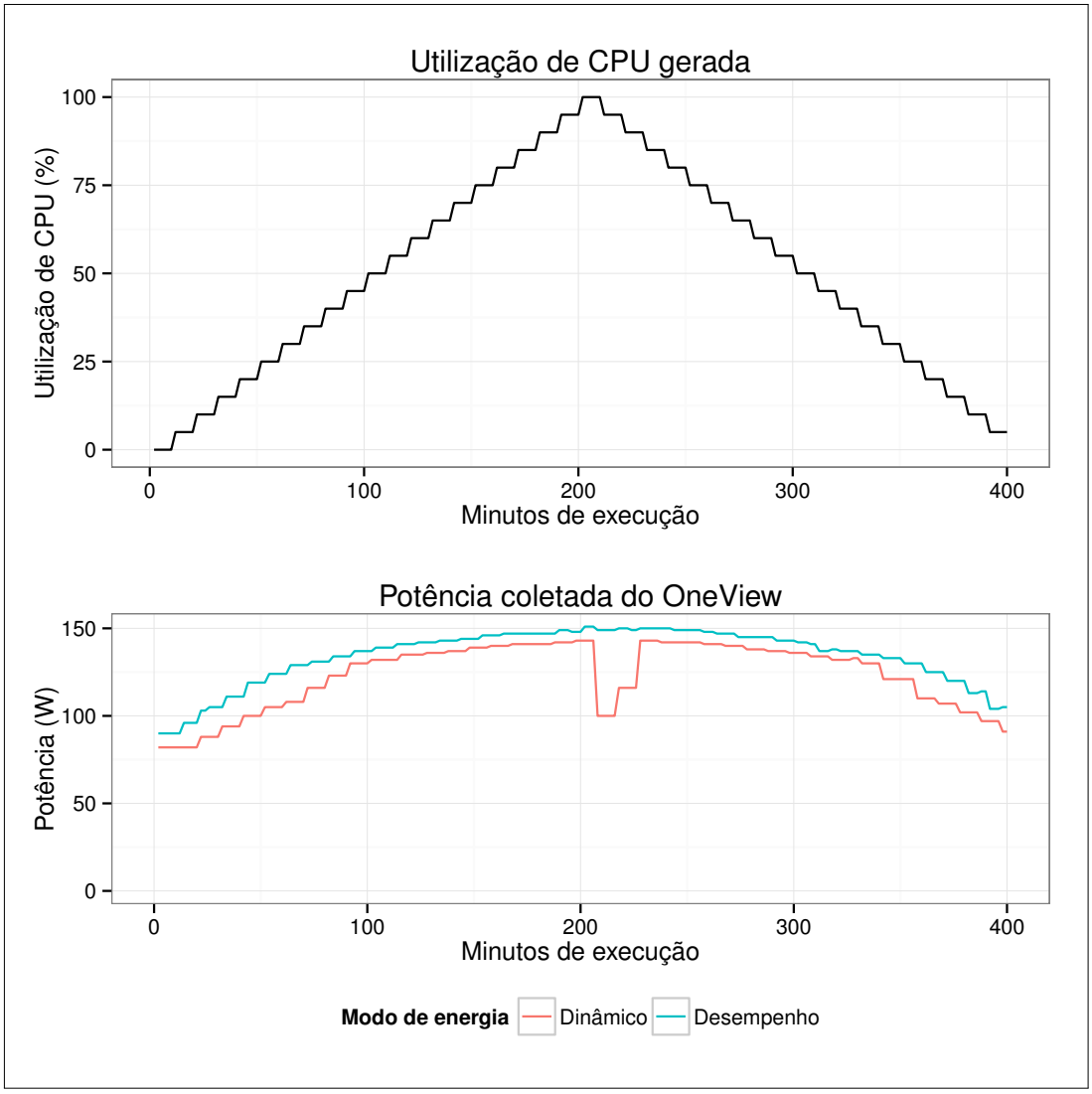


Figura 4.2: Acima, o gráfico de utilização de CPU gerada pelo *lookbusy*. Abaixo, o consumo de energia ao longo do tempo, conforme aumenta utilização de CPU, nos modos de energia dinâmico e de desempenho. Modo dinâmico apresenta uma pequena decaída ao atingir os 100% devido a otimizações feitas pelo *hardware*.

utilização de CPU é otimizada conforme o consumo de energia cresce. A taxa decresce conforme cresce o uso de CPU, o que significa que o preço de unidades computacionais tende a se reduzir.

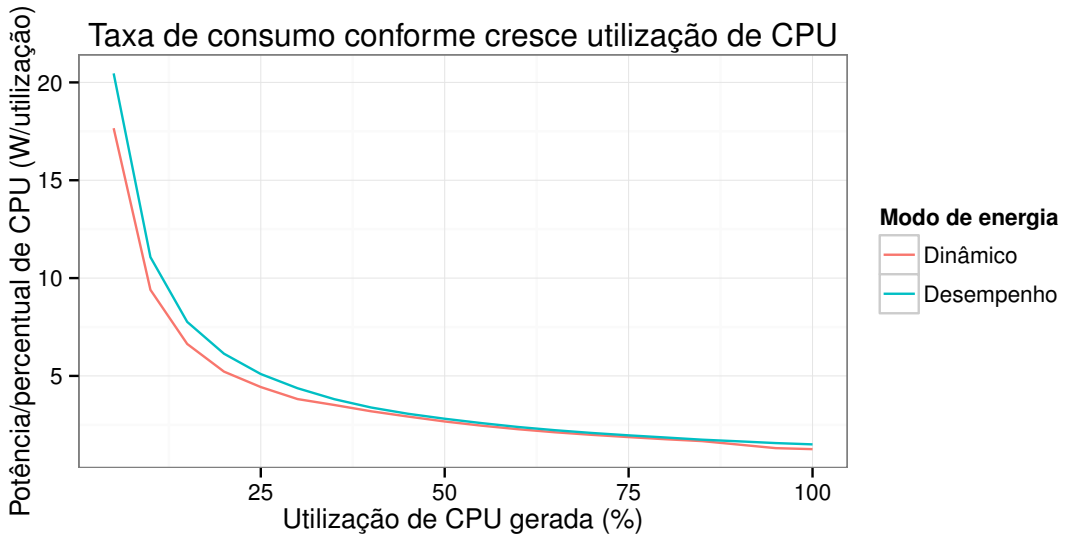


Figura 4.3: Otimização da taxa Potência/Utilização (eixo Y) conforme a utilização de CPU (eixo X) aumenta. Quanto menor o valor, mais eficiente é a taxa.

Concluímos assim, que neste tipo de servidor, quanto maior a utilização de CPU, mais eficiente energeticamente o servidor se comporta. Precisamos nesse caso, encontrar um nível de utilização que seja alto o suficiente em termos energéticos, mas que seja seguro, minimizando o risco de falhas ou degradação de desempenho. Nosso ambiente de experimentos para execução de tarefas Hadoop, dispõe de dois servidores deste modelo. Essas medições servirão para medir com mais precisão o impacto energético e financeiro da solução proposta nesta pesquisa. Mais detalhes sobre o cenários destes experimentos e a decisão do ponto ótimo de utilização desejado são discutidos na Seção 4.2.

4.1.3 Predição de carga da nuvem

Nesta seção, mostramos como se comportaram os 4 preditores (média, *drift*, *naive* e RWF) de carga utilizados. Para tanto, usamos dados de utilização dos *traces* da Google e executamos a predição na forma de uma janela deslizante. Em seguida, comparamos a quantidade de

máquinas virtuais estimadas pelo preditor com a quantidade de VMs disponíveis de fato na próxima janela de predição. A Figura 4.4 compara a quantidade de VMs prevista pelo consenso utilizando a moda, com a quantidade real de VMs disponíveis em uma máquina física da Google selecionada aleatoriamente. O valor 0 predomina na maior parte do tempo, o que dá indícios de que o preditor fazendo o consenso de acordo os 4 outros age bem nesse caso.

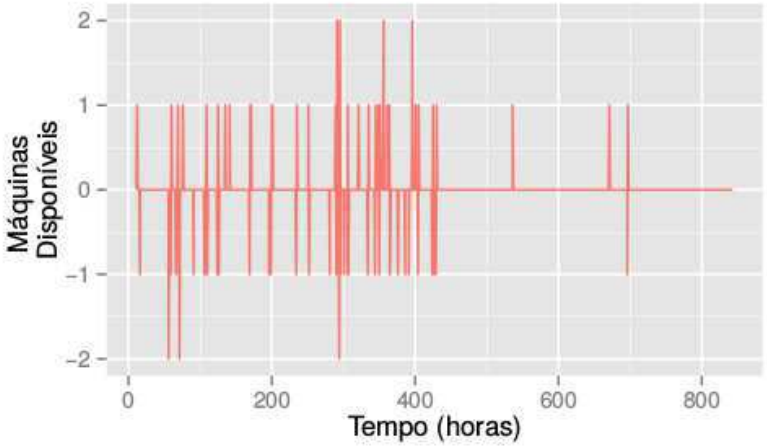


Figura 4.4: Diferença entre o número de VMs estimado pelo consenso dos preditores com o número real disponível. O eixo Y representa a diferença entre *número real disponível* − *número previsto* Máquina do *cluster* do Google com ID 121306.

Finalmente, a Figura 4.5 mostra o *boxplot* dessa diferença, coletado ao longo do tempo. O valor 0, ou seja, o preditor acertou o número de VMs disponíveis, predomina em cerca de 80% do tempo, enquanto um erro de apenas 1 máquina virtual (para mais ou para menos) é de cerca de 14%. Finalmente, temos o pior caso, que é um erro maior ou igual que 2 VMs de diferença, ocorrendo em apenas 5% dos casos. Mesmo nesse pior caso, degradação de desempenho não deve ser notada, uma vez que as máquinas excedentes dificilmente gastarão toda sua capacidade de processamento. Os preditores utilizando o consenso tem uma média de acerto maior que 80%.

No Apêndice B há outros gráficos referentes a esse preditor, mostrando que o resultado observado é similar em outras máquinas do *cluster* da Google.

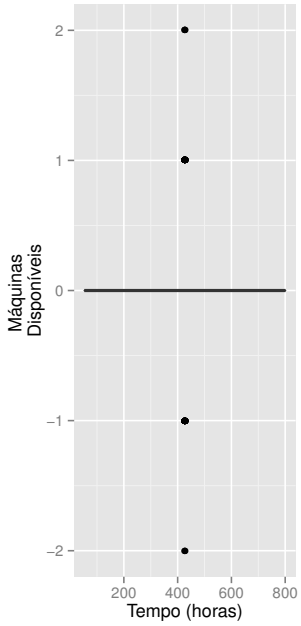


Figura 4.5: *Boxplot* da diferença entre o número de VMs estimado pelo consenso dos preditores com o número real disponível. Mediana, primeiro e terceiro quartis encontram-se em 0, o que mostra que a predominância do acerto dos preditores. Máquina do *cluster* do Google com ID 121306.

4.1.4 Predição de tempo de tarefas Hadoop

O tempo de execução de uma tarefa Hadoop no nosso modelo consiste em somar o tempo de criação do *cluster* com o tempo de execução da tarefa em si. Nesta seção descrevemos o experimento usado para estimar cada uma dessas etapas.

4.1.4.1 Tempo de criação do *cluster*

Observamos em nosso ambiente que o tempo de criação de um *cluster* não pode ser desconsiderado. A Figura 4.6 mostra os tempos de criação de um *cluster* pelo Sahara de acordo com o número de *workers*. Podemos observar que o tempo médio varia entre 5 e 9 minutos. Considerando que grande parte das tarefas Hadoop executadas em ambientes de produção tem duração média de 30 minutos [28], este tempo não pode ser desconsiderado.

Usamos na solução principal uma estimativa do tempo de criação de um *cluster* com base nestes dados. Consideramos a média de cada quantidade como o tempo necessário para criar

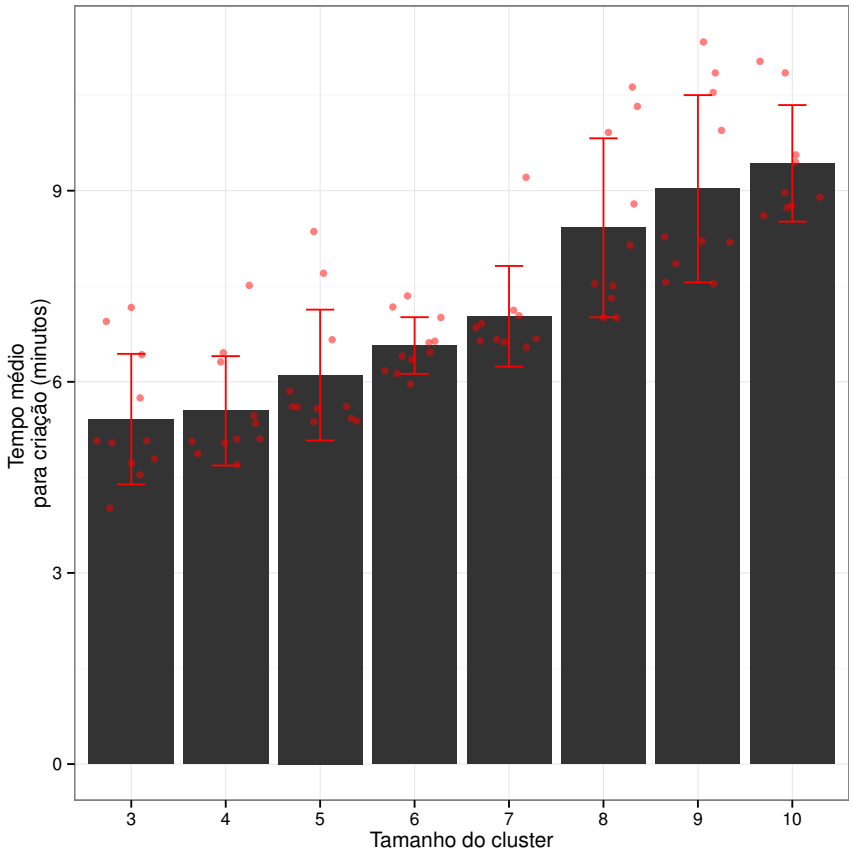


Figura 4.6: Tempo médio para criação de um *cluster* Hadoop pelo Sahara no nosso ambiente, conforme cresce o número de nós *workers*.

um *cluster* de determinado tamanho.

4.1.4.2 Execução da tarefas Hadoop

Foi feito também um experimento para validar o funcionamento do preditor de tempos de tarefas Hadoop. Para isso, foram executadas 10 repetições de cada combinação dos seguintes fatores, calculando seu tempo de execução, a fim de montar uma base de dados:

- Tipo de tarefa: *Word Count, Pi Estimator* e *TeraSort*;
- Tamanho da entrada: 3 níveis (pequena, média e grande) para cada tipo de tarefa;
- Tamanho do *cluster*: De 3 a 10 nós *worker*.

Para estes dados, definimos o número de *reducers*, que é uma configuração escolhida pelo usuário diretamente, e que tem considerável influência no tempo de duração da tarefa. Já a quantidade de *maps* é normalmente inferida de acordo com o tamanho da entrada e de algumas configurações internas do Hadoop que são bem menos frequentemente alteradas pelos usuários, já que têm impacto consideravelmente menor na execução. O número de *reducers* só pode ser alterado para as tarefas *Tera Sort* e *Word Count*.

Para definição do número de *reducers*, consideramos os valores definidos em documentos do Apache Hadoop [32] e utilizados em trabalhos relacionados (por exemplo, [37; 34]). Esta abordagem é descrita pela Equação 4.1.

$$N_r = f.N_n.M_t \tag{4.1}$$

Onde N_r é o número de *reducers*, f é uma constante, N_n é o número de nós no *cluster* e M_t é o máximo de *maps* que podem ser realizados em um mesmo *TaskTracker*. Destes valores, M_t foi mantido constante e igual a 2, que é o valor da configuração padrão do Hadoop, e N_n varia de 3 a 10. Já para o valor da constante f, usamos o valor de $0,95$. A intuição para esta constante, mostrada em [34] é que todos os *reducers* serão normalmente executados simultaneamente, deixando ainda um pouco de espaço ocioso caso ocorram problemas e algum precise ser reexecutado.

É importante destacar alguns pontos quanto ao padrão observado dos dados. A título de exemplo, vemos na Figura 4.7 os resultados da execução da tarefa Pi Estimator variando o tamanho de entrada e do *cluster*, com os valores exibidos na Tabela 3.1. Para esta tarefa é possível notar um ganho de desempenho conforme cresce o tamanho do *cluster*. Já na Figura 4.8 há um certo ganho conforme cresce o tamanho do *cluster*, mas o ganho não é tão grande quanto o anterior. Finalmente, a Figura 4.9 mostra os dados do experimento feito para o *Word Count*. Em alguns momentos, o acréscimo de máquinas virtuais se mostrou prejudicial à execução da tarefa. Esse fator negativo pode ser explicado por uma limitação de banda no ambiente utilizado. A conexão entre os servidores que executam o processamento das máquinas virtuais e o *cluster* Ceph que armazena os seus dados, bem como o Swift, que é executado o *Wordcount* fornece uma banda de apenas $1\ GB/s$, limitando uma maior vazão que resultaria em um ganho de desempenho conforme se aumentasse o número de máquinas do *cluster* Hadoop.

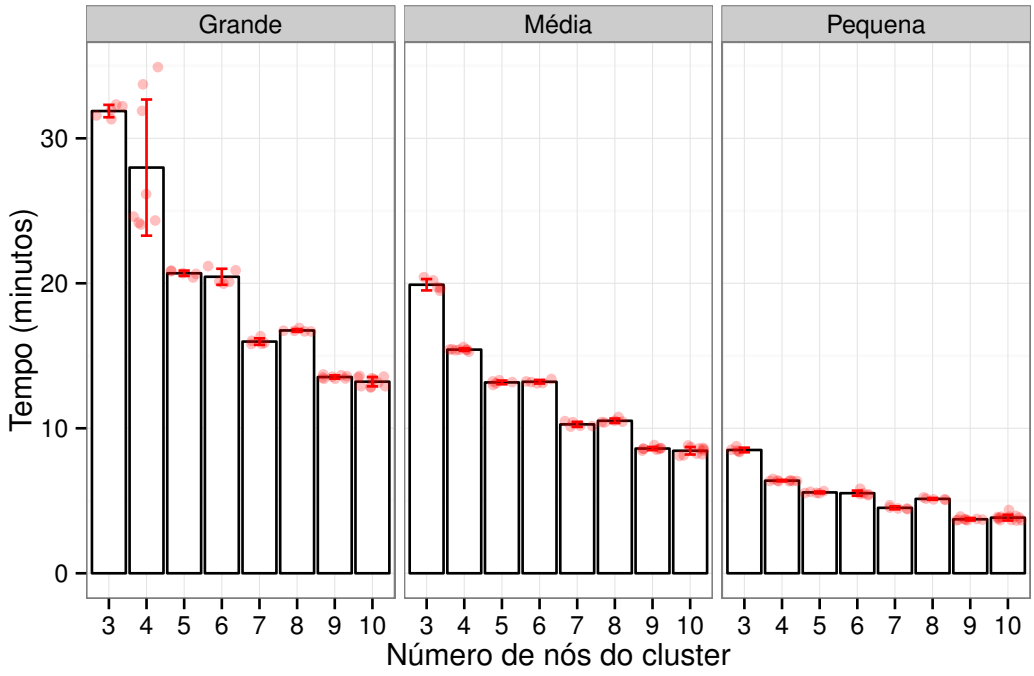


Figura 4.7: Tempos médios de execução da tarefa *Pi Estimator*, variando o tamanho da entrada e do *cluster*.

Uma vez montada essa base, foi executado o KNN de forma iterativa, uma vez para cada registro, a fim de prever o tempo de cada execução. Assim, a cada iteração, era selecionado um registro da base para servir como conjunto de teste, enquanto o restante agia como conjunto de treino. Os parâmetros passados para estimar o tempo total da tarefa no KNN foram descritos na Seção 3.3. O algoritmo usado internamente para encontrar os K vizinhos mais próximos foi o KD-Tree, implementado no pacote FNN [27]. O KD-Tree consiste de formar árvores multidimensionais muito utilizadas para resolver problemas como este de encontrar vizinhos mais próximos de uma forma mais eficiente.

Desta forma, estimamos o valor previsto de cada execução, e montamos um conjunto no qual foi possível comparar o tempo real com o tempo previsto. Essa comparação é mostrada na Figura 4.10. Já na Tabela 4.1, podemos ver o sumário desses erros para cada tipo de tarefa.

Com esses resultados, temos evidências de que o preditor funciona bem em um ambiente que tarefas similares são executadas com frequência, com variação dos dados de entrada. Nas

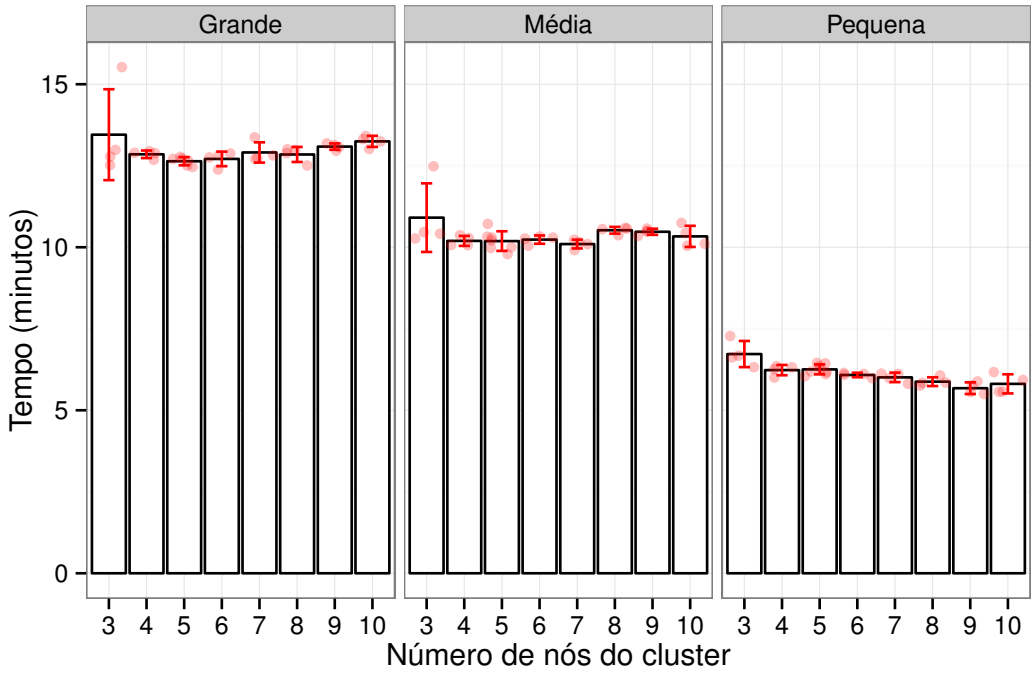


Figura 4.8: Tempos médios de execução da tarefa *Tera Sort*, variando o tamanho da entrada e do *cluster*.

próximas seções, descrevemos em mais detalhes os ambientes do experimento que mostram a eficácia da ferramenta proposta.

4.2 Ambiente dos Experimentos

Para os experimentos a seguir, dispomos de uma nuvem, controlada pelo OpenStack versão Kilo com o Sahara, também na versão Kilo. Em termos de nós de computação, essa nuvem era composta de 4 servidores 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* e em modo dinâmico de energia. Destes 4 servidores, 2 eram dedicados exclusivamente para este experimento, tendo cada um 60 GB de memória RAM. Além disso, a nuvem tem um nó controlador e um nó de rede, conectando estes servidores por uma rede de 1 Gbps. O armazenamento das máquinas virtuais é feito em um *cluster* Ceph com 3 nós, que é um sistema de armazenamento distribuído, construído para prover armazenamento

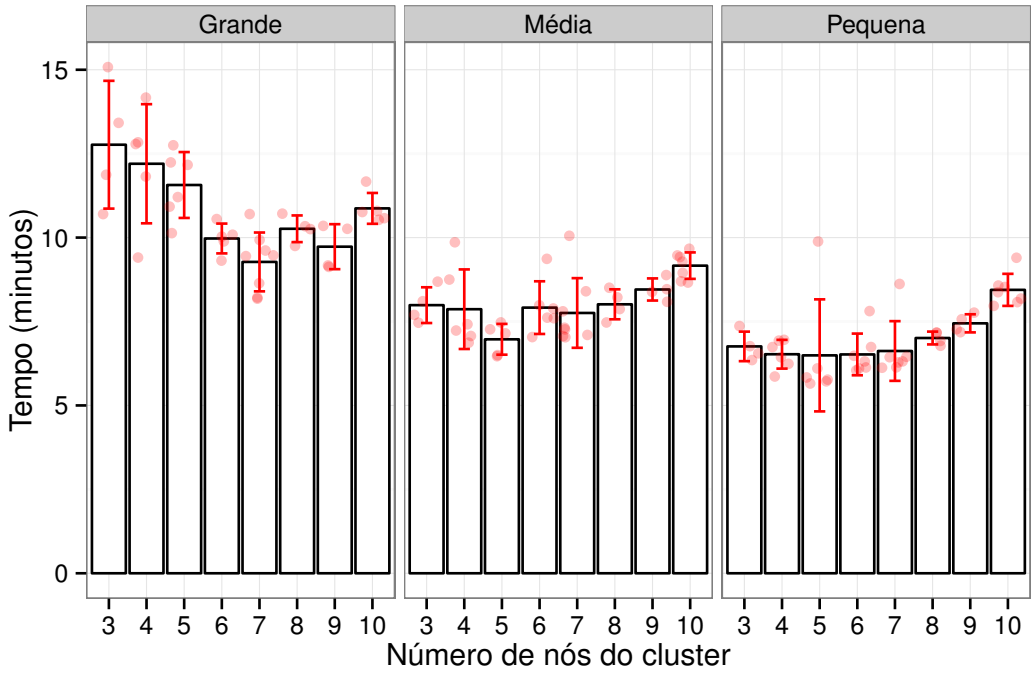


Figura 4.9: Tempos médios de execução da tarefa *Word Count*, variando o tamanho da entrada e do *cluster*.

escalável com alto desempenho, dispondo neste ambiente, 3 TB de espaço para armazenamento. De acordo com a última pesquisa de usuários e operadores do OpenStack [14], o Ceph é a solução mais utilizada para este tipo de armazenamento, estando presente em cerca de 40% dos sistemas de produção em todo o mundo. Além disso, o Ceph também serve de *backend* de armazenamento de dados do Swift, que é utilizado na tarefa *Word Count* para armazenar arquivos de entrada e saída.

Nessa nuvem, foram executados experimentos que consistiam de 60 execuções de tarefas descritas anteriormente, selecionando aleatoriamente o tipo, tamanho de entrada, do *cluster* e número de *reducers*, sendo disparadas a cada 10 minutos. Os experimentos foram executados sem previsão, usando o sistema normal de cotas do OpenStack, e utilizando a solução aqui proposta, considerando a utilização dos recursos. Durante essas execuções, é gerada uma carga de CPU e RAM nos servidores a partir de dados de servidores da Google, com o objetivo de ter um ambiente com carga real. Escolhemos três *traces* diferentes, sendo um onde as máquinas são altamente utilizadas, um onde as máquinas tem pouca utilização e um

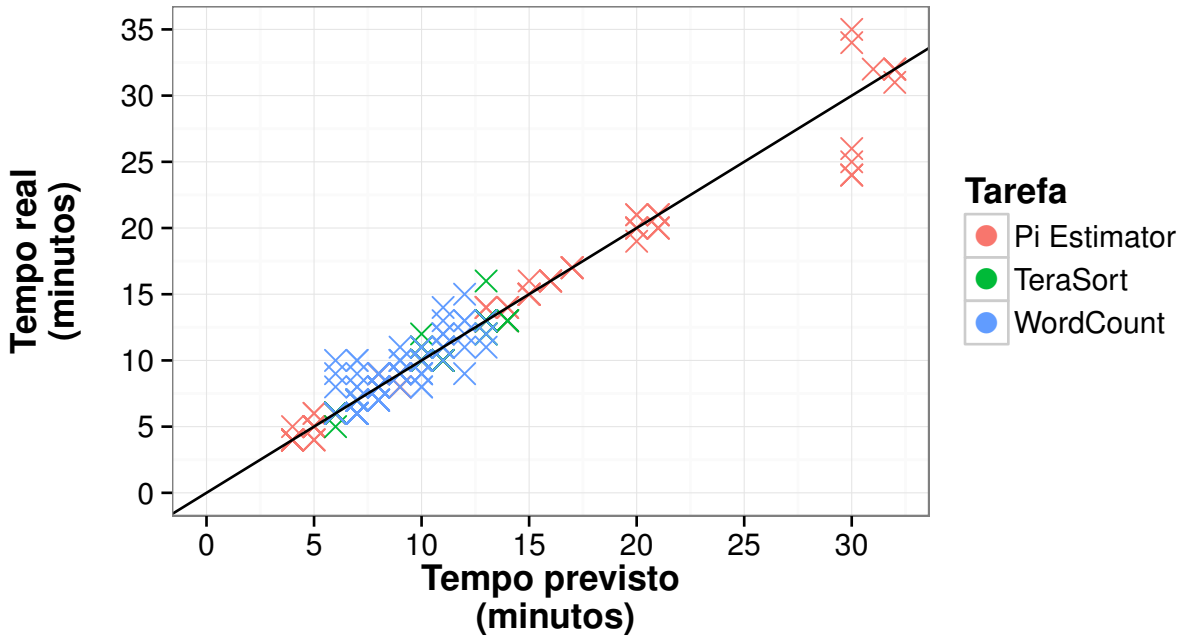


Figura 4.10: Erro do algoritmo KNN com a base de dados montada.

caso intermediário. Os *traces* escolhidos estão disponíveis no Apêndice A.

O janelamento utilizado no preditor e coleta de dados de carga considera janelas de 5 minutos. O sistema aqui utilizado foi implementado utilizando a linguagem Python para as atividades em geral, R para os cálculos de predição, além de alguns *scripts* Shell para coleta de dados em sistemas Linux.

4.3 Métricas de avaliação

Para fins de avaliação, utilizamos as seguintes métricas:

1. Utilização de CPU e memória dos servidores que compõem a nuvem, coletadas individualmente a cada 5 minutos de execução. Cada dado coletado equivale à média observada nessa última janela de tempo, para então ser feito o acompanhamento destes dados ao longo do tempo;

2. Percentual de tarefas processadas, onde verificaremos se a solução proposta processa

Tabela 4.1: Erros percentuais do KNN para cada uma das tarefas executadas.

Tarefa	Erro (%)
Word Count	8,2
Pi Estimator	2,9
Tera Sort	3,4

com sucesso mais requisições do usuário do que uma nuvem controlada por um sistema comum de cotas de usuário;

3. Tempo de execução de tarefas, onde mostramos que nossa abordagem não causa interferências no tempo de conclusão das tarefas, mesmo aumentando a utilização média dos recursos da nuvem, resultando em transparência de serviço para o cliente;

4. Consumo de energia da nuvem, estimado por meio do modelo descrito na Seção 4.1.2.

A seção a seguir descreve nossa avaliação de cada uma dessas métricas.

4.4 Resultados

Nesta seção, expomos os resultados obtidos pela instanciação da solução, iniciando com a prova de conceito com apenas a tarefa *Pi Estimator* e, em seguida, com os três tipos de tarefas aqui apresentados.

4.4.1 Execução apenas com tarefa *Pi Estimator*

Para início de validação, a fim de ter uma prova de conceito da solução, foi executado o experimento considerando apenas a tarefa *Pi Estimator*. Essa decisão foi tomada devido a esta tarefa ser a que é menos afetada pelo ambiente que dispusemos para testes. Nosso ambiente de nuvem é impactado por não conseguirmos custear exclusivamente para desenvolvimento uma nuvem com alta vazão de rede entre o nó controlador e os nós de computação. Em ambientes reais, essa vazão é atingida com redes de 10Gbps, enquanto contávamos apenas com uma rede de 1 Gbps que, como mostrado anteriormente, vem a ser gargalo em tarefas que exijam alto volume de rede, especialmente com *clusters* grandes, que aumentam o volume

de dados transferidos. O *Pi Estimator* é menos impactado por essa configuração, pelo fato de ser uma tarefa que faz uso majoritário apenas de CPU.

Na Figura 4.11 é possível ver como o *Pi Estimator* apresenta um ganho conforme o tamanho do *cluster* cresce, mesmo considerando o tempo de criação do *cluster*, devido à baixa vazão de rede que esse tipo de tarefa demanda.

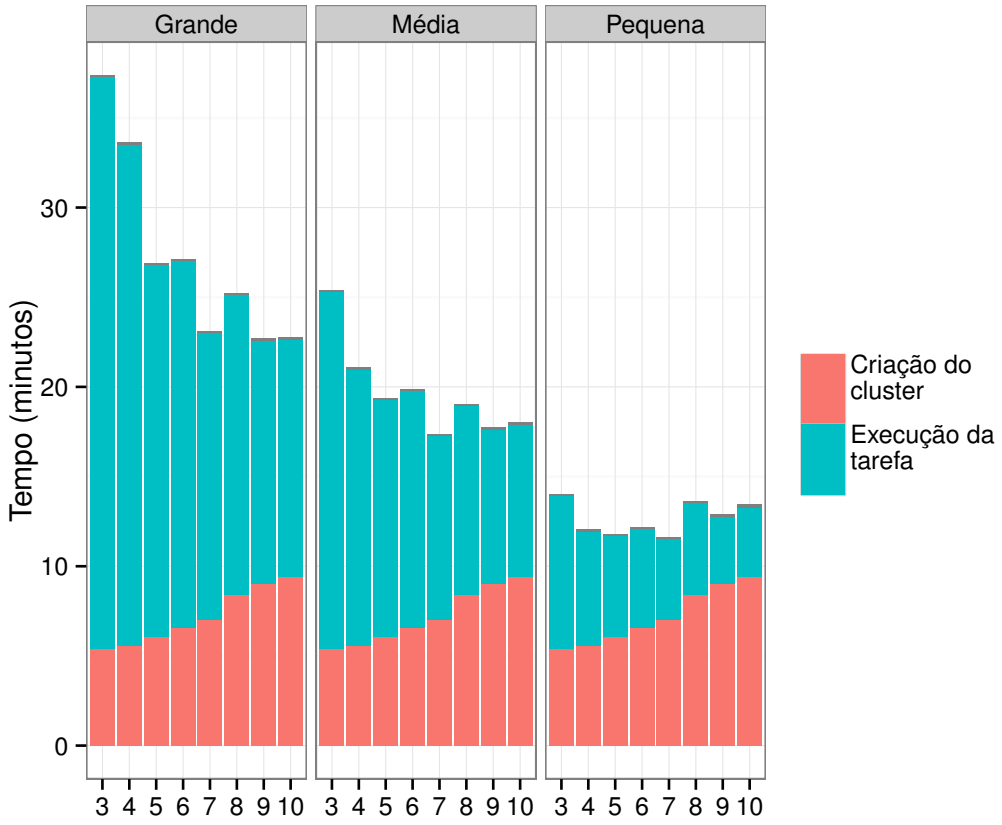


Figura 4.11: Tempo médio de execução da tarefa *Pi Estimator* somado com o tempo de criação do *cluster* e variando o tamanho de entrada.

Apesar disso, nessa figura já é possível ver empiricamente tomada de decisões não triviais, como por exemplo, saber que para a execução com a maior entrada, uma configuração com 7 nós é mais vantajosa do que com 8 nós em termos de tempo total de execução. A solução implementada considera esses dados, melhorando a perspectiva do usuário, já que ele não vai precisar contratar mais recursos por mais tempo, além de melhorar para o pro-

vedor de nuvem, que com os números ótimos de *cluster*, vai ter mais espaço para atender outros clientes.

A título de exemplo, no cenário de nuvem pública, consideremos o sistema de preços do Amazon Elastic MapReduce [8], no qual um usuário que faz uso de um sistema com arquitetura *lambda* e processa um lote de dados Hadoop uma vez por dia tem, em um cenário como o exibido, onde o preditor analisou que o processamento com 7 nós é mais rápido do que com 10, um custo mensal reduzido de R$ 270 para aproximadamente R$ 150. Este cenário implica uma redução de cerca de 40% no custo mensal para o cliente, já que menos máquinas serão requisitadas e a tarefa demorará menos tempo para ser concluída.

Já a afirmação do ponto de vista do provedor de nuvem é ilustrada com auxílio da Figura 4.12, que mostra o percentual de tarefas processadas e tarefas rejeitadas, em cenários com e sem predição de recursos. Tivemos aqui um aumento de 12% de tarefas processadas, em relação ao caso sem predição.

Esse aumento da proporção de tarefas completas resulta em um aumento da utilização média dos recursos, como mostra o box plot da Figura 4.13. Neste cenário, o ganho de utilização de CPU foi de 5%, enquanto o de RAM é de 11%. Em ambos os casos, há diferença estatística significante. Esse aumento na utilização dos recursos, junto com a maior proporção de tarefas processadas, implica em mais lucro para o provedor. Adicionalmente, a Figura 4.14 mostra esse aumento ao longo do tempo.

Consideraremos aqui o preço do kWh como sendo $R\$0,43668$, como descrito no site da Energisa [17][1]. Os servidores tiveram uma utilização média de 47%, o que implica em um consumo médio de 131W, de acordo com o consumo observado na Seção 4.1.2, utilizando o modo dinâmico destes servidores. A Equação 4.2 explica o consumo total de energia mensal com os servidores neste ambiente:

$$C_{energia} = P_{media} * N_{computes} * N_{horas} * 0,001 \tag{4.2}$$

Onde P_{media} é a potência média obtida dos servidores, $N_{computes}$ é o número de servidores (2, em nosso exemplo), e N_{horas} é o número de horas no qual deseja-se avaliar. Aqui, consideraremos o período de um mês, portanto, $N_{horas} = 720$. Por fim, multiplicamos pela constante $0,001$ para obter o valor em kWh. Assim sendo, temos um consumo mensal to-

[1]Considerando tarifa do tipo B3 para classe "Comercial, serviços e outros". Acessado em 20/07/2016.

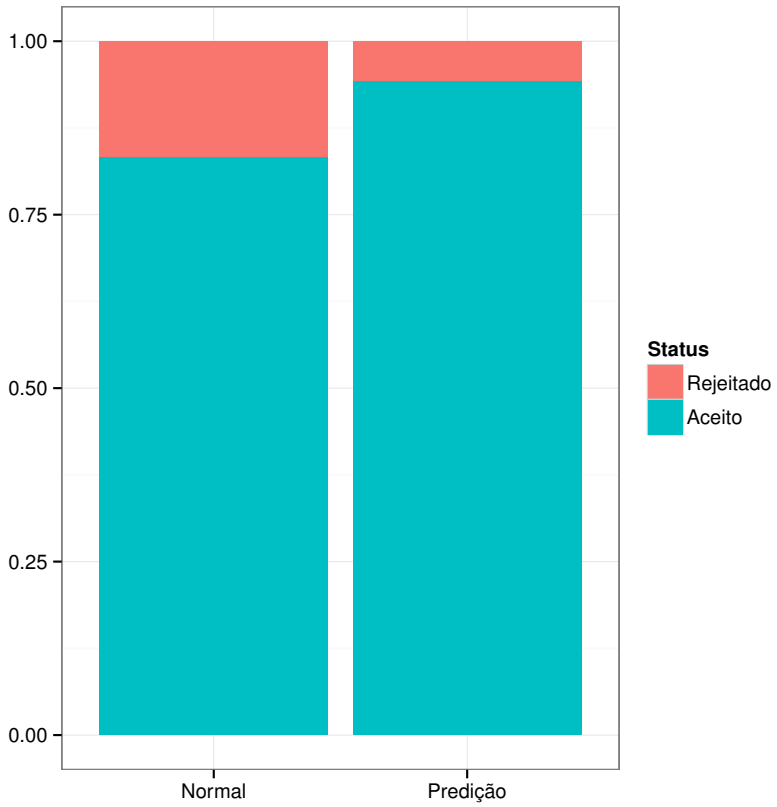


Figura 4.12: Percentual de tarefas processadas com sucesso comparando casos com e sem predição.

tal de 189 kWh para manter estes dois servidores operando, e um custo total de cerca de $R\$82,50$.

Já no caso sem predição, obtivemos uma utilização média de 42%, resultando em uma potência de 127W, com um consumo de 182 kWh, a um custo de $R\$79,50$. Apesar de o ganho de utilização ter sido significante (5% de CPU e 11% de RAM), o custo operacional da infraestrutura aumentou apenas cerca de R$ 3, equivalente a um acréscimo de 3%. Esse custo adicional, entretanto, é facilmente diluído no valor ganho com as tarefas a mais processadas.

Em nosso ambiente, considerando tarefas em *cluster* médio, com 7 nós, onde as tarefas têm duração média de 30 minutos, o custo médio por tarefa seria de cerca de 3 reais, no Amazon EMR. Na frequência de chegada testada no nosso ambiente, são requisitadas cerca de 160 tarefas por dia, das quais o ambiente sem a solução proposta processa 84%, enquanto

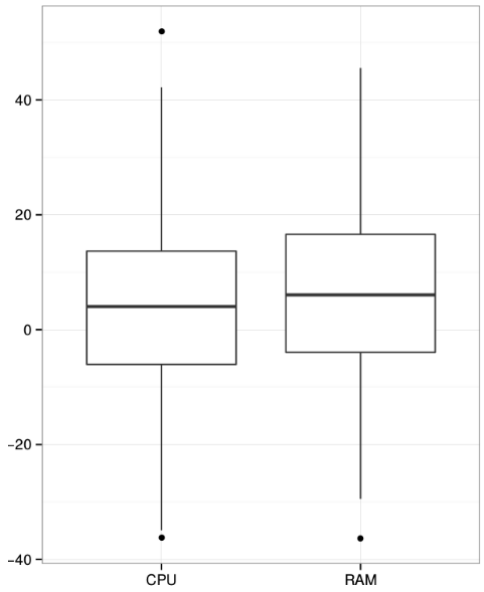


Figura 4.13: Box plot da diferença da utilização média de CPU e RAM ao longo do tempo com e sem predição. Em ambas as métricas, a mediana encontra-se acima de 0, o que significa que na maior parte do tempo o caso com predição registrou utilização maior. Em ambos os casos foi observada diferença estatística significativa.

a prova de conceito aqui mostrada conseguiu atender em média 94% da demanda. A Tabela 4.2 ilustra estes valores, e o lucro médio obtido nesse cenário.

Tabela 4.2: Lucro financeiro médio a partir dos resultados obtidos nestes experimentos.

Ambiente	Tarefas processadas	Valor bruto (R$)	Custo operacional mensal (R$)	Lucro mensal (R$)
Normal	132	490	80	410
Com predição	150	545	83	462

Esse ganho aproximado de 15% no lucro já mostra-se significativo em uma pequena nuvem com apenas 2 nós de computação. Apenas a título de exemplo, mantendo os níveis médios de utilização de recursos e de processamento de tarefas com sucesso, uma nuvem com 1000 nós geraria um aumento de lucro na ordem de dezenas de milhares de reais. Adicionalmente, o uso dessa solução preditiva, apesar de aumentar a carga média da nuvem, não causou impacto no tempo médio de execução das tarefas, como mostra a Figura 4.15.

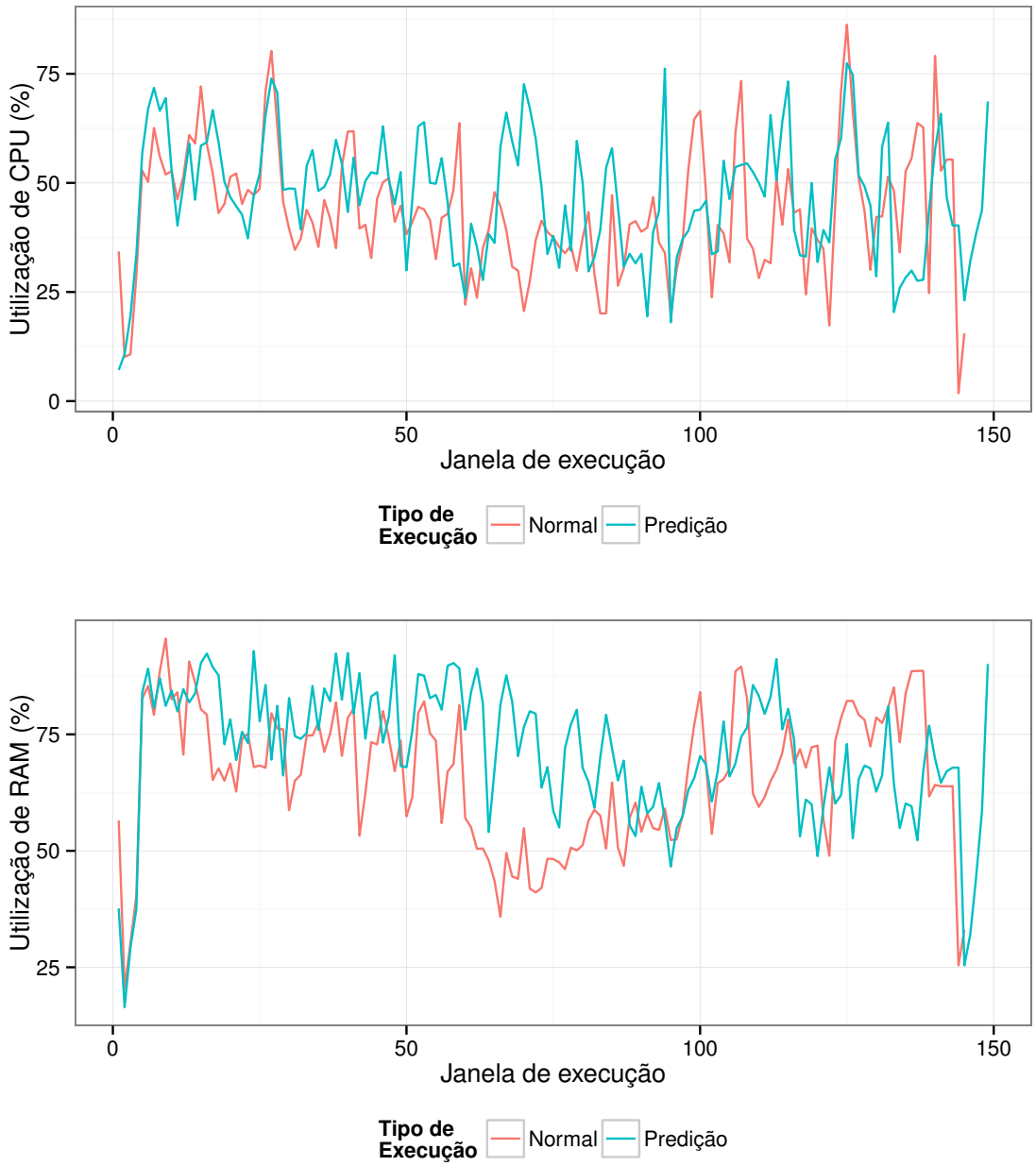


Figura 4.14: Utilização média de CPU e RAM nos dois servidores ao longo do tempo. Cada ponto corresponde à média ponto a ponto agrupada dos dois servidores considerando as 10 execuções em cada caso. As execuções com predição apresentam um aumento de 5% na utilização média de CPU e 12% na utilização de RAM.

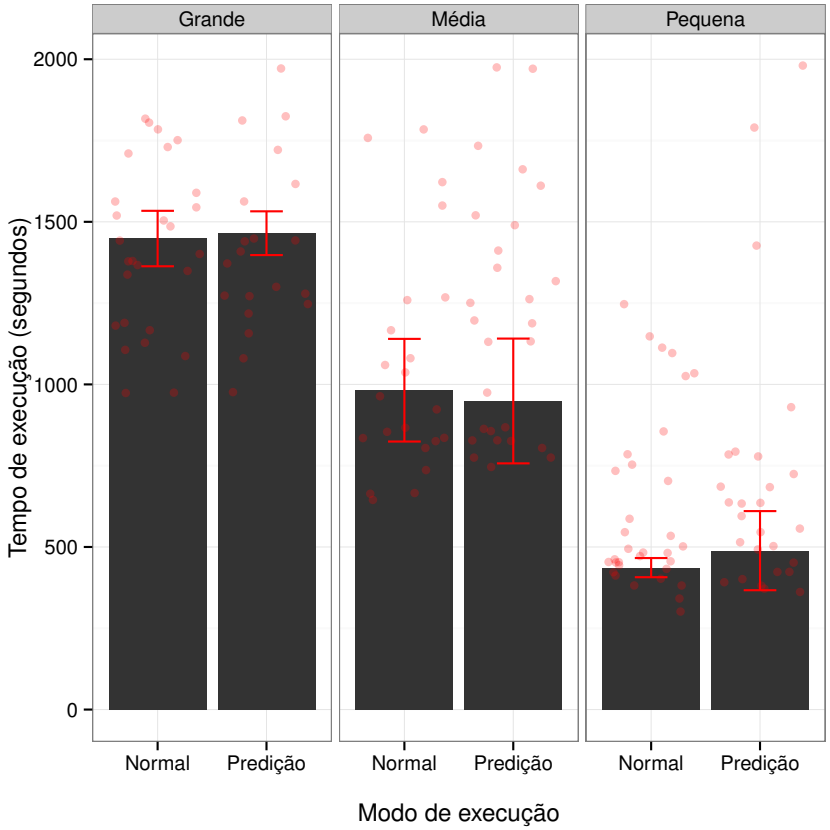


Figura 4.15: Tempo médio de execução da tarefa *Pi Estimator*, considerando os três níveis do tamanho da entrada, em *clusters* de 7 nós *workers*. Não há diferença estatística no tempo de execução das tarefas comparando o cenário com e sem o uso de predição.

4.4.2 Execução com diversos tipos de tarefa

Como já mencionado, nosso ambiente de testes tinha uma limitação de rede, o que faz a saturação de ganho de desempenho conforme aumenta o tamanho do *cluster* ocorrer mesmo com poucos nós, especialmente em tarefas que requisitam mais comunicação entre os recursos. Esse efeito foi mostrado anteriormente nas Figuras 4.8 e 4.9, e é reforçado aqui na Figura 4.16, que mostra os tempos de execução do *TeraSort* e *Word Count*, considerando o tempo de criação do *cluster*. Na maior parte dos casos, não é vantajoso para o usuário escalar o *cluster*, já que esse cenário acarreta o pagamento de mais recursos por mais tempo.

Neste caso, os ganhos foram ligeiramente menores que os observados no caso anterior apenas com Pi. Como mostra a Figura 4.17, a taxa de tarefas processadas foi levemente

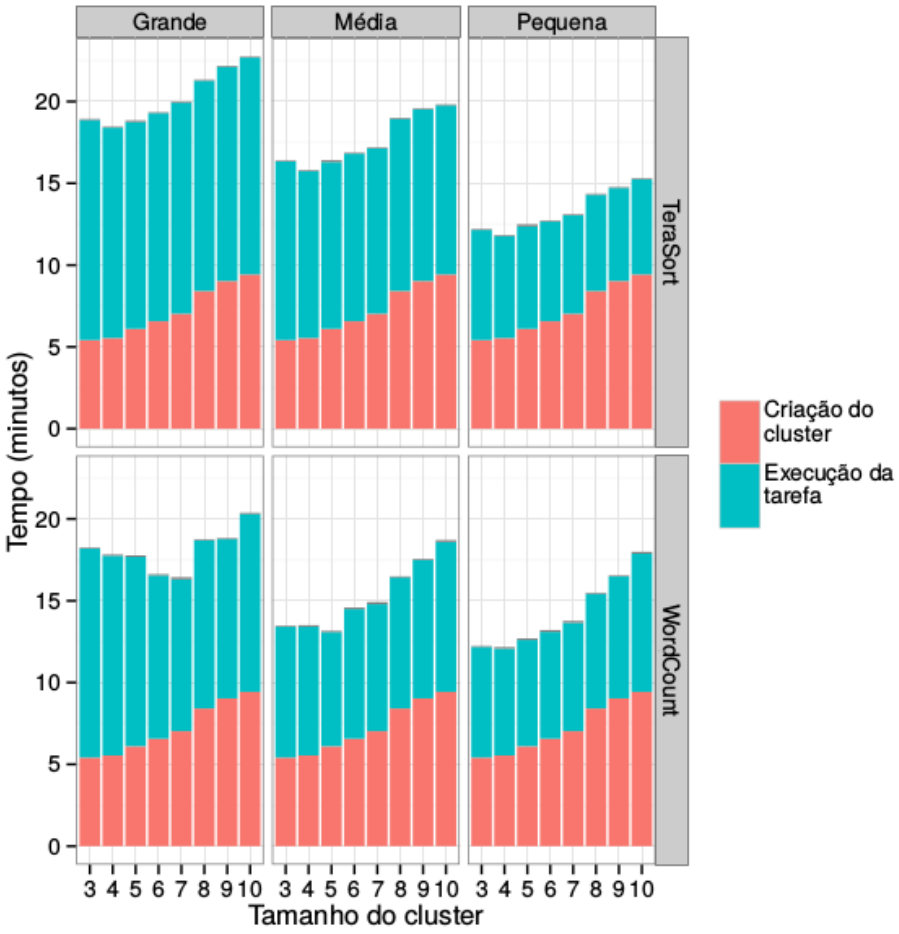


Figura 4.16: Tempo médio de execução das tarefas *TeraSort* e *Wordcount*, considerando o tempo de criação do *cluster*.

maior no caso com predição, variando de 81% para 87%, em média.

Tabela 4.3: Lucro financeiro médio obtido nos experimentos considerando todos os tipos de tarefas.

Ambiente	Tarefas processadas	Valor bruto (R$)	Custo operacional mensal (R$)	Lucro mensal (R$)
Normal	130	481	80	401
Com predição	137	506	80	426

Enquanto isso, a utilização média dos recursos aumentou 2% (40 para 42), no caso da

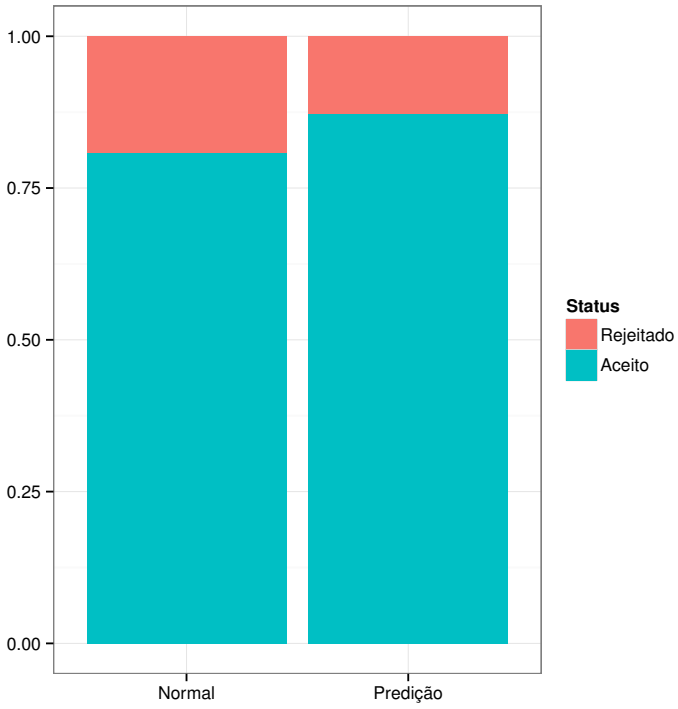


Figura 4.17: Percentual de tarefas processadas com sucesso comparando casos com e sem predição, executando os 3 tipos de tarefas. Acréscimo de 3% no caso com predição.

CPU, e 8%, no caso da memória RAM (67 para 75). No caso da utilização de CPU, não houve diferença significante estatisticamente, como suporta a Figura 4.18. A Figura 4.19 ilustra esse cenário ao longo do tempo. Finalmente, os valores de faturamento líquido e bruto deste cenário são mostrados na Tabela 4.3. Os cálculos foram feitos considerando os mesmos parâmetros utilizados na seção anterior. O acréscimo no lucro médio aqui foi de 6%, o que é justificado pelo também pequeno acréscimo na taxa de tarefas processadas.

4.5 Conclusão dos resultados

Dessa forma, temos evidências de que fazer uso da solução aqui proposta nesses cenários consegue diminuir os custos de provisionamento para o cliente ajudando-o a tomar decisões não-triviais, ao mesmo tempo que aumenta o lucro do provedor de nuvem, por meio de um melhor uso dos recursos que torna possível atender um número maior de requisições de

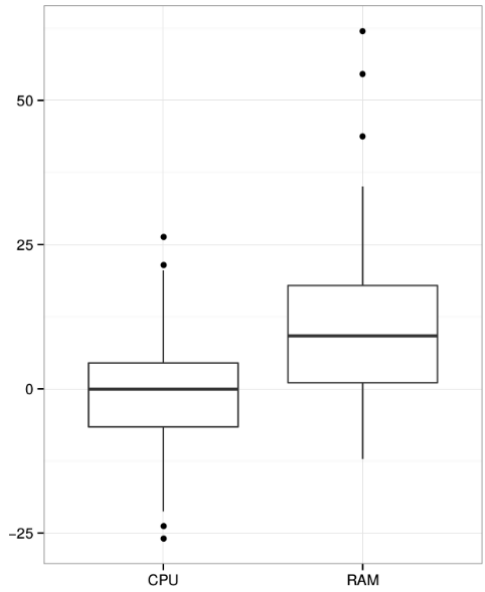


Figura 4.18: Box plot da diferença da utilização média de CPU e RAM ao longo do tempo com e sem predição. Em ambas as métricas, a mediana encontra-se acima de 0, sendo uma diferença pequena no caso da CPU. Apenas no caso da memória RAM há diferença estatística significativa.

clientes. Mesmo no pior caso que obtivemos, que foi o cenário com limitações de vazão de rede, ainda foi observado um lucro de cerca de 3% com aumento na utilização de recursos de 2%, no caso da CPU, e 8% no caso da RAM.

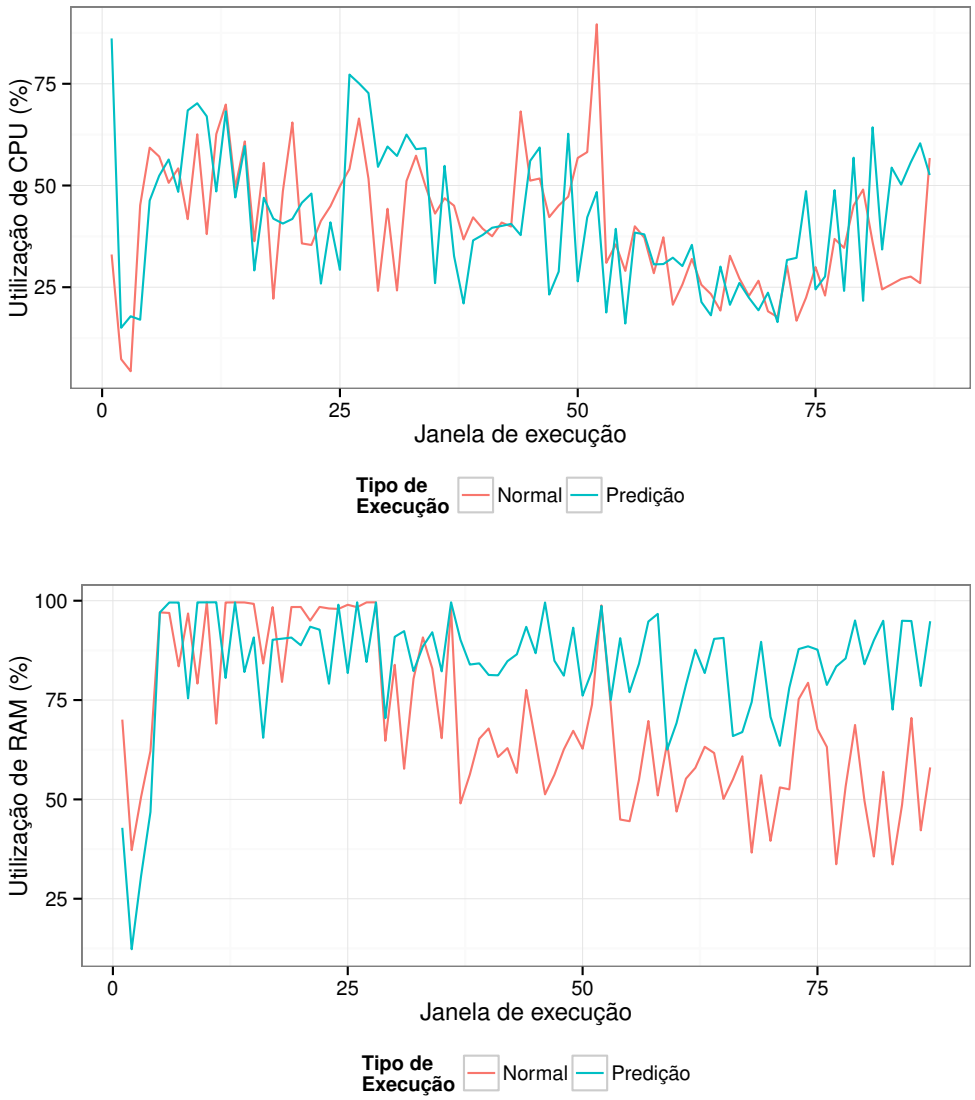


Figura 4.19: Utilização média de CPU e RAM nos dois servidores ao longo do tempo. As execuções com predição apresentam um aumento de 2% na utilização média de CPU e 10% na utilização de RAM.

Capítulo 5

Conclusão

Esse capítulo apresenta, na Seção 5.1, uma sumarização do que foi proposto e realizado neste trabalho, bem como algumas considerações sobre os resultados obtidos na instanciação da solução proposta. Por fim, a Seção 5.2 descreve potenciais limitações apresentadas por este trabalho, bem como possíveis trabalhos futuros.

5.1 Sumário

Neste trabalho de dissertação foi proposto uma abordagem preditiva para gerenciamento de recursos em nuvens de PaaS voltadas para processamento de dados. Esta abordagem serve como uma alternativa ao sistema tradicional de cotas estáticas de recursos adotado na maior parte dos sistemas de computação na nuvem. A solução aqui proposta faz uso de técnicas preditivas, tanto a nível operacional, prevendo com base em dados históricos a utilização dos servidores da nuvem a curto prazo, como também a nível de aplicação de forma não-intrusiva, prevendo por meio de características da aplicação Hadoop, como parâmetros de entrada e quantidade de recursos utilizada, o tempo que essa tarefa levaria para ser executada. Com base nessas predições, esse sistema toma decisões não triviais, como adiar uma tarefa devido à falta de recursos no momento, caso haja previsão de ter recursos disponíveis a curto prazo, ou acelerar a tarefa, caso haja mais recursos disponíveis do que requisitados no momento, e previsão de que continue havendo no curto prazo.

Este trabalho foi motivado pelo problema de otimização de uso de recursos em ambientes computacionais, observado em investigações do estado da arte de gerenciamento destes

mesmos recursos. A partir desse ponto, validamos, por meio de experimentos realizados em nosso ambiente, como esse desperdício de recursos afetava o custo de operação de nuvens computacionais, por meio de um modelo de estimativa do consumo de energia dos servidores do ambiente de testes. Com isso, constatamos que o custo de manter servidores ociosos é proporcionalmente muito alto, e uma forma de otimizar esse custo seria mantendo-os em um nível mais alto de utilização, e, consequentemente, atendendo demandas de mais clientes.

Para isso, usamos técnicas preditivas simples, como média móvel, para prever a carga da nuvem com base em dados recentes de utilização, e um algoritmo de agrupamento, chamado *KNN*, para prever o tempo de duração de tarefas Hadoop de acordo com as tarefas mais similares executadas anteriormente. Mostramos, para cada um dos casos, como prova de conceito, que a solução aqui proposta funciona bem mesmo fazendo uso de preditores relativamente simples, que podem ser substituídos por técnicas mais robustas em trabalhos futuros. Em seguida, montamos uma base de dados com diversos tipos de tarefas Hadoop, tamanhos de *cluster* e entrada, com intuito de usar no KNN, e instanciamos a solução proposta em nosso ambiente, para realizamos experimentos a fim de validá-la.

Nestes experimentos, geramos uma carga sintética de CPU e RAM nos servidores, com base em dados de *clusters* de produção da Google, a fim de simular um ambiente real. Então, executamos diversos experimentos submetendo tarefas de tipo, entrada e tamanho de *cluster* aleatórios, em determinados momentos de tempo, comparando o caso normal, usando cotas de recursos, e o caso utilizando a solução proposta. Executadas as tarefas, coletamos várias métricas ao longo de cada execução, como utilização total de CPU e RAM nos servidores que compõem a nuvem, tempo de execução das tarefas, ações tomadas (no caso com predição), e quantidade de tarefas que tiveram seu processamento rejeitado por falta de recursos. A partir dessa utilização de CPU, estimamos, por meio do modelo de energia montado, o custo operacional do ambiente, de acordo com a utilização.

Com os resultados desses experimentos, observamos aumento de lucro que varia entre 10 e 20%, quando usamos os valores de energia atuais em conjunto com o modelo de tarifação de clientes de um provedor de nuvem bem estabelecido no mercado. Além disso, a média de utilização dos recursos da nuvem foi aumentada em mais de 10%, quando comparando com o caso sem predição de recursos. Isso tudo acontece ao mesmo tempo que o cliente também consegue obter um custo de provisionamento menor, tendo como base informações não-

triviais fornecidas pelo preditor de carga, que mostra que nem sempre é vantajoso aumentar o tamanho do *cluster* esperando resultados mais rápidos.

Esses resultados enfatizam o quão importante é um gerenciamento de recursos adequado em ambientes de nuvem computacionais. Um bom gerenciamento pode melhorar tanto o lucro do provedor, quanto diminuir o custo de provisionamento para os clientes, e atender mais clientes, satisfazendo casos tanto de nuvens públicas como privadas.

A Seção 5.2 discute algumas limitações deste trabalho e como mitigá-las em trabalhos futuros.

5.2 Limitações e Trabalhos Futuros

A solução apresentada neste trabalho apresenta algumas limitações que são decorrentes tanto de simplificações realizadas durante sua construção, como de deficiências na estrutura de *hardware* do ambiente de experimento montado. As limitações identificadas nesse processo foram:

- A solução proposta tem o escopo reduzido ao suporte a tarefas implementadas para o Hadoop, que, embora seja uma das mais comuns ferramentas para o processamento de dados em lote, também há outros arcabouços comuns como Hydra, *Cluster Map Reduce*, e Spark. A previsão de tempo e carga de cada uma dessas ferramentas é diferente do que aconteceria no caso do Hadoop, por ter arquitetura e parâmetros de configuração diferentes que podem impactar o desempenho da execução. Além disso, há também arcabouços voltados para o processamento contínuo de dados, como Storm, Stream Mine 3g e novamente o Spark, que como processam os dados em tempo real, à medida que chegam, requerem análises de desempenho diferentes do processamento em lote;

- A medição de custo operacional aqui realizada é bastante simplista, uma vez que consideramos apenas a potência dos servidores de computação, como sendo uma função do uso de CPU, desconsiderando outros fatores que, são responsáveis por uma fração menor do consumo, tais como uso de RAM, rede e disco dos servidores. Além disso, para medir com precisão o custo total, o consumo oriundo de outros elementos deve ser

considerado, como *switches*, *enclosures*, *clusters* de armazenamento (Ceph, no nosso caso), além dos custos com refrigeração;

- A base de dados utilizada pelo KNN para predição de tempo de tarefas Hadoop foi montada antecipadamente. Em um ambiente real, o ideal seria que essa base fosse montada de forma *online*, e sendo alimentada e retreinada a cada vez que uma nova tarefa fosse processada;

- A análise e predição dos recursos da nuvem, bem como as tomadas de decisão do escalonador são feitas apenas no momento que a tarefa é recebida para processamento. Em cenários onde a decisão é um adiamento, fazendo a tarefa esperar por N janelas para ser executada, a avaliação poderia ter resultados diferentes caso fosse repetida ao longo dessas janelas de espera, já que os recursos necessários poderiam eventualmente estar disponíveis antes do tempo esperado;

Possíveis trabalhos futuros envolvem, além de solucionar os pontos acima mencionados, avaliar a criação de *clusters* arbitrariamente grandes para avaliar melhor o retorno de desempenho em relação ao tamanho do cluster. Além disso, uma perspectiva que pode ser adotada é a análise do custo-beneficio de execução das tarefas, onde, por exemplo, haja um cenário que executar com mais nós torna a execução mais rápida, enquanto uma leve diminuição torna a tarefa mais barata. Essa abordagem permite que o usuário analise o *trade-off* entre tempo e custo de execução, de acordo com suas necessidades.

Bibliografia

[1] Amazon EC2 Spot Instances. http://aws.amazon.com/ec2/purchasing-options/spot-instances/, feb 2015.

[2] Amazon EMR. http://aws.amazon.com/elasticmapreduce/, feb 2015.

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

[4] Google cluster-usage Traces. https://github.com/google/cluster-data, 2015.

[5] OpenStack Cloud Computing Software. http://www.openstack.org/, dec 2015.

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

[7] Apache Spark - Lightning-fast cluster computing. http://spark.apache.org/, jun 2016.

[8] Calculadora de tarifas do AWS. http://calculator.s3.amazonaws.com/index.html, jul 2016.

[9] Hadoop examples. https://svn.apache.org/repos/asf/hadoop/common/branches/branch-1/src/examples/org/apache/hadoop/examples/, 2016.

[10] HPE OneView. https://www.hpe.com/br/pt/integrated-systems/software.html, 2016.

[11] Lambda Architecture. http://lambda-architecture.net/, may 2016.

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

[13] Openstack scheduler configuration guide. http://docs.openstack.org/kilo/config-reference/content/section_compute-scheduler.html, jul 2016.

[14] OpenStack User Survey. https://www.openstack.org/user-survey/survey-2016-q1/landing, apr 2016.

[15] The R Project for Statistical Computing. https://www.r-project.org/, jun 2016.

[16] Sysstat - Performance monitoring tools for Linux. http://sebastien.godard.pagesperso-orange.fr/, 2016.

[17] Tipos de tarifas - Energisa. http://www.energisa.com.br/Paginas/informacoes/taxas-prazos-e-normas/tipos-tarifas.aspx, jul 2016.

[18] Welcome to python.org. https://www.python.org/, jun 2016.

[19] A. Fox R. Katz A. Ganapathi, Y. Chen and D. Patterson. Statistics-driven workload modeling for the cloud. In *Data Engineering Workshops (ICDEW), 2010 IEEE 26th International Conference on*, pages 87 – 92, Long Beach, CA, USA, 2010. IEEE.

[20] S. Akaho. A kernel method for canonical correlation analysis. *Learning*, (4):1–7, 2006.

[21] L. A. Barroso and U. Hölzle. The case for energy-proportional computing. *Computer*, 40(12):33–37, Dec 2007.

[22] A. Beloglazov and R. Buyya. Energy Efficient Resource Management in Virtualized Cloud Data Centers. *2010 10th IEEE/ACM International Conference on Cluster, Cloud and Grid Computing*, pages 826–831, 2010.

[23] I. Carrera and C. Geyer. Modeling the performance of mapreduce applications for the cloud. 2015.

[24] M. Carvalho, F. Brasileiro, and J. Wilkes. Long-term SLOs for reclaimed cloud computing resources.

[25] J. Chase, D. Anderson, P. Thakar, A. Vahdat, and R. Doyle. Managing energy and server resources in hosting centers. In *Proceedings of the Eighteenth ACM Symposium on Operating Systems Principles*, SOSP '01, pages 103–116, New York, NY, USA, 2001. ACM.

[26] N. Chohan, C. Castillo, M. Spreitzer, Ma. Steinder, A. Tantawi, C. Krintz, and S. Barbara. See Spot Run: Using Spot Instances for MapReduce Workflows. In *2nd USENIX Workshop on Hot Topics in Cloud Computing, HotCloud*, 2010.

[27] R CRAN. Fnn package. https://cran.r-project.org/web/packages/FNN/FNN.pdf, 2014.

[28] S. Daneshyar and M. Razmjoo. Large-scale data processing using mapreduce in cloud computing environment. 2012.

[29] J. Dean and S. Ghemawat. Mapreduce: Simplified data processing on large clusters. *Commun. ACM*, 51(1):107–113, January 2008.

[30] F. Farahnakian, A. Ashraf, and T. Pahikkala. Using Ant Colony System to Consolidate VMs for Green Cloud Computing. *IEEE Transactions on Services Computing*, 8(2):187–198, 2015.

[31] R. Ghosh and V. 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.

[32] Apache Hadoop. Partitioning your job into maps and reduces. http://wiki.apache.org/hadoop/HowManyMapsAndReduces, 2014.

[33] J. Hamilton. Cooperative expendable micro-slice servers (cems): Low cost, low power servers for internet-scale services.

[34] P. Houle. Choosing the number of reducers. https://github.com/paulhoule/infovore/wiki/Choosing-the-number-of-reducers, 2013.

[35] C. Hsu, K. Slagter, S. Chen, and Y. Chung. Optimizing energy consumption with task consolidation in clouds. *Information Sciences*, 258:452–462, 2014.

[36] D. Huang, D. Yang, and H. Zhang. Energy-aware virtual machine placement in data centers. *2012 IEEE Global Communications Conference (GLOBECOM)*, pages 3243–3249, December 2012.

[37] C. Geyer I. Carrera, F. Scariot and P. Turin. An example for performance prediction for map reduce applications in cloud environments. Master's thesis, UFGRS, 2013.

[38] O. de Carvalho Junior, S. Bruschi, R. Santana, and M. Santana. GreenMACC : Uma Arquitetura para Metaescalonamento Verde com QoS em Nuvens Privadas.

[39] L. Ponciano, A. Brito, L. Sampaio, and F. Brasileiro. Energy efficient computing through productivity-aware frequency scaling. In *Cloud and Green Computing (CGC), 2012 Second International Conference on*, pages 191–198, Nov 2012.

[40] C. Reiss, G. Ganger, R. Katz, M. Kozuch, and A. Tumanov. Heterogeneity and Dynamicity of Clouds at Scale: Google Trace Analysis. In *IEEE SOC Conference*, page 13, 2012.

[41] N. Roy, A. Dubey, and A. 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.

[42] Z. Shen, S. Subbiah, X. Gu, and J. Wilkes. CloudScale: elastic resource scaling for multi-tenant cloud systems. *Proceedings of the 2nd Symposium on Cloud Computing*, pages 5:1—-5:14, 2011.

[43] D. Sun, G. Zhang, S. Yang, W. Zheng, S. Khan, and K. Li. Re-Stream : Real-time and energy-efficient resource scheduling in big data stream computing environments. *INFORMATION SCIENCES*, 2015.

[44] F. Tian and K. Chen. Towards optimal resource provisioning for running mapreduce programs in public clouds. In *Cloud Computing (CLOUD), 2011 IEEE International Conference on*, pages 155 – 162, Washington, DC, USA, 2010. IEEE.

[45] M. Villari, A. Celesti, M. Fazio, and A. Puliafito. Alljoyn lambda: An architecture for the management of smart environments in iot. In *Smart Computing Workshops (SMARTCOMP Workshops), 2014 International Conference on*, pages 9–14, Nov 2014.

[46] W. Voorsluys and R. 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.

[47] R. Weingärtner, B. Bräscher, and C. Westphall. Cloud resource management: A survey on forecasting and profiling models. *Journal of Network and Computer Applications*, 47:99–106, January 2015.

[48] G. Wu, M. Tang, Y. Tian, and W. Li. Energy-efficient Virtual Machine Placement in Data Centers by Genetic Algorithm. 2012.

[49] Z. Gong e J. Wilkes X. Gu. PRESS: PRedictive Elastic reSource Scaling for cloud systems. *Proceedings of the 2010 International Conference on Network and Service Management, CNSM 2010*, pages 9–16, 2010.

Apêndice A

Registros de máquinas da Google utilizados nos experimentos

Mostramos nesse apêndice os registros de utilização de máquinas do *cluster* do Google usados em nosso ambiente de testes, a fim de criar um ambiente com carga similar a uma nuvem real.

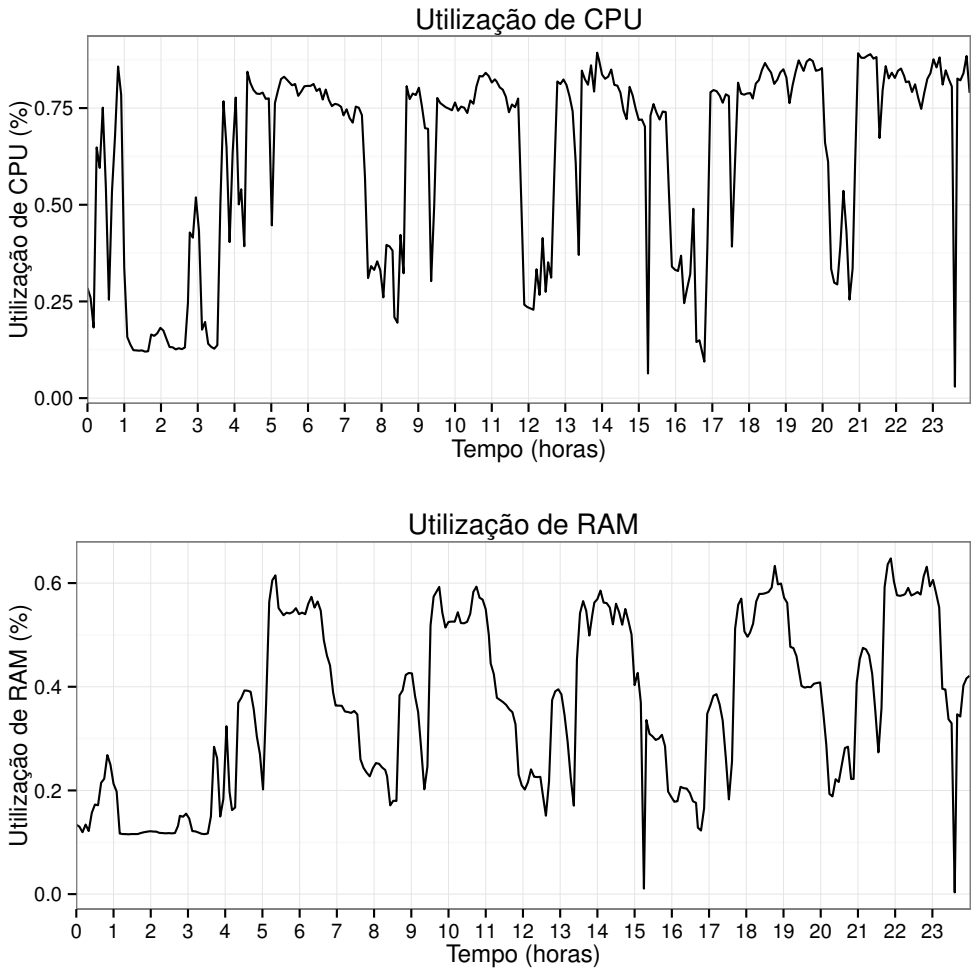


Figura A.1: Dados de utilização de CPU e RAM da máquina do *cluster* do Google com ID 317489255.

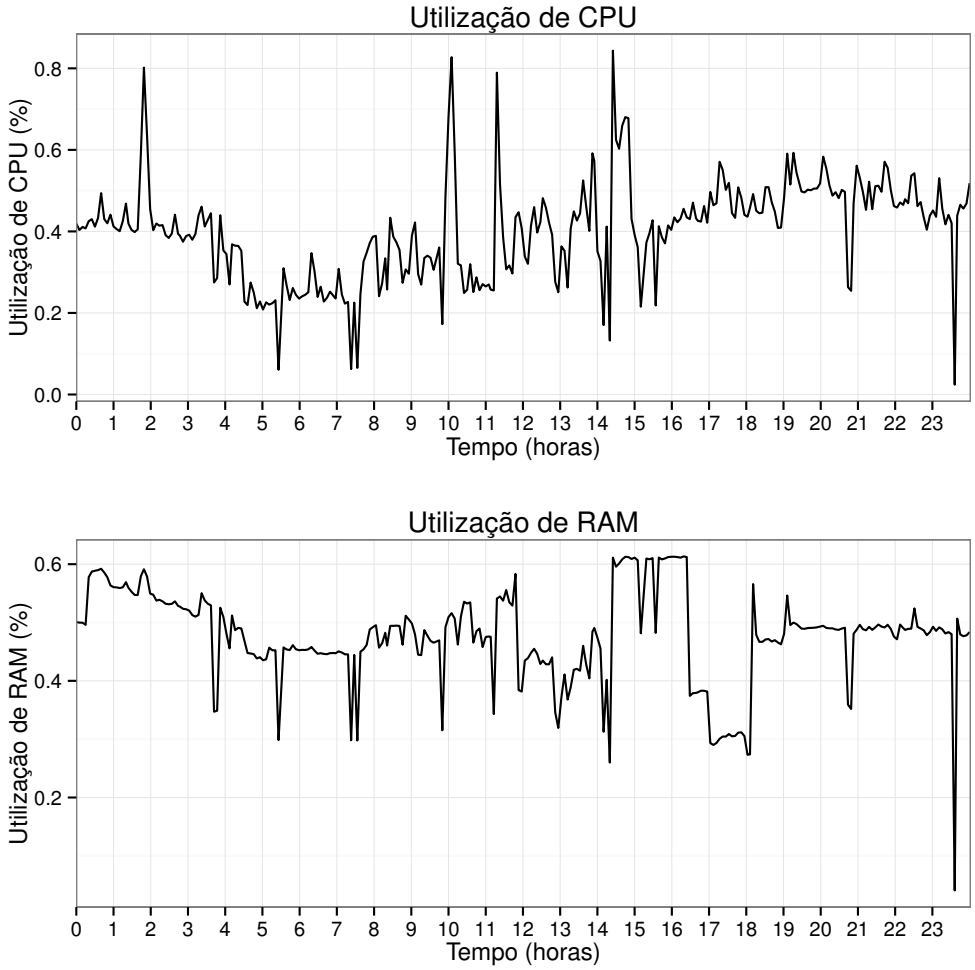


Figura A.2: Dados de utilização de CPU e RAM da máquina do *cluster* do Google com ID 121306.

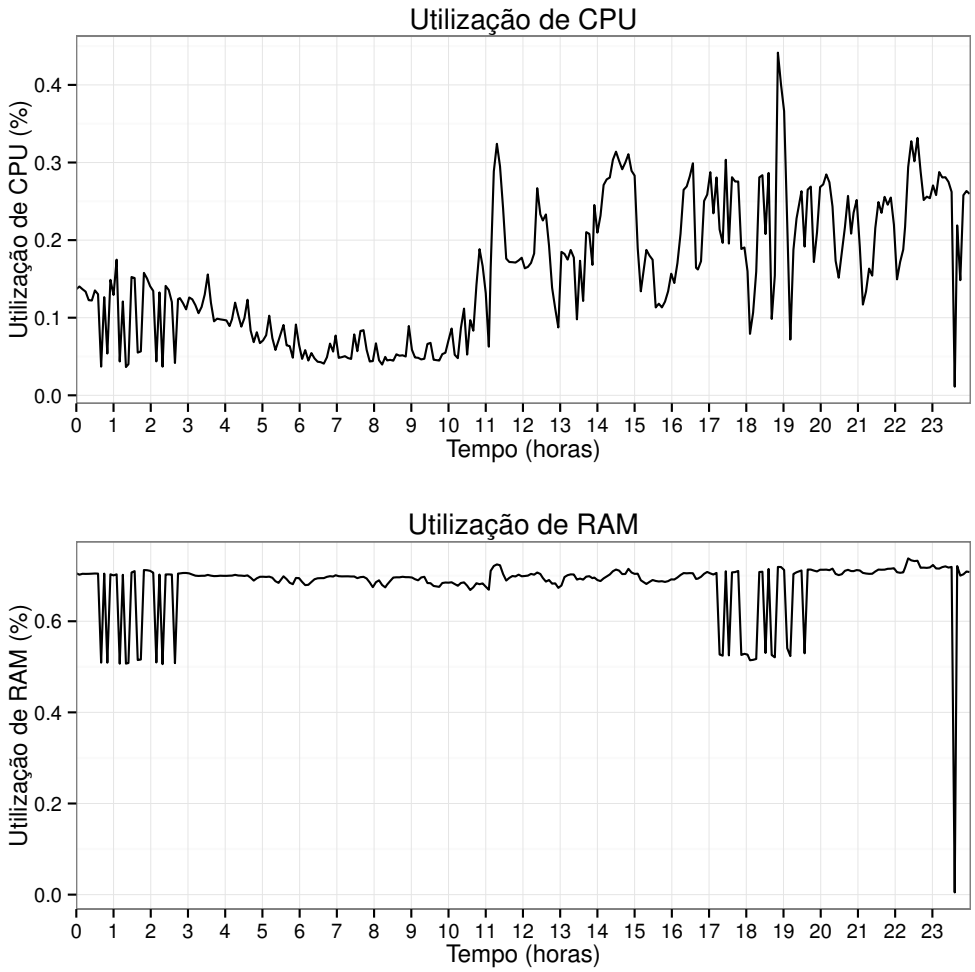


Figura A.3: Dados de utilização de CPU e RAM da máquina do *cluster* do Google com ID 277433540.

Apêndice B

Resultados do preditor de utilização de carga

Mostramos nesse apêndice resultados da predição de carga da nuvem em máquinas aleatórias do *cluster* da Google, executando a predição na forma de janelas deslizantes ao longo de todo o tempo coletado pelos registros de utilização.

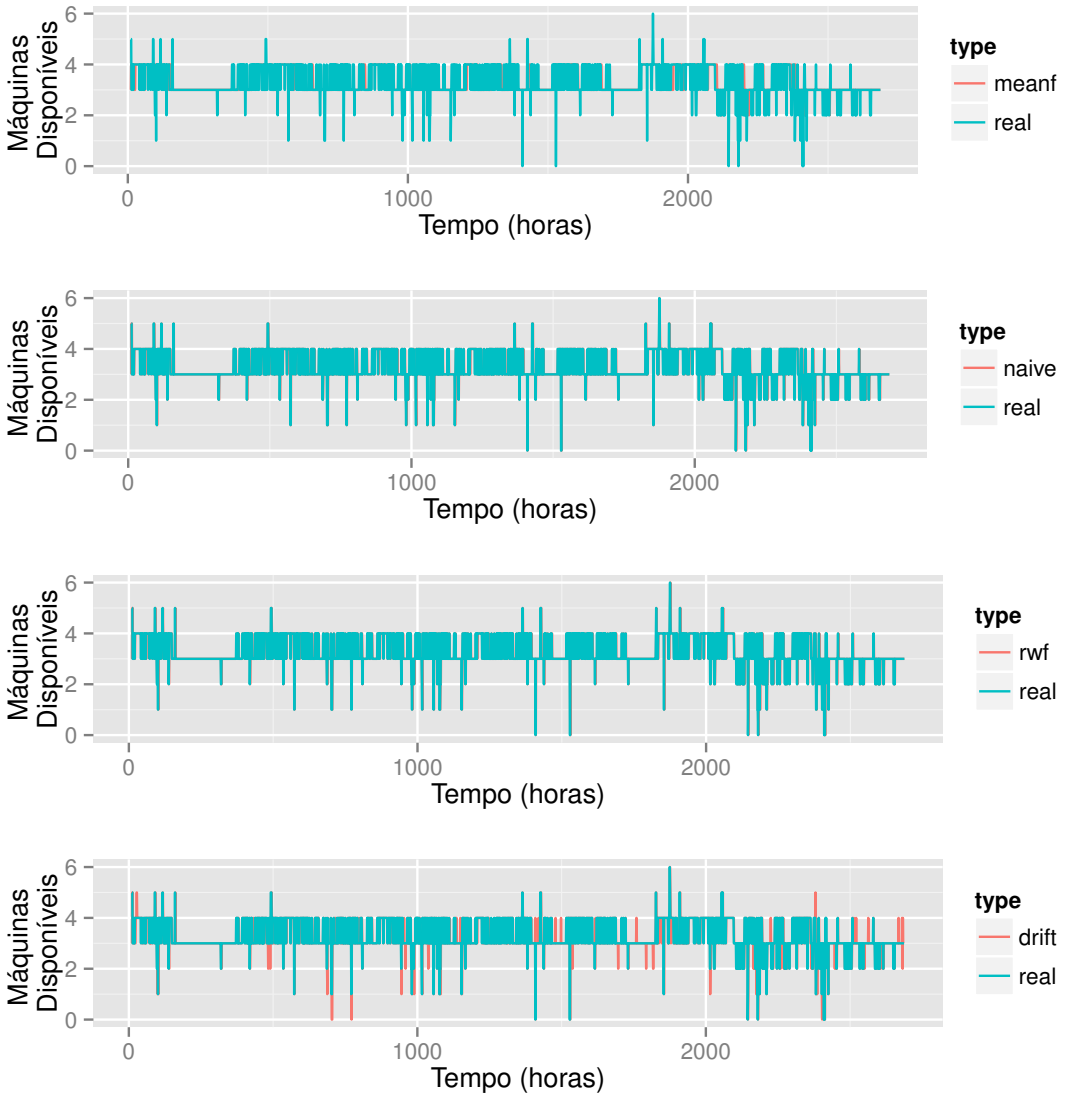


Figura B.1: Comparação entre o número de VMs estimado por cada um dos preditores com o número real disponível. Máquina do *cluster* do Google com ID 1331690.

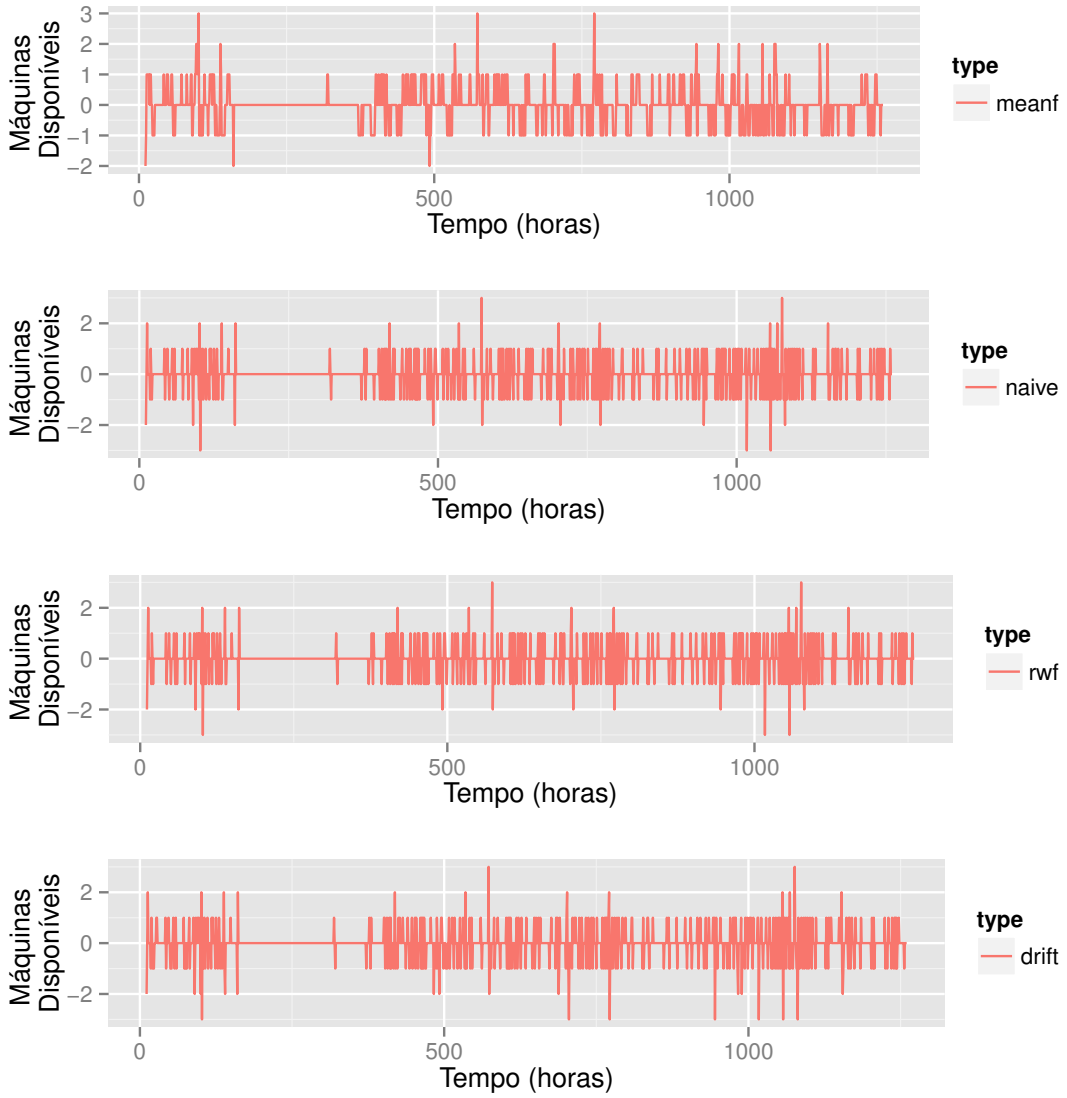


Figura B.2: Erro de previsão do número de máquinas, de acordo com cada preditor. O eixo Y representa a diferença entre *número real disponível − número previsto*. Máquina do *cluster* do Google com ID 1331690.

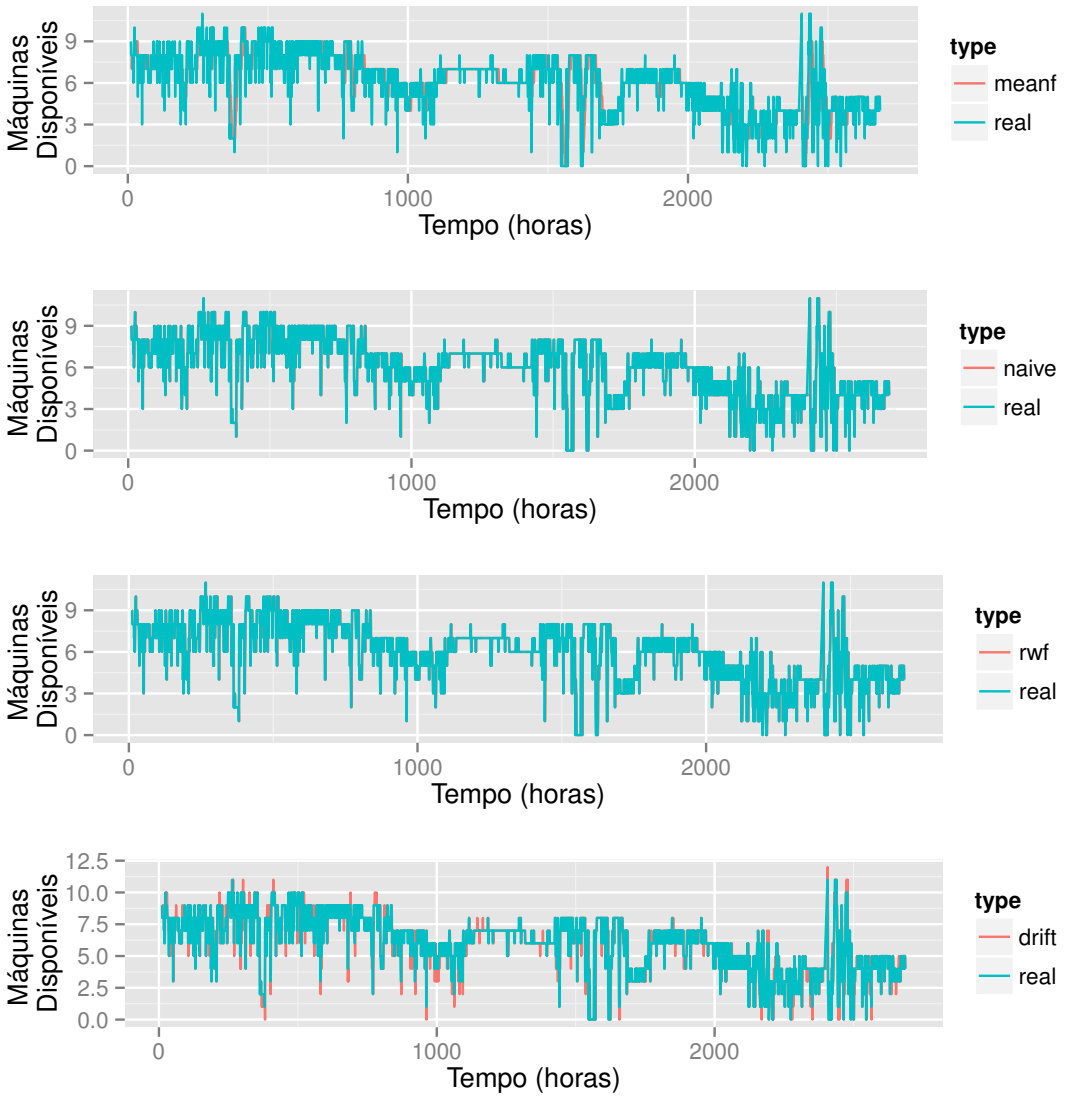


Figura B.3: Comparação entre o número de VMs estimado por cada um dos preditores com o número real disponível. Máquina do *cluster* do Google com ID 4302737021.

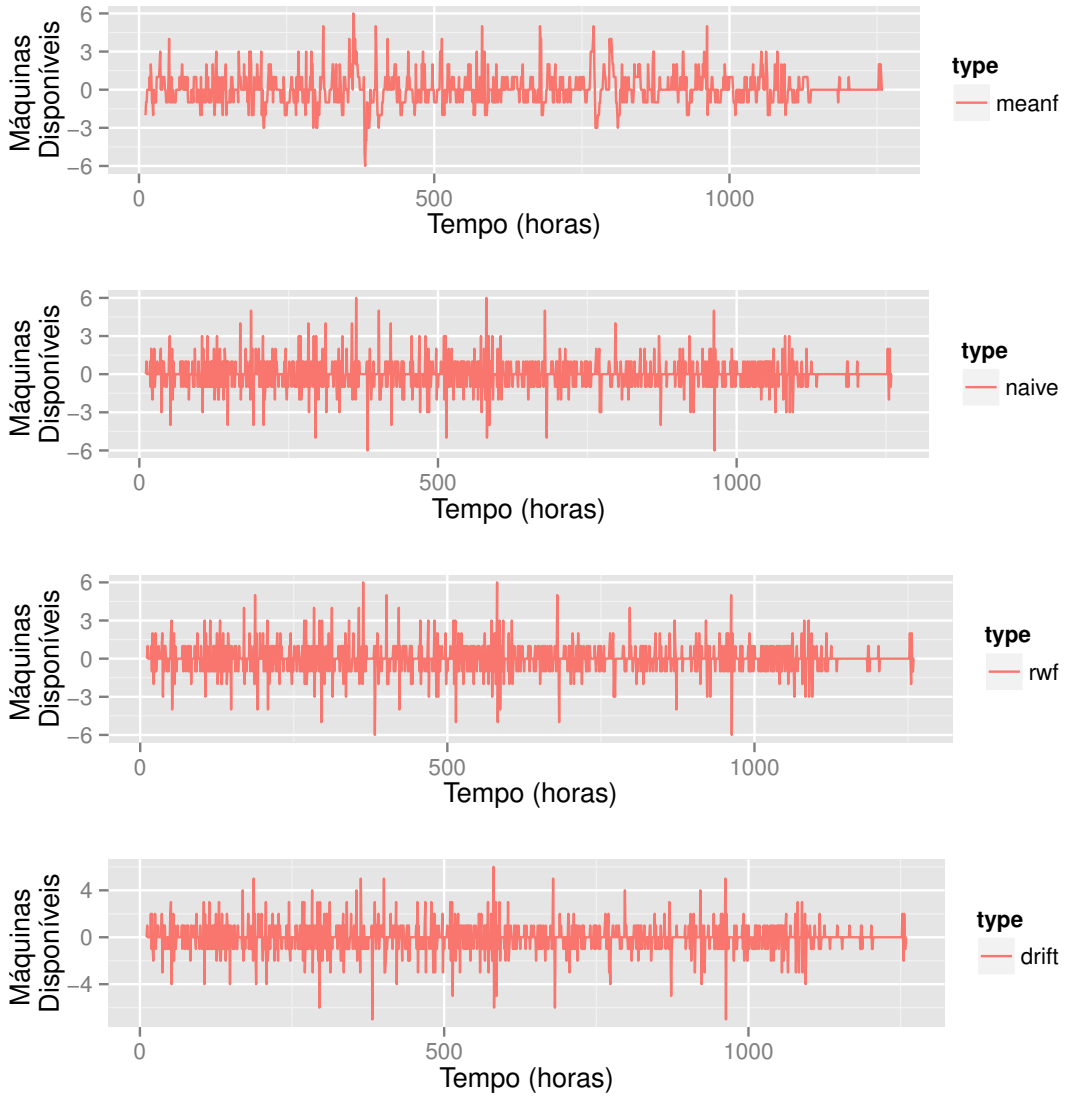


Figura B.4: Erro de previsão do número de máquinas, de acordo com cada preditor. O eixo Y representa a diferença entre *número real disponível − número previsto*. Máquina do *cluster* do Google com ID 4302737021.

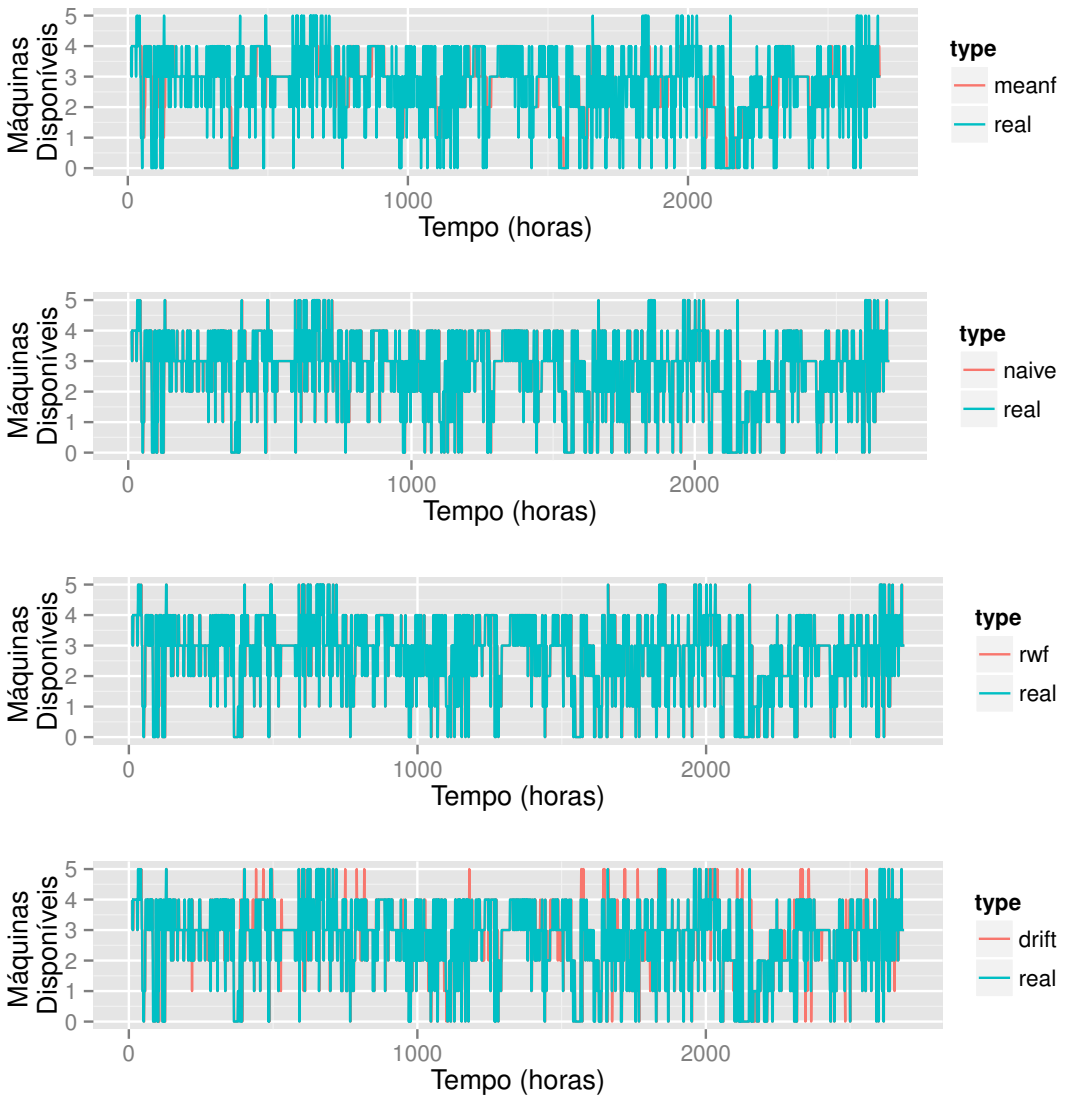


Figura B.5: Comparação entre o número de VMs estimado por cada um dos preditores com o número real disponível. Máquina do *cluster* do Google com ID 1390814016.

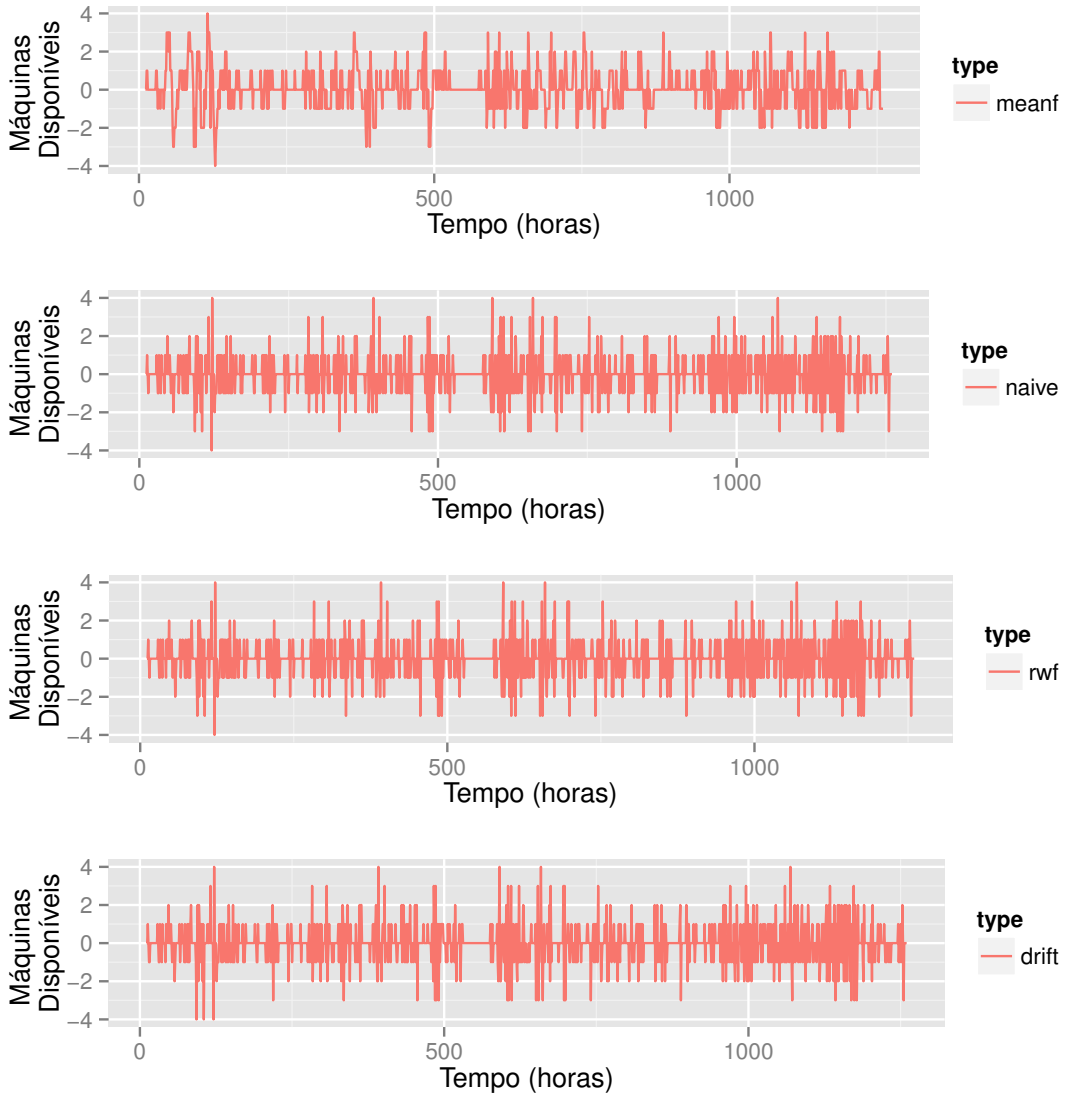


Figura B.6: Erro de previsão do número de máquinas, de acordo com cada preditor. O eixo Y representa a diferença entre *número real disponível − número previsto*. Máquina do *cluster* do Google com ID 1390814016.

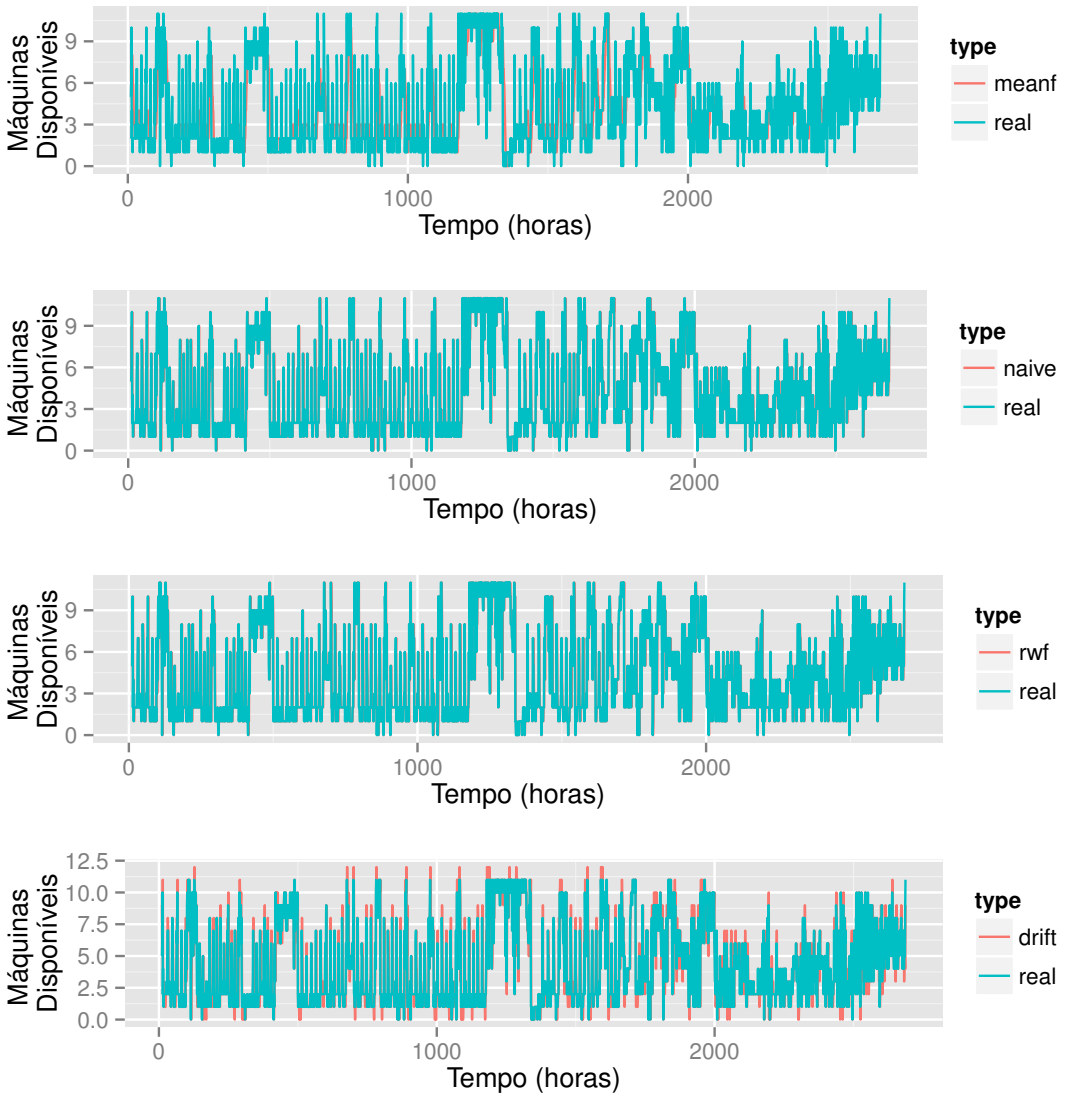


Figura B.7: Comparação entre o número de VMs estimado por cada um dos preditores com o número real disponível. Máquina do *cluster* do Google com ID 317489255.

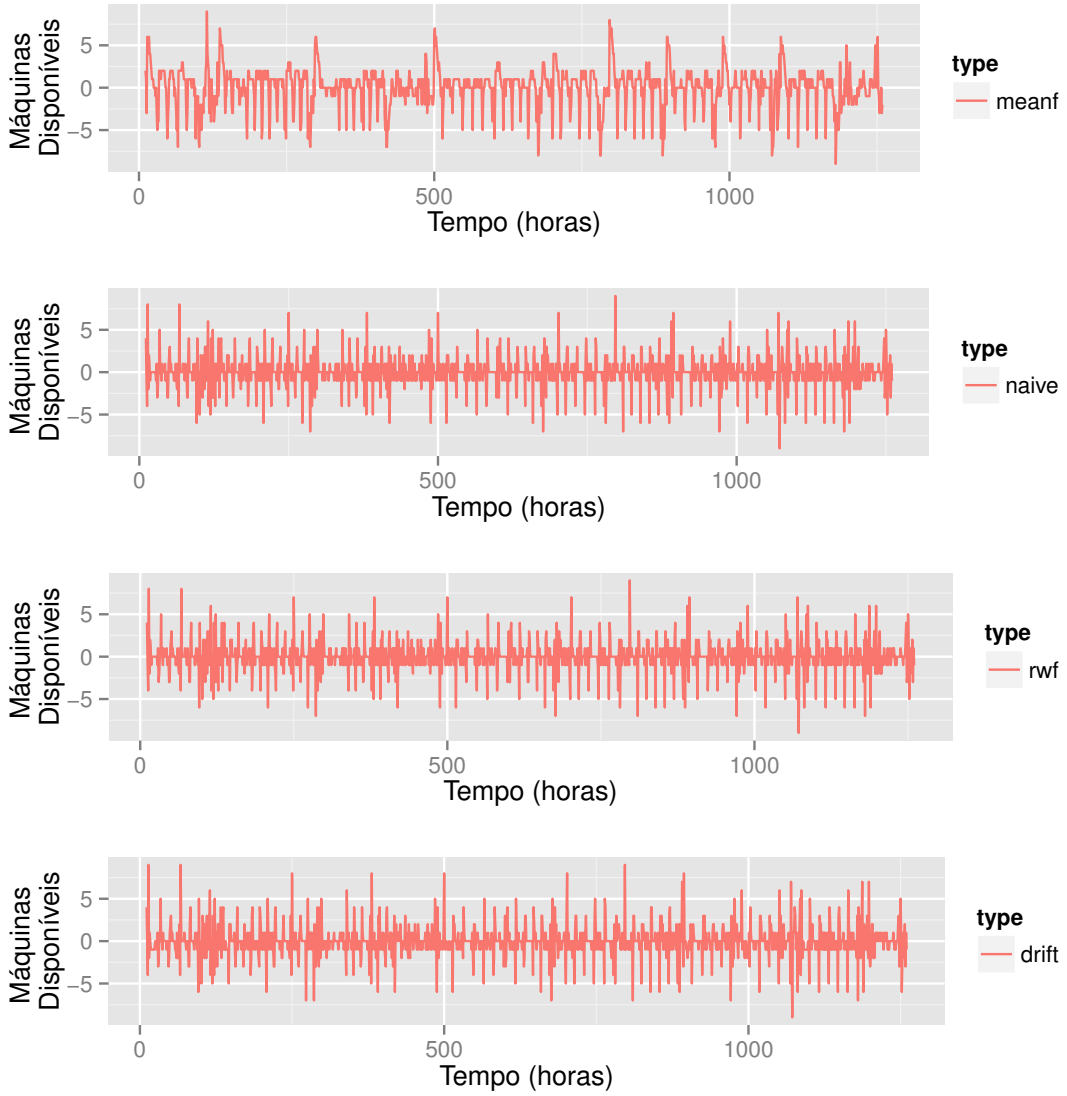


Figura B.8: Erro de previsão do número de máquinas, de acordo com cada preditor. O eixo Y representa a diferença entre *número real disponível − número previsto*. Máquina do *cluster* do Google com ID 317489255.