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

# Heurísticas para balanceamento de carga de máquinas em infraestruturas de nuvem

Iury Gregory Melo Ferreira

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)
Lívia Maria Rodrigues Sampaio Campos (Orientadora)

Campina Grande, Paraíba, Brasil

**FICHA CATALOGRÁFICA ELABORADA PELA BIBLIOTECA CENTRAL DA UFCG**

F383h Ferreira, Iury Gregory Melo.

Heurísticas para balanceamento de carga de máquinas em infraestruturas de nuvem / Iury Gregory Melo Ferreira. – Campina Grande, 2018.

80 f. : il. color.

Dissertação (Mestrado em Ciência da Computação) – Universidade Federal de Campina Grande, Centro de Engenharia Elétrica e Informática, 2017.

"Orientação: Prof. Dr. Andrey Elísio Monteiro Brito, Profa Dra. Lívia Maria Rorigues Sampaio Campos".

Referências.

1. Computação na Nuvem. 2. Balanceamento de Carga. 3. Migração - Computação. 4. Gerenciamento de Recursos. 5. Provisionamento Vertical. 6. OpenStack. I. Brito, Andrey Elísio Monteiro. II. Campos, Lívia Maria Rorigues Sampaio. III. Título.

CDU 004.65(043)

# Resumo

Em ambientes de Computação na Nuvem, principalmente os que utilizam o modelo de infraestrutura como um serviço, a característica de elasticidade no provisionamento de recursos traz consigo a necessidade de gerenciar os recursos físicos de forma apropriada para preservar a qualidade de serviço aos seus usuários, e o bom desempenho da infraestrutura. Este trabalho propõe heurísticas que são capazes de auxiliar no balanceamento de carga dos servidores em uma infraestrutura de nuvem, propondo migrações para diminuir a sobrecarga nos servidores que foram identificados como sobrecarregados, visto que, com o passar do tempo há uma variação natural na quantidade de recursos em uso. Esta variação é uma consequência da remoção ou adição de aplicações, ou até mesmo de tentativas de melhoramento do desempenho das aplicações através do provisionamento vertical. Uma ferramenta foi implementada para fazer uso dos algoritmos das heurísticas e assim auxiliar nos experimentos para a validação das mesmas. As métricas utilizadas vem diretamente de servidores heterogêneos da nuvem OpenStack do Laboratório de Sistemas Distribuídos. Os resultados obtidos mostram que além da diminuição no consumo de CPU dos servidores dos quais que estavam sobrecarregados, também é possível melhorar o desempenho destes servidores em alguns casos.

**Palavras-chave**: Computação na Nuvem, Balanceamento de Carga, Migração, Gerenciamento de Recursos, Provisionamento Vertical, OpenStack.

# Abstract

In Cloud Computing environments, especially those using the infrastructure as a service model, the elasticity characteristic in resource provisioning comes with the need to manage resources so the quality of service can continue to be guaranteed to users and also to maintain a good performance of the infrastructure. This work proposes heuristics that are able to assist in the load balancing of the servers in a Cloud infrastructure, proposing migrations to reduce the overhead in the servers that were identified as overloaded, since with the passage of time there is a natural variation in the amount of resources in use. This variation in a consequence of removal or addition of applications and even of the usage of vertical scaling to improve application's performance. A tool was implemented to make use of the heuristic algorithms and thus to aid in the experiments and their validation, the metrics used come directly from heterogeneous servers of the OpenStack Cloud of the Distributed Systems Laboratory. The results show that in addition to the decrease in CPU consumption of servers that were overloaded, it is also possible to improve the performance of these servers in some cases.

**Keywords**: Cloud Computing, Load Balancing, Migration, Resource Management, Vertical Scaling, OpenStack.

# Agradecimentos

Agradeço primeiramente a Deus, por tudo que fez na minha vida, pois, sei que nada acontece sem que seja pela sua vontade. Agradeço a minha família, pelo apoio prestado durante estes três anos, em especial, aos meus pais Gedeão e Gorete, minha irmã Ingride, meu cunhado Ronaebson e também a todos os meus tios, tias e primos.

Muito obrigado a Andrey, que me orientou e acolheu aqui no laboratório nestes três anos de mestrado, sou grato pelos conselhos e ensinamentos que obtive além da confiança que me foi depositada para seguir firme e terminar. Agradeço também a Lívia que foi minha co-orientadora e se prontificou em ajudar quando necessário.

Obrigado aos companheiros e amigos de salas e projetos do Laboratório de Sistemas Distribuídos aqui na UFCG: Alessandro, Amanda, Beatriz, Chico, Dalton, Eduardo Falcão, Eduardo J, Ely, Flávio, Fábio Jorge, Fábio Silva, Gabriel, Giovanni, Guilherme, Ícaro, Igor Natanael, Marcos, Markin, Matteus, Pedro Pacheco, Ramon, Ricardo, Roberto, Rodolfo, Samuel e Thiago Paiva.

Obrigado aos amigos de João Pessoa e de Campina Grande, por todos os momentos que passamos e vocês me apoiaram indiretamente ou diretamente: Ada, Adolfo, Barbosa, Bryan, Caio Bo, Caynan, Clark, Coutinho, Danillo, Diogo, Erickson, Fitarelli, Flávio, Gutierre, Igor, Isaque, Juan, Kláudio, Kohako, Ledy, Lucas, Magno, Mari, Paulo Montini, Ray, Ris, Rodolfo, Sergio Montini, Ulisses e Werton.

Todos os aqui citados têm uma participação nesta dissertação, obrigado por tudo pessoal!

*Seize the dreams you had!*

*Protect your beloved friends!*

*You can become stronger*

*Unknown power dwells in your heart when its fire is lit*

*Any wish, it's true*

*Will surely be granted...show me your brave heart*

*(Wada Kouji - Brave Heart)*

# Conteúdo

# Lista de Símbolos

API - *Application Programming Interface*

CLI - *Command Line Interface*

CPU - *Central Processing Unit*

IaaS - *Infrastructure as a Service*

KVM - *Kernel-based Virtual Machine*

NIST - *National Institute of Standards and Technology*

PaaS - *Plataform as a Service*

QoS - *Quality of Service* SaaS - *Software as a Service*

SLA - *Service Level Agreement*

VCPU - *Virtual CPU*

VM - *Virtual Machine*

# Lista de Figuras

# Lista de Tabelas

# Capítulo 1

# Introdução

Neste capítulo serão apresentados os conceitos essenciais à dissertação, como também a contextualização sobre as áreas que englobam a pesquisa e qual problema deseja-se resolver. Também elencamos os objetivos, a relevância deste trabalho e a metodologia a ser utilizada. Ao final descreveremos como está definida a organização desta dissertação.

## 1.1 Contextualização

Nesta seção abordaremos os principais conceitos necessários para uma melhor compreensão deste trabalho.

### 1.1.1 Computação na nuvem

No âmbito acadêmico e comercial uma das definições para Computação na nuvem é a fornecida pelo NIST (*National Institute of Standards and Technology*): "A Computação na nuvem é um modelo para permitir acesso de forma conveniente e sob demanda a um conjunto compartilhado de recursos configuráveis (ex., redes, servidores, armazenamento, aplicações e serviços), que podem ser rapidamente provisionados e liberados com o mínimo esforço de gerenciamento ou interação do provedor de serviço." [1].

Ainda, de acordo com Mell e Grance [1], o modelo de nuvem é composto por cinco características essenciais, três modelos de serviço e quatro modelos de implantação, que serão detalhados a seguir.

### Caraterísticas Essenciais

Conforme citado por Mell e Grance [1], as cinco características essenciais para a nuvem são as seguintes:

- Serviço por demanda: o usuário pode requisitar recursos computacionais unilateralmente, de acordo com a sua necessidade, de forma automática e dispensando a interação humana com os provedores de serviço;
- Amplo acesso à rede: o acesso aos recursos da nuvem são disponibilizados por meio da rede e acessados por mecanismos padrão por meio de plataformas (ex., telefones celulares, tablets, laptops e estações de trabalho);
- Compartilhamento de recursos: os recursos computacionais do provedor de infraestrutura são agrupados para atender aos diversos usuários, utilizando um modelo *multi-tenant*[1], com diferentes recursos físicos e virtuais, atribuídos e reatribuídos, dinamicamente, conforme a demanda do usuário. O usuário geralmente não dispõe do controle ou conhecimento sobre a localização exata dos recursos que lhe foram fornecidos, mas pode ser capaz de especificar a localização em um nível de alta abstração (ex., país, estado, ou *datacenter*). Dentre os recursos oferecidos, podemos especificar o armazenamento, o processamento, a memória e a taxa de transmissão;
- Elasticidade rápida: os recursos necessários aos usuários, podem ser provisionados e liberados rapidamente e, em alguns casos, estes podem ser disponibilizados de forma automática, para um provisionamento rápido, e aumentar ou diminuir a quantidade de recursos de acordo com a demanda. Do ponto de vista dos usuários, os recursos que a nuvem provê são infinitos e podem ser requisitados a qualquer hora;
- Serviço medido: os sistemas em nuvem controlam e otimizam automaticamente o uso de recursos com base em medições que se adequam ao tipo de serviço que é ofertado (ex., armazenamento, processamento, taxa de transmissão e contas de usuários ativos). O uso de recursos pode ser monitorado, controlado e reportado, proporcionando transparência tanto para o provedor, como para o consumidor do serviço utilizado.

[1] Vários clientes(*tenant*) compartilham a infraestrutura sem estar ciente e sem comprometer a privacidade e segurança dos dados de cada um.

### Modelos de Serviços

Os modelos de serviços delimitam as atribuições as quais o usuário pode gerenciar, e as responsabilidades conferidas ao provedor de nuvem. A classificação para cada modelo de serviço é descrita abaixo, conforme Mell e Grance [1] propõem, e pode ser observada na Figura 1.1 por meio da divisão de responsabilidades de gerenciamento/controle.

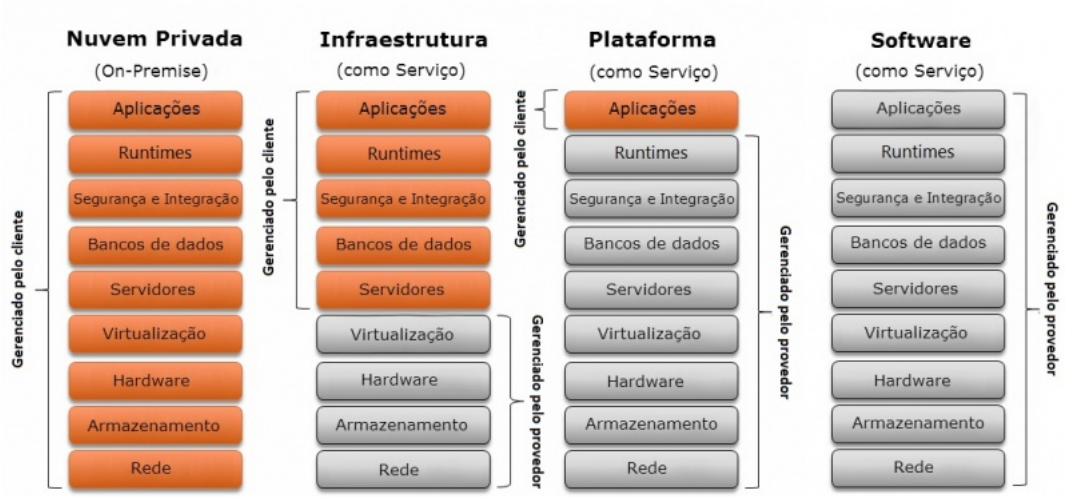


Figura 1.1: Responsabilidade de Gerenciamento/Controle nos diferentes modelos de Serviços.[2]

- Software como um serviço *(SaaS)*: o usuário final faz o uso de aplicações do provedor que estão em execução na infraestrutura de nuvem. Estas aplicações são acessíveis a partir de diversos dispositivos, por meio de uma interface simples, como um navegador web, ou mesmo uma interface mais complexa do programa. O usuário não gerencia ou controla nenhum tipo de recurso da infraestrutura da nuvem, com a exceção de configurações para a aplicação;

- Plataforma como um serviço *(PaaS)*: o usuário final é capaz de implantar aplicações na nuvem fazendo uso de linguagens de programação, bibliotecas, serviços e ferramentas que são suportadas pelo provedor de nuvem. O usuário não gerencia ou controla a

[2]Disponível em:
http://www.geraldoloureiro.com/wiki/index.php?title=Licitação_de_Serviços_de_Computação_em_nuvem
Acesso: 20 de Julho de 2017.

infraestrutura de nuvem subjacente, incluindo rede, servidores, sistemas operacionais ou armazenamento, mas tem controle sobre os aplicativos implantados e, possivelmente, as configurações para o ambiente de hospedagem de aplicativos;

- Infraestrutura como um serviço *(IaaS)*: ao usuário é fornecido como um serviço o acesso aos recursos fundamentais de computação (ex.: processamento, armazenamento, rede), no qual ele poderá implantar e executar diversos softwares, desde sistemas operacionais a aplicações. O usuário possui controle dos recursos que lhe foram fornecidos.

**Modelos de Implantação**

Ainda conforme Mell e Grance [1], para os tipos de serviço de Computação na nuvem, existem quatro tipos de implantação que determinam o grau de segurança e compartilhamento de informações. Os tipos de implantação são descritos a seguir:

- Privada: a infraestrutura da nuvem é de uso exclusivo de uma única organização, podendo ser gerenciada por ela própria ou por terceiros;

- Comunitária: a infraestrutura da nuvem é provisionada para uso exclusivo de usuários, que possuem interesses em comum (ex.: missão, requisitos de segurança, políticas e considerações de conformidade). Pode ser de propriedade, gerenciado e operado por uma ou mais das organizações da comunidade, por terceiros ou alguma combinação delas;

- Pública: a infraestrutura da nuvem é provisionada para uso aberto pelo público em geral. Pode ser de propriedade, gerenciado e operado por uma organização comercial, acadêmica ou governamental, ou alguma combinação deles;

- Híbrida: a infraestrutura é composta por duas ou mais infraestruturas de nuvem distintas (privada, comunitária ou pública) que permanecem entidades únicas, mas, são vinculadas por uma tecnologia proprietária ou padronizada que permite a portabilidade de dados e aplicações.

### 1.1.2 Provisionamento de Recursos em Ambientes de Nuvem

O provisionamento de recursos em ambientes de nuvem se dá conforme a característica de elasticidade e do serviço por demanda. Tratando-se do modelo de serviço *IaaS*, os usuários recebem máquinas virtuais para implantar suas aplicações. Muitas vezes a demanda das aplicações varia com o tempo fazendo com o que o usuário requisite mais recursos para as mesmas. A obtenção de mais recursos pode vir por meio do provisionamento caso a arquitetura das aplicações permita.

Há duas formas de provisionamento, vertical e horizontal. Quando falamos de escalonar horizontalmente (do inglês *Horizontal Scaling* ou *Scaling-Out*), refere-se à adição de novas máquinas virtuais para permitir que a aplicação passe a operar com um melhor desempenho. Esse tipo de provisionamento consegue prover uma maior liberdade de crescimento para alguns tipos de aplicações, apenas [2] é apropriado para situações em que as mudanças da demanda da aplicação ocorra gradualmente ou possa ser prevista [3]. Já quando remetemos ao termo escalonar verticalmente (do inglês *Vertical Scaling* ou *Scaling-Up*), refere-se à modificação dos recursos (ex., CPU, memória, disco) de uma máquina virtual existente durante seu tempo de execução. Este provisionamento é limitado pelas configurações da máquina física [2] e demora menos tempo que o provisionamento horizontal [3]. A Figura 1.2 representa estes tipos de provisionamento.

A abordagem utilizada para provisionamento vertical neste trabalho consiste na mudança do CAP[3] das máquinas virtuais nos servidores, isto é, limitar a velocidade das VCPUs [4].

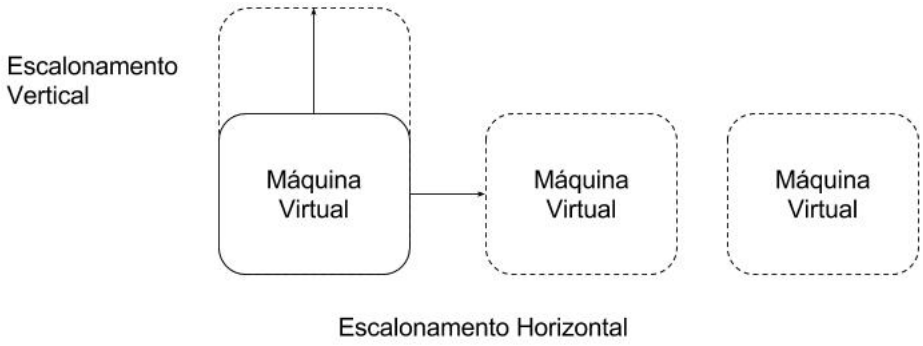


Figura 1.2: Tipos de provisionamento[4]

[3]Limite de capacidade de CPU. Permite limitar a porcentagem de CPUs físicas utilizadas pelas VCPUs.

[4]Modificado do trabalho de Turowski e Lenk [3]

### 1.1.3 OpenStack

No cenário de Computação na nuvem temos o OpenStack [5], definido como um sistema operacional de nuvem, com código aberto, para criação de Nuvens públicas e privadas, o qual permite o gerenciamento dos recursos da infraestrutura de nuvem. No mercado mundial, grandes empresas fazem uso do OpenStack na sua infraestrutura, dentre as quais podemos citar: CERN, Walmart, Volkswagem [6].

O OpenStack é composto por diversos componentes que se relacionam para prover a infraestrutura de nuvem, e cada componente é responsável por um serviço. Os principais componentes necessários em uma infraestrutura de nuvem seriam os seguintes: *Keystone* (serviço de identidade), *Glance* (serviço de imagens), *Nova* (serviço de computação), *Neutron* (serviço de rede) e *Horizon* (interface gráfica). Todos os componentes disponibilizam APIs REST, clientes em Python e terminais para que se tenha acesso as funcionalidades por eles providas.

Os serviços do OpenStack que compõem a nuvem de produção do Laborátorio de Sistemas Distribuídos (LSD) da Universidade Federal de Campina Grande (UFCG) estão ilustrados na Figura 1.3.

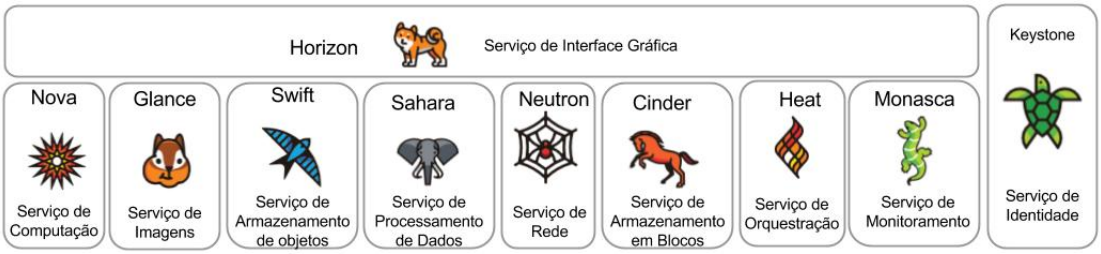


Figura 1.3: Serviços que compõem a Nuvem OpenStack no LSD

Dentre todos os componentes existentes, para os objetivos deste trabalho, é necessário entender a responsabilidade de três componentes, a saber:

- *Keystone* [7]: é o serviço responsável por prover autorização e autenticação para os usuários, restringindo os tipos de ações que o usuário pode executar na nuvem.
- *Nova* [8]: é o serviço responsável por prover acesso aos recursos computacionais de forma escalável e sob demanda. Dentre suas responsabilidades, podemos destacar a criação de máquinas virtuais e sua alocação, migração viva de máquinas virtuais.

- *Monasca* [9]: é uma solução de monitoramento como um serviço, tolerante a falhas, altamente escalável e responsável por coletar métricas da infraestrutura de nuvem (ex., utilização de CPU de máquinas virtuais e de servidores). Também é possível estendê-lo para coletar métricas de diferentes recursos fazendo uso da sua API e dos clientes disponíveis.

## 1.2 Problema Investigado

Os provedores de nuvem buscam constantemente por formas de aumentar suas receitas e reduzir custos [10]. Estudos apontam que a utilização de recursos da infraestrutura é em torno de 53% para memória alocada correspondente, enquanto que, o uso de CPU é apenas 40% [11]. O gerenciamento de recursos de uma nuvem é um dos problemas mais comuns no âmbito da Computação na nuvem [12; 13], visto que, a demanda por recursos varia com o passar do tempo e os provedores desejam garantir os acordos de nível de serviço (do inglês *Service Level Agreement – SLA*[5])

Os provedores de nuvem podem apresentar cenários onde suas infraestruturas apresentam servidores que estão sobrecarregados com o passar do tempo, pois, há o surgimento de novas aplicações no ambiente ou até mesmo máquinas já existentes passam a operar sob uma maior carga de trabalho, levando a uma maior utilização dos recursos virtuais e físicos. Para estes tipos de cenários se faz necessário balancear as cargas dos servidores, para melhoria de desempenho.

## 1.3 Objetivo

Este trabalho tem como objetivo propor e analisar heurísticas, que auxiliem no balanceamento de carga de servidores físicos em uma infraestrutura de nuvem, na qual alguns servidores se encontram sobrecarregados, de forma a prover melhoria no desempenho dos servidores por meio da realocação de máquinas virtuais por meio da migração. Desta forma, a questão de pesquisa deste trabalho de dissertação é a seguinte:

[5]Consiste num contrato entre duas partes: entre a entidade que pretende fornecer o serviço e o cliente que deseja se beneficiar deste.

- *A utilização de heurísticas no balanceamento de carga de servidores, possibilita algum ganho de desempenho de CPU dos servidores?*

Para atingir o objetivo, algumas atividades foram necessárias, as quais estão elencadas nos objetivos específicos abaixo:

- Propor diferentes heurísticas que balanceiem a carga de servidores, realocando máquinas virtuais quando necessário;
- Implementar as heurísticas propostas;
- Executar experimentos para avaliar e monitorar a redução de utilização de CPU dos servidores sobrecarregados;
- Executar experimentos para validar a possível melhoria de desempenho dos servidores.

## 1.4 Metodologia

Para responder à questão de pesquisa definida na Seção 1.3, inicialmente foi necessário definir as heurísticas que possibilitariam o balanceamento de carga dos servidores. Em seguida, para a implementação das heurísticas, foi essencial o uso da linguagem de programação Python [14], visto que, os serviços OpenStack necessários, conforme a Seção 1.1.3, proveem acesso à suas APIs via bibliotecas Python [15; 16; 17]. Como resultado, esta implementação gerou um *Daemon* que periodicamente avaliou a infraestrutura para verificar a sobrecarga dos servidores de acordo com a heurística selecionada.

Após a implementação, o *Daemon* foi instanciado e configurado para a execução de experimentos na nuvem de produção do LSD, e os resultados destes experimentos foram analisados, utilizando a linguagem de programação R [18], visto que, é uma linguagem útil para análises estatísticas e gráficos.

Para avaliar a performance dos servidores e auxiliar na validação dos experimentos, utilizou-se o *Sysbench*, que é uma suíte de *benchmarks* multi-plataforma que permite a avaliação de diversos parâmetros do sistema operacional, que são importantes para sistemas que executam banco de dados sob carga intensiva [19]. O *benchmark* de CPU que é fornecido

[6]Código Disponível em: https://github.com/bigsea-ufcg/bigsea-loadbalancer

realiza o cálculo de números primos [20]. Na Figura 1.4 é ilustrado o comando e o resultado obtido. Foram utilizadas oito *threads* com um número primo máximo 25000, e o tempo total da execução foi de aproximadamente 8.67 segundos.

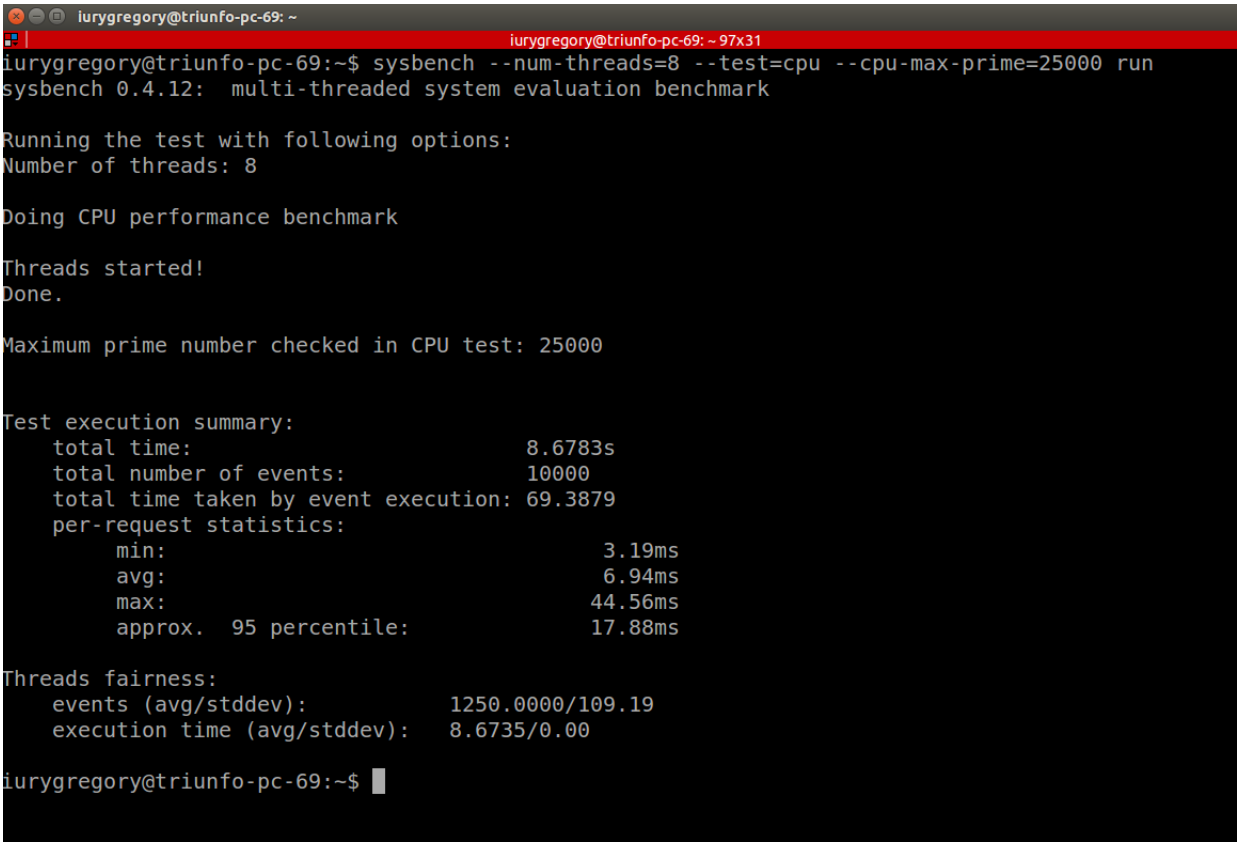


Figura 1.4: Execução do *Sysbench* com teste de CPU em uma máquina do LSD

Foi desenvolvido também um *Daemon* para executar o *Sysbench* nos servidores da nuvem do LSD e publicar os resultados no *Monasca*.

## 1.5 Organização da Dissertação

Esta dissertação está organizada em 5 capítulos. No Capítulo 2, apresentamos os trabalhos relacionados na literatura que serviram de base e motivação, além dos que se assemelham a este. Em seguida, no Capítulo 3 é apresentado a abordagem que foi realizada, constando informações como a arquitetura utilizada, as métricas que foram utilizadas e também as que

[7]Código Disponível em: https://github.com/bigsea-ufcg/monitoring-hosts

foram propostas, detalhes sobre o balanceador de carga desenvolvido e as heurísticas propostas. Já no Capítulo 4 pode ser encontrado uma avaliação e discussão de resultados obtidos pelos experimentos realizados. Ao final, no Capítulo 5 são apresentadas as considerações finais e levantamento de trabalhos futuros.

# Capítulo 2

# Trabalhos Relacionados

Nesse capítulo, listamos os trabalhos relacionados a esta dissertação. Na Seção 2.1, detalhamos os trabalhos que tratam sobre o gerenciamento de recursos em ambientes virtualizados e que contribuíram para a definição do problema abordado; em seguida, na Seção 2.2, comparamos trabalhos existentes que possuem objetivo similar; por fim, na Seção 2.3, explicamos o diferencial deste trabalho.

## 2.1 Gerenciamento de Recursos

Como introduzido na Seção 1.2, o gerenciamento de recursos em ambientes de nuvem se faz necessário para que o provedor consiga garantir os acordos com os usuários e também conseguir uma melhor utilização da sua infraestrutura.

Uma técnica que visa melhorar a utilização dos recursos é o *overcommit*, o qual consiste em um processo de alocação de mais recursos virtuais (CPUs ou memória) do que os recursos físicos existentes [21]. De acordo com Ghosh e Naik [10], o *overcommit* pode trazer problemas de desempenho (aumento do tempo de resposta do serviço) e problemas de disponibilidade (máquinas virtuais podem falhar), devido a quantidade de recursos virtuais em uso é maior que a capacidade física que o provedor pode fornecer, mas também, pode ser benéfico se utilizado de maneira cuidadosa, por meio do monitoramento da utilização dos recursos e predição da utilização dos mesmos.

Existem estudos inseridos na área de computação verde (do inglês, *Green Computing*), a qual se preocupa em manter o desempenho computacional enquanto reduz o consumo de

energia e a emissão de gás carbônico [22], os quais fazem uso da migração de VMs para obter melhoria da utilização de recursos, através da consolidação destas, sem causar degradação à qualidade de serviço ofertada ao usuário. O trabalho de Farahnakian et al. [23] aborda o problema de consolidação de VMs com o objetivo de reduzir o consumo de energia dos *datacenters* e satisfazer os requisitos de QoS (do inglês *Quality of Service*), apresentando uma arquitetura distribuída que consolida as VMs melhorando a utilização de recursos dos servidores e reduzindo seu consumo de energia, a consolidação das VMs faz uso de um algoritmo meta-heurístico de otimização *on-line* altamente adaptável chamado ACS-VMC (*AntColonySystem-based VM Consolidation*) com base em informações obtidas por meio de agentes que monitoram as VMs e os servidores.

Também, destaca-se o trabalho de Wood et al. [24], que apresenta o *Sandpiper*, um sistema capaz de automatizar a tarefa de monitoramento e detecção de pontos de alta utilização (*hotspots*) a fim de manter os SLAs das aplicações, ficando responsável por determinar um novo mapeamento de recursos físicos para virtuais, redimensionar as VMs e iniciar migrações caso seja necessário. Possui uma abordagem caixa preta que é agnóstica às aplicações e sistema operacional e uma abordagem caixa cinza que se baseia em estatísticas da aplicação e do sistema operacional. Como resultados, apresentam que o sistema é capaz de resolver *hotspots* de um único servidor dentro de 20 segundos e é possível escalar para grandes ambientes. A abordagem de caixa cinza ainda permite que sejam tomadas decisões melhores, principalmente quando a memória é o recurso considerado como gargalo.

Afim de evitar a violação de *SLAs* Xu et al. [25] propõem o iAware, que avalia métricas de VMs com base em diferentes *workloads* para determinar a interferência da migração viva, mostrou que é possível utilizar a mesma sem degradação de desempenho.

## 2.2 Comparação de Trabalhos

O trabalho de Carvalho [26] propôs o VOLTAIC (*Volume Optimization Layer To AssIgn Cloud resources*), um sistema de gerência automatizada de recursos para computação na nuvem. Também fornece algoritmos que migram elementos virtuais afim de evitar a perda de desempenho por saturação dos recursos. O algoritmo determina quais máquinas são consideradas em uma situação crítica, e ordena as máquinas físicas em função da carga do sistema

e da quantidade de máquinas virtuais críticas que cada uma das máquinas físicas possuem. Após esta etapa, o algoritmo de seleção de máquinas virtuais candidatas à migração é executado. Este algoritmo itera sobre as máquinas físicas candidatas e busca por máquinas virtuais críticas ordenadas pelo grau de criticidade. Por fim, o sistema itera sobre as máquinas virtuais candidatas, observa o perfil de consumo mais recente e analisa a correlação entre este perfil e o perfil oferecido pelas máquinas físicas, ordenadas da menor carga para a maior carga. Caso o perfil tenha correlação e o consumo médio da máquina virtual seja menor que o oferecido pela máquina física, a máquina virtual é migrada. Se não for encontrado um perfil compatível, a próxima máquina é avaliada e assim sucessivamente.

Alkmin [27] teve o objetivo de minimizar a quantidade de servidores ativos que são necessários para execução dos ambientes virtuais com base no problema *Variable Size Vector Bin Packing*. Para isso, propós a heurística *Grouping VSVBP* que trabalha em conjunto com as heurísticas *FirstFit*, *BestFit Decreasing* e *Slim BFD*. *Grouping VSVBP* é uma adaptação da heurística *Fast Vector Packing algorithm* que é utilizada para resolver o problema de alocação de itens multidimensionais em cestos de tamanhos fixo. Também criou uma ferramenta de simulação de plataformas de Computação na nuvem intitulada SimMyCloud [28], o qual facilita a comparação dos efeitos de se utilizar diferentes estratégias para gerenciamento de recursos.

Magalhães [29] desenvolveu uma estratégia de alocação dinâmica de VMs em ambientes virtualizados com o objetivo de economizar energia e combater sobrecargas que afetem o desempenho de serviços ofertados, visto que, podem provocar violações de SLA. Sua solução é composta principalmente por dois algoritmos, um de monitoramento e um de mitigação. O de monitoramento é responsável por verificar a necessidade de ajustes no mapeamento dos recursos virtuais em físicos. Já o de mitigação pode ser utilizado para consolidação ou distribuição das VMs, fazendo uso de heurísticas de alocação (ex., *Best-Fit*, *Worst-Fit*, *Best-Fit Modified*, *Worst-Fit Modified*) para definir a origem e o destino para o processo de migração, quando usado para distribuição, ainda faz uso de políticas de seleção (Menor Tempo de Migração e Máximo Volume de Carga) para selecionar as VMs mais aptas e consideram o grau de sobrecarga/ócio para memória e processador.

## 2.3 Considerações Finais

Este trabalho se difere dos já citados, visto que, as heurísticas propostas levam em consideração um parâmetro configurável para o *overcommit* desejável da infraestrutura que deve ser informado pelo operador da nuvem, e outro parâmetro que permite informar o número de execuções do balanceador de carga que uma máquina virtual deve esperar antes de voltar a ser considerada na tomada de decisão(mais detalhes serão dados na Seção 3.3). Estes parâmetros são utilizados juntamente com outras métricas, a fim de determinar se os servidores estão sobrecarregados e determinar as migrações que são necessárias para que, na medida do possível, deixe o servidor menos sobrecarregado. O CAP é uma métrica não convencional, que fazemos uso para o balanceamento de carga, visto que, as VMs podem apresentar diferentes valores pois, podem ser escaladas verticalmente. As heurísticas são completamente agnósticas às aplicações que estão nas VMs dos servidores e se preocupam em tentar prover uma melhoria de desempenho aos servidores.

# Capítulo 3

# Abordagem

Neste capítulo detalhamos a arquitetura do balanceador de carga, quais as métricas que foram utilizadas, como se deu a implementação do balanceador de carga e a descrição das heurísticas que foram propostas.

## 3.1 Arquitetura Proposta

O balanceador de carga foi implementado seguindo um arquitetura de *plugins*, com o objetivo de prover uma maior flexibilidade para adição ou remoção de novas heurísticas de balanceamento de carga e suporte para diferentes provedores de *IaaS*. Conforme pode ser observado na Figura 3.1, o balanceador possui cinco módulos principais, cada um com as seguintes responsabilidades:

- CLI: responsável por gerenciar a execução;
- Conectores de *IaaS*: responsável por se comunicar com a infraestrutura de nuvem e repassar as informações dos servidores e de suas VMs para auxiliar o módulo de heurísticas na tomada de decisão. Também é responsável por requisitar as decisões de migração que as heurísticas definiram;
- Serviço de Monitoramento: responsável por prover todas as métricas necessárias a respeito das VMs e dos servidores que serão utilizadas pelas heurísticas;
- Conector KVM: responsável por coletar o CAP das VMs que será utilizado pelas heurísticas;

- Heurísticas de Balanceamento: módulo onde são implementada as heurísticas, as quais são responsáveis por determinar se há servidores sobrecarregados e, caso existam, quais as migrações necessárias para tentar diminuir a carga atual sem sobrecarregar outros servidores.

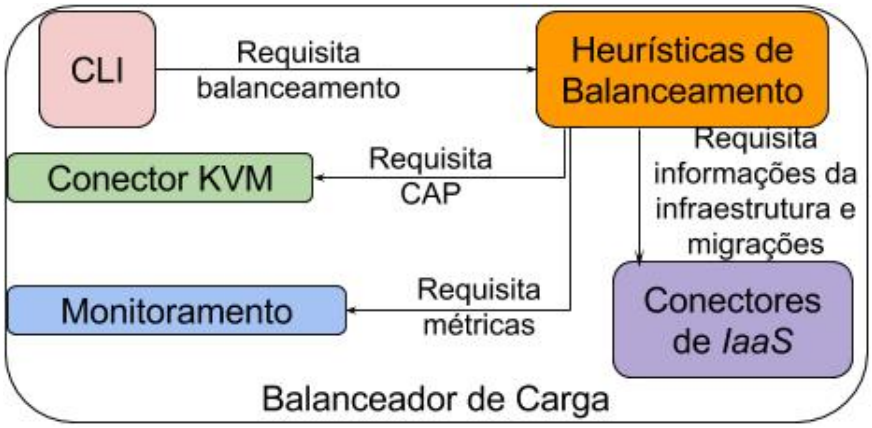


Figura 3.1: Arquitetura dos módulos que compõem o balanceador de carga

## 3.2 Métricas

Esta seção define todas as métricas que foram utilizadas, a fim de, possibilitar uma melhor compreensão das heurísticas que estão descritas nas Seções 3.4.1 a 3.4.3 deste capítulo, e dos resultados que serão apresentados no Capítulo 4. A Tabela 3.1 mostra quais métricas são utilizadas por cada heurística.

Para máquinas virtuais temos as seguintes métricas:

- *vm_vcpus*: número total de VCPUs que a máquina virtual possui. Esta informação é fornecida pelo *Nova*;
- *vm_cap*: valor entre ]0,1] representando a porcentagem de *CAP* que a máquina virtual possui. Esta informação é obtida por meio do *KVM* em cada servidor da nuvem;
- *vm.cpu.utilization_norm_perc*: refere-se à porcentagem normalizada da utilização de CPU, com valores entre ]0,100];

[8] https://github.com/openstack/monasca-agent/blob/master/docs/Libvirt.md#per-instance-metrics

- *vm_used_cap*: refere-se à quantidade de recursos alocados para a máquina virtual, levando em consideração o número de vcpus e o *cap*;

$$vm\_used\_cap = vm\_vcpus \times vm\_cap \tag{3.1}$$

- *vm_consumption*: refere-se à quantidade de recursos que a máquina virtual faz uso, levando em consideração o número de vcpus, *cap* e sua utilização normalizada.

$$vm\_consumption = vm\_vcpus \times vm\_cap \times \frac{vm.cpu.utilization\_norm\_perc}{100} \tag{3.2}$$

Já para os servidores temos as métricas abaixo:

- *host_total_cpu*: número total de CPUs que o servidor possui;

- *host_used_cpu*: número total de CPUs ocupadas pelas máquinas virtuais no servidor, dado pela fórmula;

$$host\_used\_cpu = \sum_{i=0}^{n} vm\_vcpus_i \tag{3.3}$$

- *host_used_cpu_ratio*: proporção de CPU usadas do servidor, dado pela fórmula;

$$host\_used\_cpu\_ratio = \frac{host\_used\_cpu}{host\_total\_cpu} \tag{3.4}$$

- *cpu.percent*: refere-se à porcentagem de tempo que a CPU é utilizada no total;

- *sysbench.cpu.performance*: tempo total (em segundos) que o *Sysbench* levou para concluir a execução do teste de CPU com 8 *threads*;

- *host_cap*: refere-se à capacidade em uso do servidor, dado pelo somatório da capacidade em uso de todas as máquinas virtuais existentes naquele servidor dividido pelo total de CPUs que o servidor possui, conforme a fórmula;

$$host\_cap = \frac{\sum_{i=0}^{n} vm\_used\_cap_i}{host\_total\_cpu} \tag{3.5}$$

- *host_consumption*: refere-se ao consumo do servidor, dado pelo somatório do consumo de todas as máquinas virtuais, existentes naquele servidor dividido pelo total de CPUs que o servidor possui, conforme a fórmula;

$$host\_consumption = \frac{\sum_{i=0}^{n} vm\_consumption_i}{host\_total\_cpu} \tag{3.6}$$

[9] https://github.com/openstack/monasca-agent/blob/master/docs/Plugins.md#cpu

Tabela 3.1: Métricas utilizadas para auxiliar na tomada de decisão das heurísticas

| | **Heurísticas** | | |
|---|---|---|---|
| **Métricas** | Balanceamento de Instâncias | Capacidade de CPU | Performance com Sysbench |
| *vm_vcpus* | ✓ | ✓ | ✓ |
| vm_cap | | ✓ | ✓ |
| *vm.cpu.utilization_norm_perc* | | ✓ | ✓ |
| *vm_used_cap* | | ✓ | ✓ |
| *vm_consumption* | | ✓ | ✓ |
| *host_total_cpu* | ✓ | ✓ | ✓ |
| *host_used_cpu* | ✓ | ✓ | ✓ |
| *host_used_cpu_ratio* | ✓ | | |
| *sysbench.cpu.performance* | | | ✓ |
| *host_cap* | | ✓ | ✓ |
| *host_consumption* | | ✓ | ✓ |

## 3.3 Balanceador de Carga

Conforme definido no Capítulo 1 desta dissertação, este trabalho tem como objetivo a elaboração de heurísticas que auxiliem no balanceamento de carga de servidores sobrecarregados. Para isso, foi implementado o *Daemon*, um balanceador de carga que, até o presente momento só pode ser utilizado em nuvens OpenStack. Sua implantação pode ser feita em qualquer máquina (física ou virtual) desde que, seja garantido o acesso aos serviços necessários para seu funcionamento (*Keystone*, *Nova* e *Monasca*) e acesso ao KVM dos servidores que se deseja balancear para monitorar o CAP das VMs.

Para execução do balanceador é necessário fornecer um arquivo de configuração que contém todas as informações necessárias. Este arquivo é composto por quatro seções (*monitoring*, *heuristic*, *infrastructure* e *openstack*) conforme pode ser visto na Figura 3.2. A Tabela 3.2 oferece a descrição de cada parâmetro das seções deste arquivo.

O fluxo de execução do balanceador pode ser visto na Figura 3.3 e é detalhado abaixo:

1. O administrador da nuvem define o arquivo de configuração (i.e., qual heurística deve ser utilizada e seus parâmetros, credenciais para os serviços, por quais servidores o balanceador será responsável) e inicia-o.

2. Autenticação com o *Keystone* para obter autorização para realizar as operações futuras.

3. Requisição sobre informações de todos os servidores designados, assim como das máquinas virtuais que neles existem, ao *Nova*.

4. Requisição das métricas dos servidores e das máquinas virtuais ao *Monasca*.

5. Requisição de informações extras das máquinas virtuais ao KVM de cada servidor.

6. Tomada de decisão baseada em todas as informações obtidas e requisição das migrações necessárias ao *Nova*.

```
[monitoring]
username=<@username>
password=<@password>
project_name=<@project_name>
auth_url=<@auth_url>
monasca_api_version=2_0

[heuristic]
# A float value that represents the ratio of number of cores in the hosts. (overcommit factor)
cpu_ratio=0.5
# An integer that represent the number of rounds that a instance need to wait before be migrated again
# Each round represents an execution of the loadbalancer
wait_rounds= 1
# The filename for the module that is located in /loadbalancer/service/heuristic/
# without .py extension
module=<module_name>
# The class name that is inside the given module, this class should implement BasicHeuristic
class=<class_name>
#Number of seconds before execute the heuristic again
period=<value>

[infrastructure]
# The user that have access to each host
user=<username>
#List of full hostnames of servers that the loadbalancer will manage (separated by comma).
#e.g compute1.mylab.edu.br
hosts=<host>,<host2>
#The key used to access the hosts
key=<key_path>
#The type of IaaS provider on your infrastructure e.g OpenStack, OpenNebula
provider=OpenStack

[openstack]
username=<@username>
password=<@password>
user_domain_name=<@user_domain_name>
project_name=<@project_name>
project_domain_name=<@project_domain_name>
auth_url=<@auth_url>
```

Figura 3.2: Exemplo de arquivo de configuração do balanceador de carga

Tabela 3.2: Descrição dos parâmetros de cada seção do arquivo de configuração

| Seção | Parâmetro | Descrição |
|---|---|---|
| monitoring | username | nome do usuário para acessar o serviço do *Monasca* |
| | password | senha do usuário para acessar o serviço do *Monasca* |
| | project_name | nome do projeto ao qual o usuário está associado |
| | auth_url | url de autenticação |
| | monasca_api_version | versão da API do Monasca |
| heuristic | cpu_ratio | valor com casa decimal representando a proporção entre número de CPUs virtuais e o número de CPUs física a partir do qual o servidor passa a ser considerado sobrecarregado (fator desejável de *overcommit*) |
| | wait_rounds | número de rodadas que uma máquina virtual que já foi migrada deve esperar antes que possa ser migrada novamente, cada rodada representa uma execução do balanceador. |
| | module | o nome do arquivo onde a heurística foi implementada (sem o .py) |
| | class | o nome da classe da heurística contida no módulo definido |
| | period | número de segundos que se esperar antes de executar a heurística novamente |
| infrastructure | user | nome do usuário com acesso aos servidores |
| | hosts | lista com o nome completo dos servidores separados por vírgula. (ex: compute.mylab.edu.br) |
| | key | localização da chave para acessar os servidores) |
| | provider | nome do provedor de infraestrutura (OpenStack) |
| openstack | username | nome do usuário para acessar os serviços da nuvem |
| | password | senha do usuário para acessar os serviços da nuvem |
| | user_domain_name | nome do domínio da nuvem ao qual o usuário faz parte |
| | project_name | nome do projeto ao qual o usuário está associado |
| | project_domain_name | nome do domínio da nuvem ao qual o projeto está associado |
| | auth_url | url de autenticação |

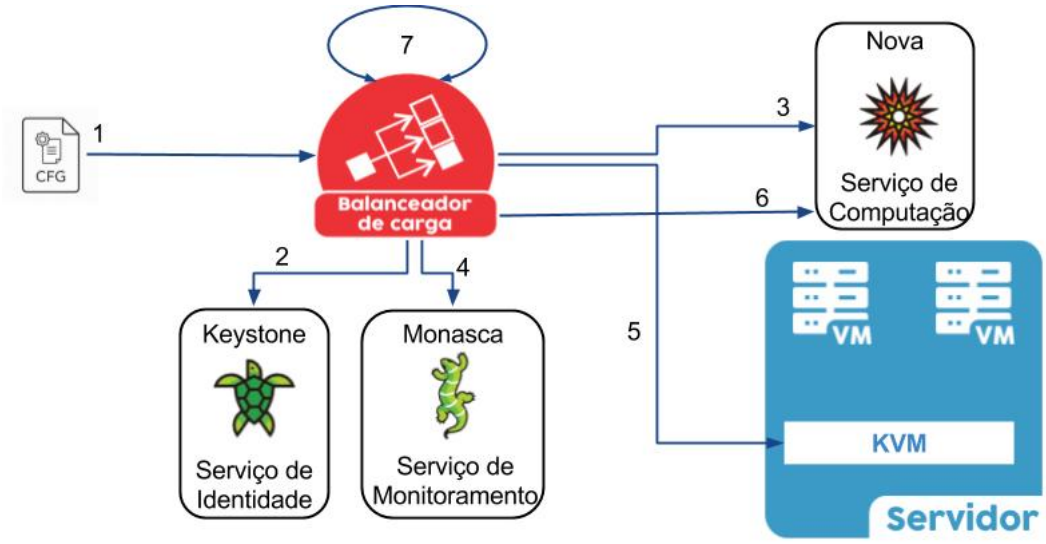


Figura 3.3: Fluxo de execução do Balanceador de Carga

## 3.4 Heurísticas

As heurísticas são responsáveis por verificar periodicamente, quais servidores estão sobrecarregados, tomando ações para realocar as VMs destes servidores para outros, tentando torná-los menos sobrecarregados do que antes, quando possível. Nas subseções abaixo apresentamos as heurísticas que foram desenvolvidas, elas diferem pela métrica utilizada para comparar com *cpu_ratio* para determinar que um servidor está sobrecarregado, e também na escolha das VMs que devem ser migradas.

### 3.4.1 Heurística *BalanceInstancesOS*

Esta é a heurística base, que determina que um servidor está sobrecarregado caso o *host_used_cpu_ratio* do mesmo exceda o limiar dado pela configuração do *cpu_ratio*, conforme apresentado na Inequação 3.7, não se leva em consideração se há realmente utilização nas VMs. A seleção de VMs, para tentar balancear cada servidor, é ordenada de forma decrescente, conforme o número de VCPUs que elas possuem, e que não foram previamente migradas (i.e, *wait_rounds* destas VMs é zero).

Existem alguns cenários em que esta heurística pode ser útil, como por exemplo, quando

há a necessidade de criação e remoção de VMs que possuem aplicações intensivas em CPU, causando o desbalanceamento da infraestrutura, pois, alguns servidores podem ficar além do *overcommit* desejado. Entretanto, quando há VMs, mas suas aplicações não fazem uso de CPU, a infraestrutura pode ser considerada desbalanceada e neste caso a heurística não tende a ser útil.

$$host\_used\_cpu\_ratio > cpu\_ratio \tag{3.7}$$

Para utilizar esta heurística o arquivo de configuração deve receber as seguintes configurações na seção *heuristic*: o parâmetro *module* irá usar o valor *instances* e o parâmetro *class* utilizará *BalanceInstancesOS*. O fluxo de execução desta heurística pode ser visto na Figura 3.4 abaixo.

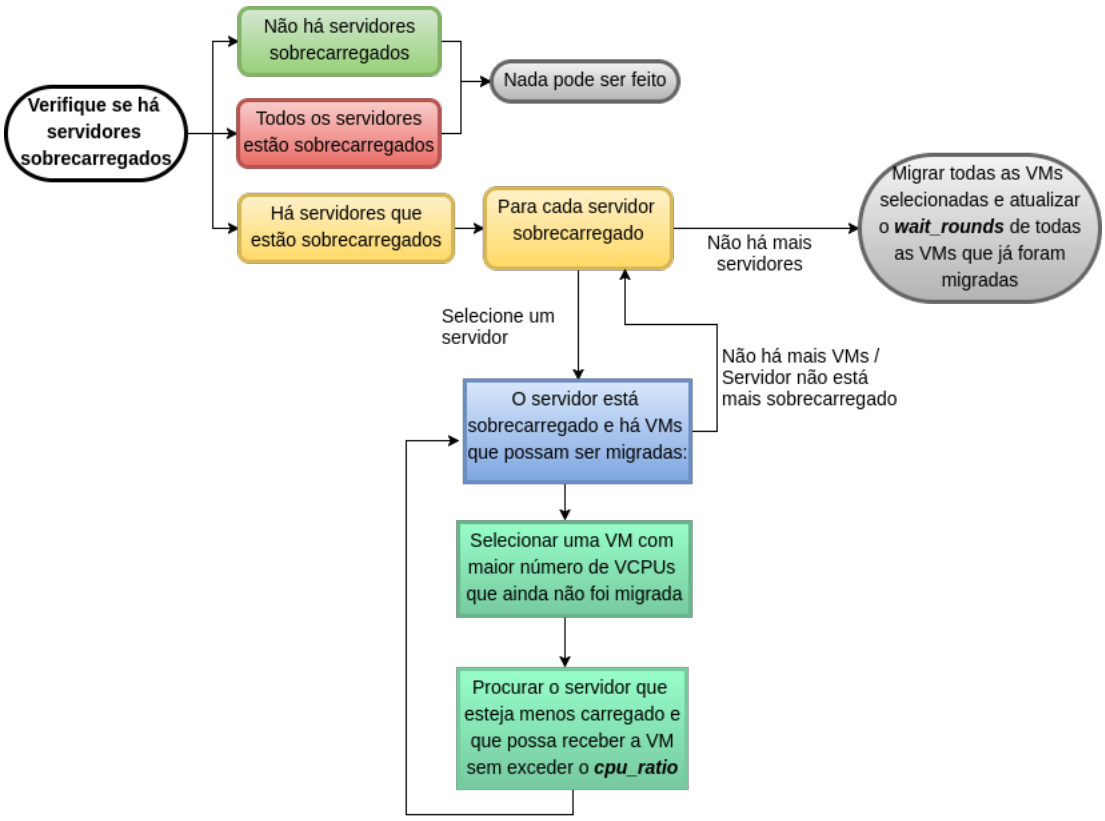


Figura 3.4: Fluxo de execução da Heurística *BalanceInstancesOS*

### 3.4.2 Heurística *CPUCapAware*

Esta heurística diferente da heurística anterior, leva em consideração a utilização de CPU das VMs e mudanças no CAP das VMs que são feitas para fornecer provisionamento vertical às mesmas. Para determinar se um servidor está sobrecarregado, analisamos se o limiar dado pela configuração *cpu_ratio* é extrapolado conforme a inequação 3.8. Em cada servidor sobrecarregado as VMs são ordenadas de forma decrescente conforme o seu consumo (Equação 3.2) e que não foram previamente migradas ainda.

$$host\_consumption > cpu\_ratio \quad ou \quad host\_cap > cpu\_ratio \tag{3.8}$$

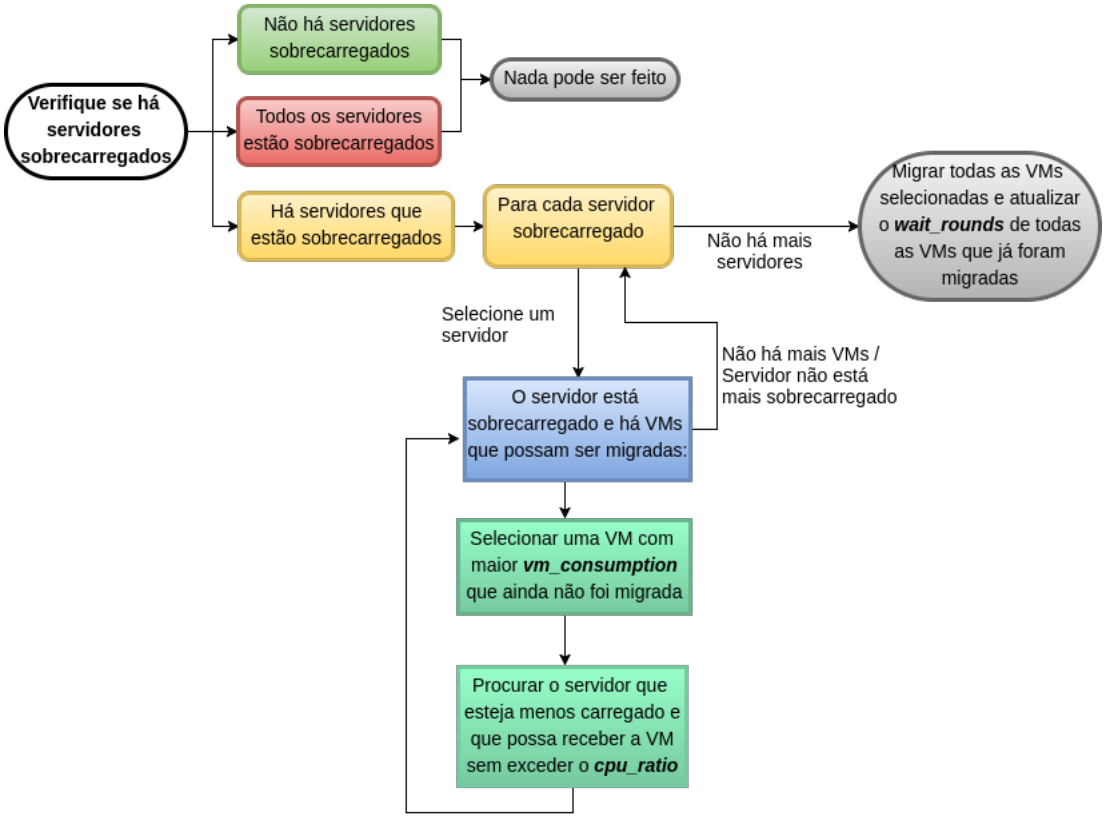


Figura 3.5: Fluxo de execução da Heurística *CPUCapAware*

Quando a infraestrutura faz uso de provisionamento vertical esta heurística é útil, visto que, a utilização de CPU e o CAP passam a ser importantes para determinar que um servidor está além do *overcommit* desejado, porém, ela pressupõem que os servidores da infraestrutura são iguais na questão de desempenho. É importante ressaltar que, o OpenStack não leva em consideração o CAP para alocar as VMs.

Para utilizar esta heurística o arquivo de configuração deve receber as seguintes configurações na seção *heuristic*: o parâmetro *module* irá usar o valor *cpu_capacity* e o parâmetro *class* utilizará *CPUCapAware*. O fluxo de execução desta heurística pode ser visto na Figura 3.5 abaixo.

### 3.4.3 Heurística *SysbenchPerfCPUCap*

Esta heurística é uma melhoria da heurística *CPUCapAware*, que utiliza as mesmas métricas para determinar se um servidor está sobrecarregado, porém, se diferencia na forma de selecionar as VMs que serão migradas. Em cada servidor sobrecarregado é verificado se há servidores livres que possuem melhor desempenho (i.e, os servidores livres possuem tempo de execução menor para o *Sysbench*), caso haja as VMs são ordenadas de forma decrescente conforme o seu consumo (Equação 3.2) e que não foram previamente migradas ainda, caso o servidor ainda continue sobrecarregado é verificado se há servidores com desempenho inferior e as VMs são ordenadas de forma crescente conforme o seu consumo.

Diferente da heurística *CPUCapAware*, esta leva em consideração o desempenho dos servidores, o que leva a uma melhor tomada de decisão na hora de escolher qual VM migrar e para onde.

Para utilizar esta heurística o arquivo de configuração deve receber as seguintes configurações na seção *heuristic*: o parâmetro *module* irá usar o valor *benchmark_performance* e o parâmetro *class* utilizará *SysbenchPerfCPUCap*. O fluxo de execução desta heurística pode ser visto na Figura 3.6.

## 3.5 Considerações Finais

Neste capítulo apresentamos a abordagem para resolver o problema a arquitetura do balanceador de carga e como se da à comunicação dos seus módulos, a definição das métricas utilizadas por cada heurística na tomada de decisão, o balanceador de carga e as heurísticas desenvolvidas.

A heurística *BalanceInstancesOS* faz uso de uma quantidade bem menor de métricas, visto que, não se preocupa com o CAP ou consumo das VMs nos servidores, tratando eles como 100%, sendo assim, a infraestrutura fica sobrecarregada mais vezes mesmo que, as

VMs não estejam consumindo muita CPU.

Na heurística *CPUCapAware* utilizamos mais métricas para avaliar se o servidor está sobrecarregado, já que, passamos a considerar o CAP e a utlização das VMs, porém, não consegue detectar diferença de desempenho nos servidores e sempre migra VMs que causam o maior consumo nos servidores sobrecarregados.

Já a heurística *SysbenchPerfCPUCap* é a heurística *CPUCapAware* melhorada, pois, considera uma métrica a mais, para poder comparar o desempenho dos servidores. Além de que a decisão de quais VMs mimgram varia de acordo com se há o desempenho do servidor que não está sobrecarregado é melhor ou pior do que o sobrecarregado.

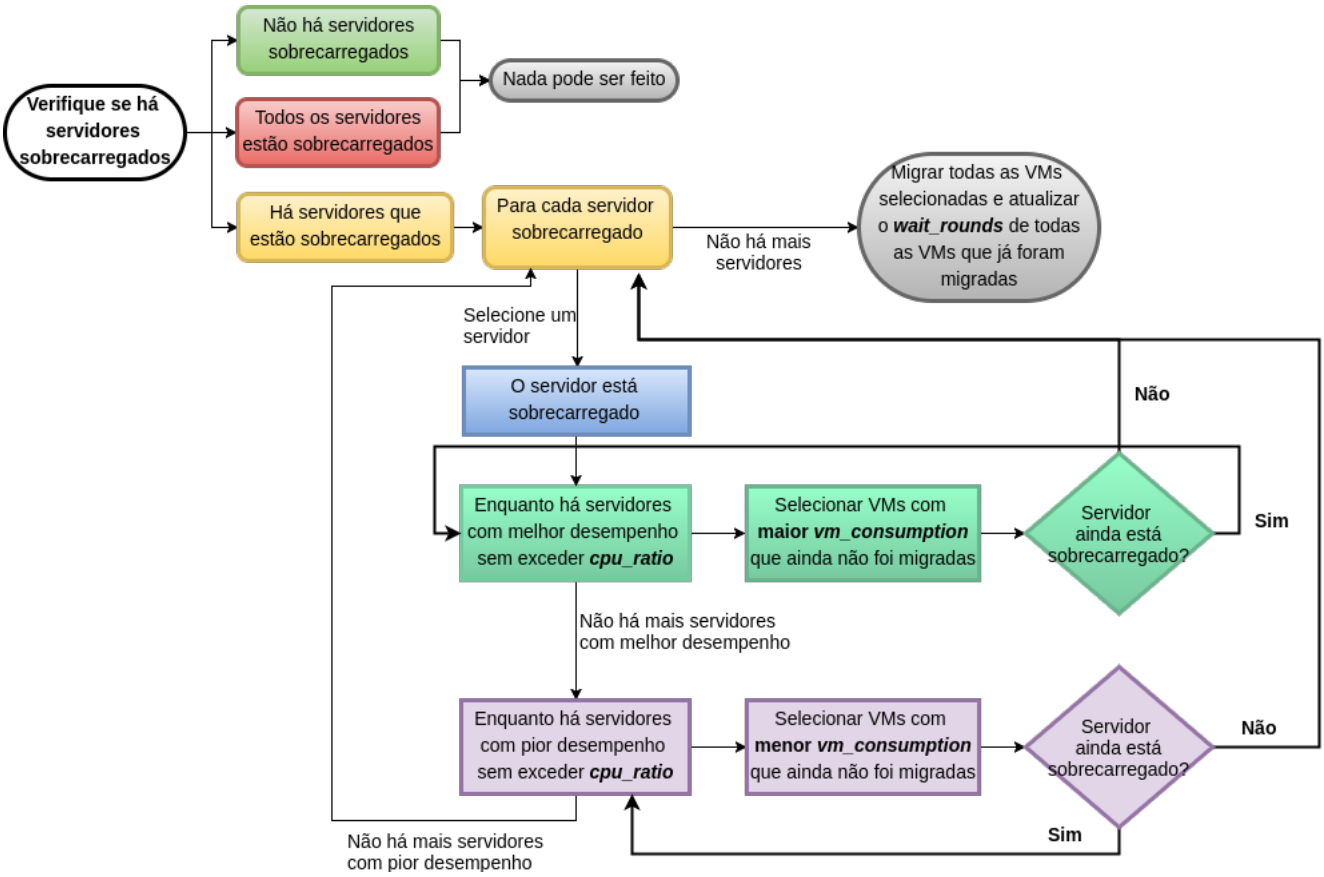


Figura 3.6: Fluxo de execução da Heurística *SysbenchPerfCPUCap*

# Capítulo 4

# Avaliação

Nesse capítulo será detalhado o projeto de experimentos que embasaram o resultado desta pesquisa. Primeiro detalhamos as informações do ambiente de nuvem, no qual os experimentos foram executados. Em seguida tratamos da avaliação do *benchmark Sysbench* para comparar o desempenho dos servidores existentes. Por fim, detalhamos os experimentos que foram realizados com o objetivo de validar as heurísticas propostas.

## 4.1 Ambiente dos Experimentos

Os experimentos realizados para este trabalho foram executados na nuvem de produção do Laboratório de Sistemas Distribuídos (LSD) na Universidade Federal de Campina Grande (UFCG), que utiliza a versão *Newton* do OpenStack. O ambiente disponibilizado foi um conjunto de três servidores dedicados, aos quais se tinha total acesso. Dois destes servidores são *Dell PowerEdge R410* e um *Dell PowerEdge R420*, juntos totalizam 52 CPUs e 82,39 GB *ram*, são conectados através de uma rede Gigabit e utilizam armazenamento compartilhados com o uso do Ceph[10]. O Ceph é amplamente utilizado em Nuvens OpenStack, o relatório atual [30] mostra que 48% das Nuvens de produção fazem uso do Ceph para armazenamento. Podemos ver na Tabela 4.1 as configurações que os servidores utilizados para os experimentos possuem, o servidor3 foi utilizado apenas na comparação do desempenho dos servidores (pré-experimento).

[10]http://ceph.com/

Tabela 4.1: Informações dos Servidores

| Servidores | Modelo | Processador | CPUs | Memória (GB) |
|---|---|---|---|---|
| servidor3 | *Dell PowerEdge R410* | Intel Xeon X5675 3.07GHz | 24 | 31.4 |
| servidor1 | *Dell PowerEdge R410* | Intel Xeon X5675 3.07GHz | 24 | 31.4 |
| servidor2 | *Dell PowerEdge R420* | Intel Xeon E5-2407 2.20GHz | 4 | 19.56 |

## 4.2 Pré-experimento

Nesta secção serão apresentados os resultados obtidos da avaliação do *Sysbench*, que tem por objetivo verificar se o *Sysbench* pode ser utilizado para comparar o desempenho de servidores.

### 4.2.1 Avaliação de Desempenho de CPU dos servidores com *Sysbench*

O *Sysbench* tem um papel importante neste trabalho, visto que, ele é o indicador para o desempenho dos servidores. Dito isso, faz-se necessário avaliá-lo em diferentes cenários. A Tabela 4.2 apresenta os cenários que foram executados, cada um é definido por: dois servidores dos três disponíveis (conforme Tabela 4.1), o número de *threads*, o número primo máximo passado como parâmetro para o *Sysbench* e o número de VCPUs que estão em uso em cada servidor (0 VCPUs indica que não há VMs nos servidores).

Para cada cenário apresentado houve 30 replicações de execuções para cada servidor de forma pareada, sendo coletado do tempo de execução do *Sysbench* para fins de segurança estatística.

Tabela 4.2: Cenários do experimento de avaliação do *Sysbench*

| Cenários para avaliação do *Sysbench* | | | |
|---|---|---|---|
| Servidores | *Threads* no *Sysbench* | Número primo máximo | VCPUs em uso |
| servidor1<br>servidor2 | 4 | 25000 | 0 |
| servidor1<br>servidor2 | 8 | 2500 | 0 |
| servidor1<br>servidor2 | 4 | 25000 | 4 |
| servidor1<br>servidor2 | 8 | 25000 | 4 |
| servidor3<br>servidor1 | 4 | 25000 | 0 |
| servidor3<br>servidor1 | 8 | 25000 | 0 |
| servidor3<br>servidor1 | 4 | 25000 | 16 |
| servidor3<br>servidor1 | 8 | 25000 | 16 |

**Avaliação entre servidor1 e servidor2**

O servidor servidor1 apresentou resultados melhores que o servidor servidor2 em todas as execuções e cenários. Para auxiliar nesta investigação foram gerados os *boxplots* dos servidores servidor1 e servidor2 que podem ser vistos respectivamente nas Figuras 4.1 e 4.2. Os resultados para oito *threads* são melhores que os de quatro *Threads* no servidor servidor1, mesmo quando possuem VMs com aplicações consumindo CPU, já no servidor servidor2 isto não é observado já que seu limite físico de CPUs é quatro, levando a resultados próximos ao de quatro *Threads* apenas quando o servidor se encontra sem VMs. O intervalo de confiança com 95% de confiança para a média do tempo total dos cenários apresentado na

Figura 4.3 auxilia na comprovação de que há diferença entre o desempenho dos servidores quando utilizado o *Sysbench*.

Pode-se observar nas Figuras 4.4 e 4.5 que para os cenários os servidores aparentam seguir uma distribuição normal, porém, o servidor servidor1 nos cenários com VMs não segue uma distribuição normal e o servidor servidor2 para os cenários sem VMs também não segue já que o *p-value* encontrado foi menor que 0,05, os resultados dos testes de normalidade se encontram na Tabela 4.3, sendo assim, foi escolhido o teste de *Wilcoxon* para avaliar se pertencem a uma mesma distribuição, todos os resultados apresentaram *p-value* menor que 0,05 refutando que a hipótese nula de que seguem a mesma distribuição, a Tabela 4.4 contém os resultados do teste.

Tabela 4.3: Resultados do Teste de Normalidade *Shapiro-Wilk* para servidor1 e servidor2

| Servidor | *Threads* no *Sysbench* | VCPUs em uso | *P-value* |
|---|---|---|---|
| servidor1 | 4 | 0 | 0,5598 |
| servidor1 | 4 | 4 | 3,996e-08 |
| servidor1 | 8 | 0 | 0,1623 |
| servidor1 | 8 | 4 | 0,01398 |
| servidor2 | 4 | 0 | 0,02274 |
| servidor2 | 4 | 4 | 0,6195 |
| servidor2 | 8 | 0 | 0,02294 |
| servidor2 | 8 | 4 | 0,9667 |

Tabela 4.4: Resultados do Teste de*Wilcoxon* para servidor1 e servidor2

| Servidor | *Threads* no *Sysbench* | VCPUs em uso | *P-value* |
|---|---|---|---|
| servidor1 e servidor2 | 4 | 0 | 1,863e-09 |
| servidor1 e servidor2 | 4 | 4 | 1,823e-06 |
| servidor1 e servidor2 | 8 | 0 | 1,863e-09 |
| servidor1 e servidor2 | 8 | 4 | 1,863e-09 |

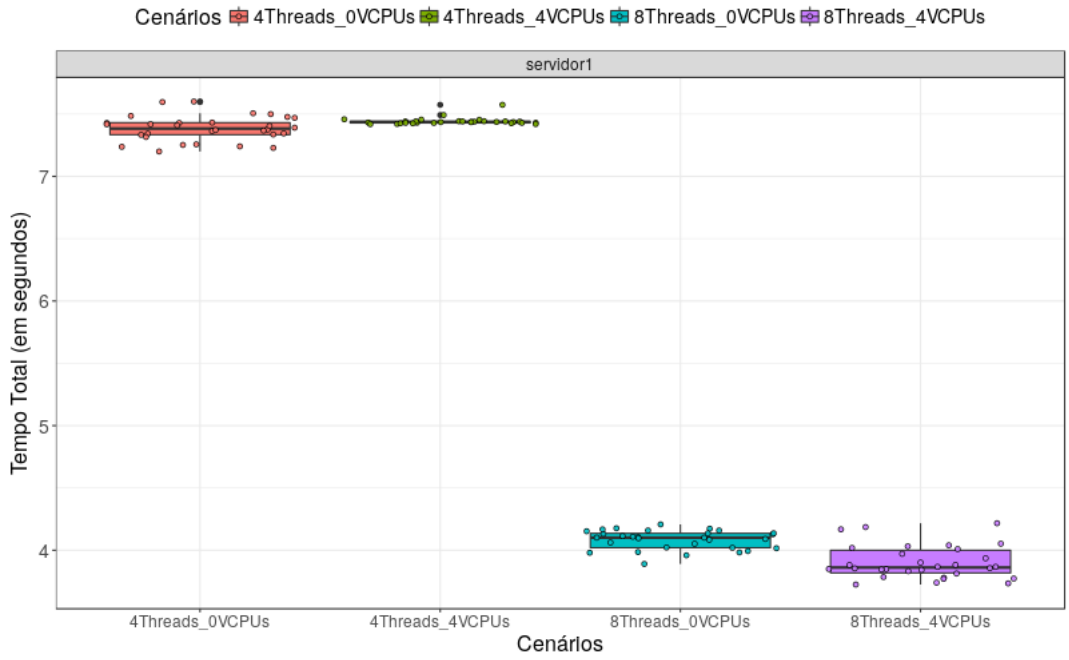


Figura 4.1: *Boxplot* do tempo total de execução do servidor1 em cada cenário

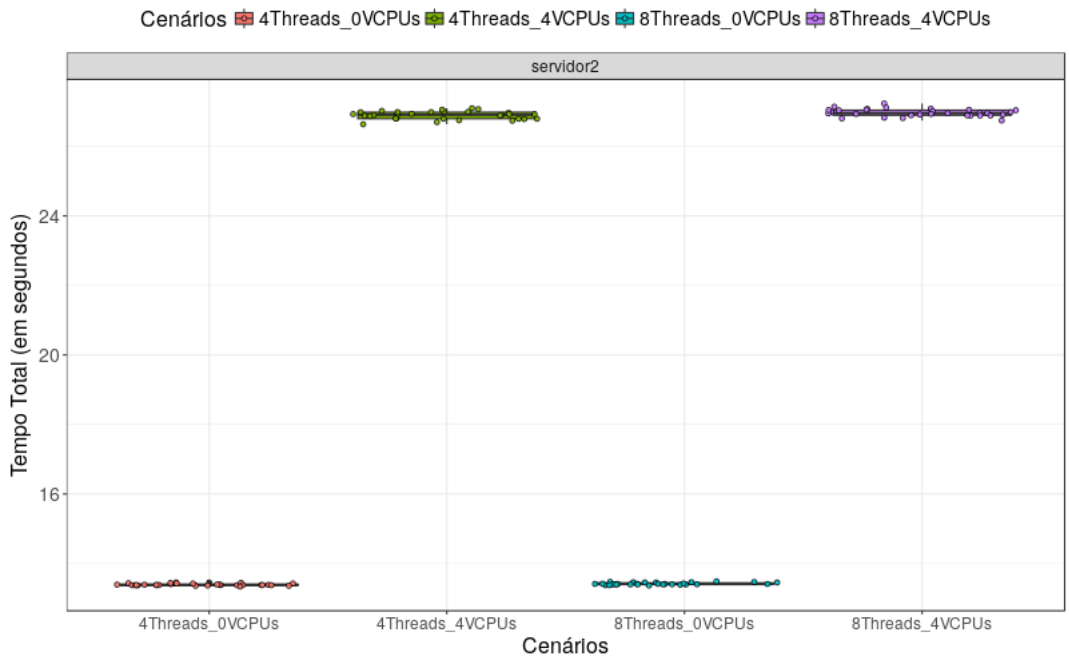


Figura 4.2: *Boxplot* do tempo total de execução do servidor2 em cada cenário

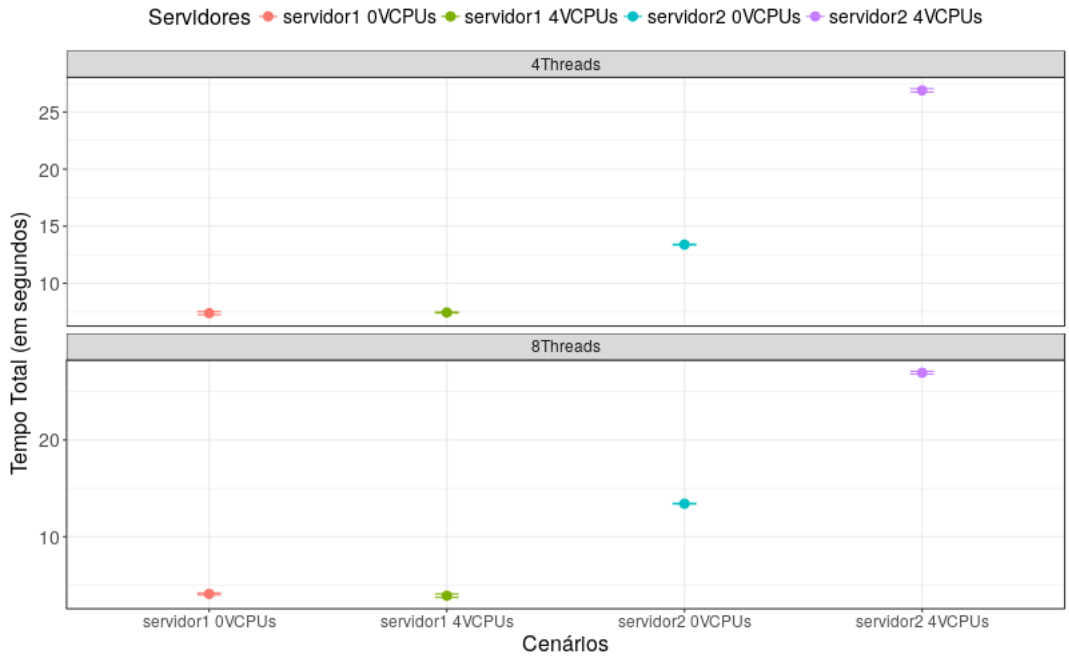


Figura 4.3: Intervalo de Confiança para Tempo Total de execução com 95% de confiança para cada cenário (servidor1/servidor2)

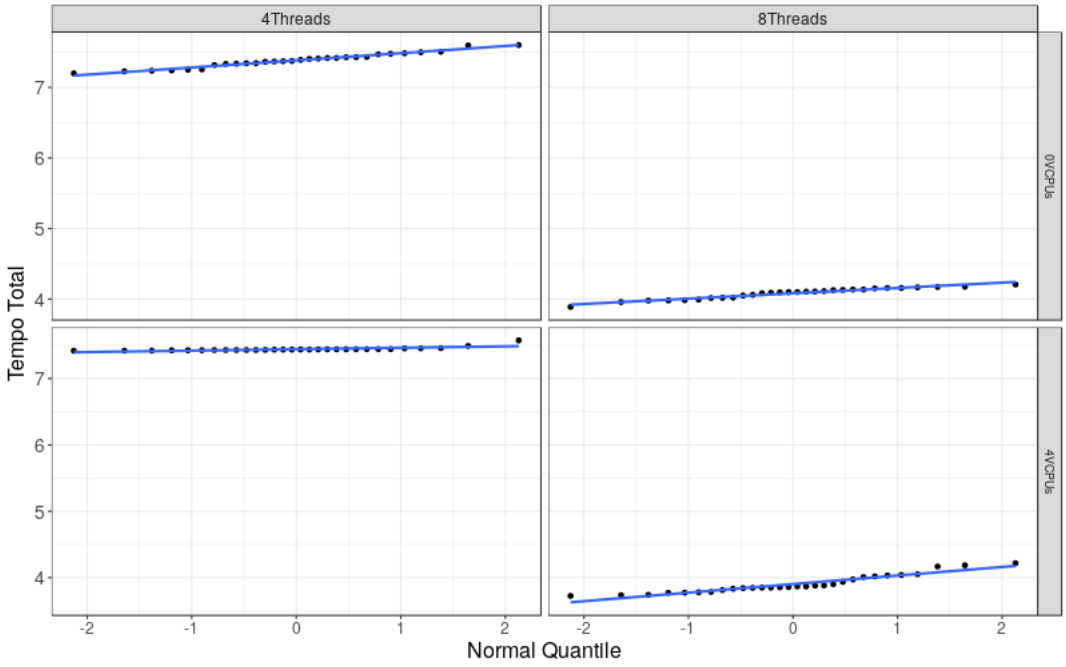


Figura 4.4: Normal QQ-Plot para o servidor1 separado por cenários

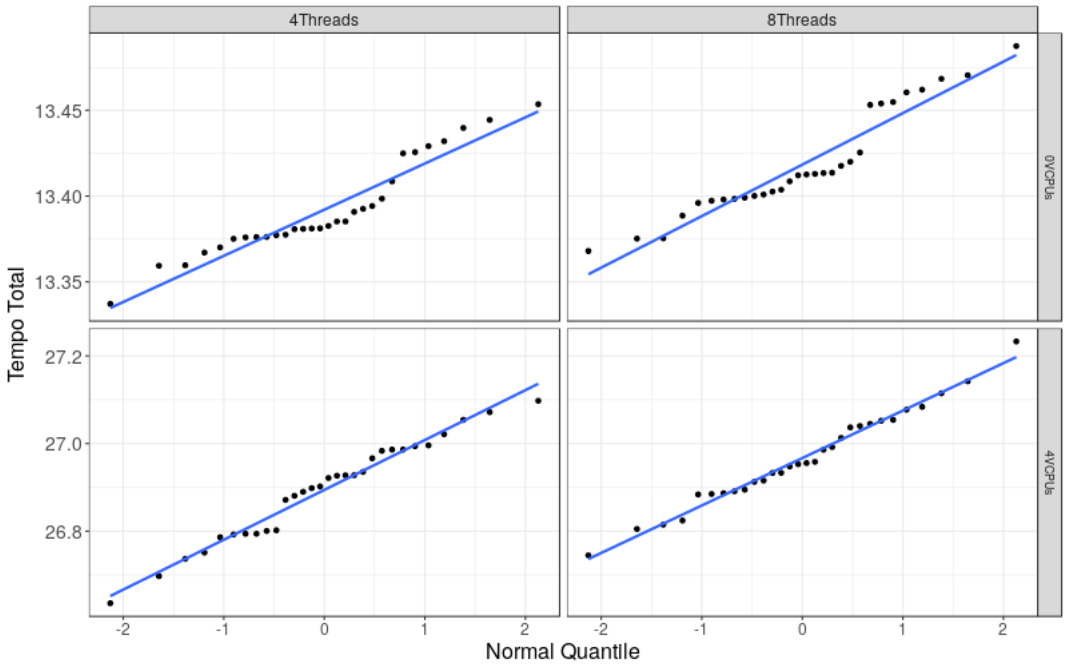


Figura 4.5: Normal QQ-Plot para o servidor2 separado por cenários

### Avaliação entre servidor1 e servidor3

Conforme pode ser visto nos *boxplots* dos servidores servidor3 e servidor1 nas Figuras 4.6 e 4.7 respectivamente, as médias em todos os cenários quando comparamos os servidores são próximas e há uma dispersão maior nos dados do servidor servidor3. O resultado dos intervalos de confiança, que pode ser visto na Figura 4.8, mostra que à média do tempo total de execução do *Sysbench* em todos os cenários para os servidores não são diferentes, já que há uma sobreposição dos intervalos.

Pode-se observar nas Figuras 4.9 e 4.10 que para os cenários os servidores aparentam seguir uma distribuição normal, porém, para o teste de normalidade de *Shapiro-Wilk*, apenas o servidor servidor1 apresentou *p-value* em todos os cenários e o servidor servidor3 para os cenários com quatro *Threads* apresentou *p-value* maior que 0,05, os resultados do teste se encontram na Tabela 4.5. Como houve resultados que indicam que os dados não são normais, foi escolhido o teste de *Wilcoxon* para avaliar se pertencem a mesma distribuição, todos os resultados apresentaram *p-value* maior que 0,05, não refutando a hipótese nula de que seguem uma mesma distribuição, a Tabela 4.6 contém os resultados do teste.

Tabela 4.5: Resultados do Teste de Normalidade *Shapiro-Wilk* para servidor1 e servidor3

| Servidor | *Threads* no *Sysbench* | VCPUs em uso | *P-value* |
|---|---|---|---|
| servidor1 | 4 | 0 | 0,9175 |
| servidor1 | 4 | 16 | 0,5716 |
| servidor1 | 8 | 0 | 0,6725 |
| servidor1 | 8 | 16 | 0,3244 |
| servidor3 | 4 | 0 | 0,08434 |
| servidor3 | 4 | 16 | 0,7676 |
| servidor3 | 8 | 0 | 1,744e-07 |
| servidor3 | 8 | 16 | 7,132e-10 |

Tabela 4.6: Resultados do Teste de*Wilcoxon* para servidor1 e servidor3

| Servidor | *Threads* no *Sysbench* | VCPUs em uso | *p-value* |
|---|---|---|---|
| servidor1 e servidor3 | 4 | 0 | 0,8078 |
| servidor1 e servidor3 | 4 | 4 | 0,1579 |
| servidor1 e servidor3 | 8 | 0 | 0,6583 |
| servidor1 e servidor3 | 8 | 4 | 0,7611 |

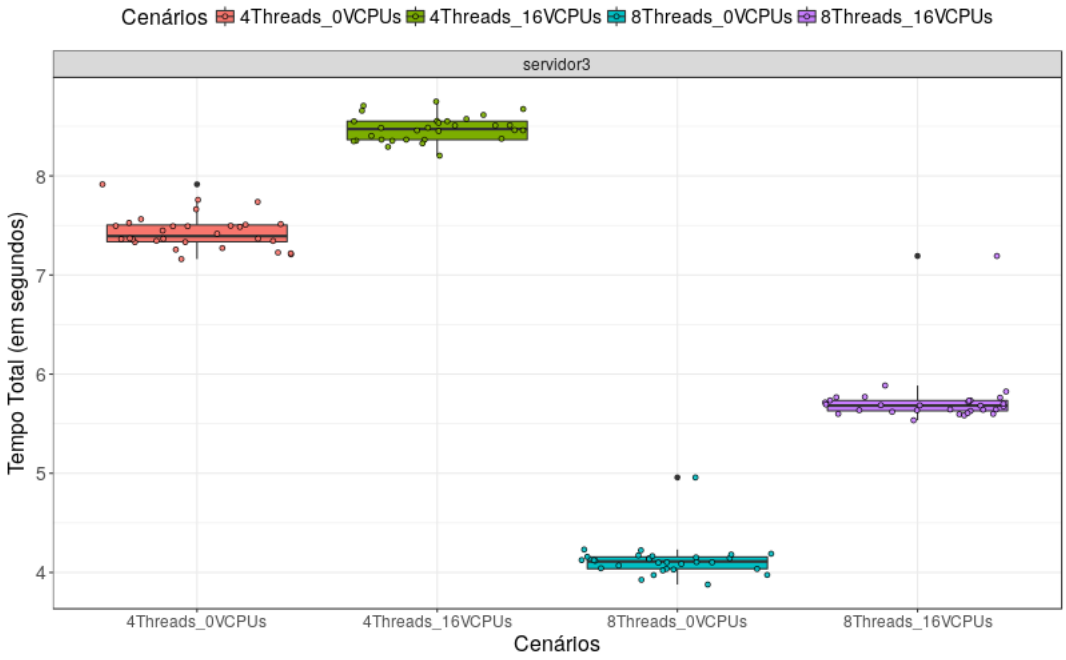


Figura 4.6: *Boxplot* do tempo total de execução do servidor3 em cada cenário

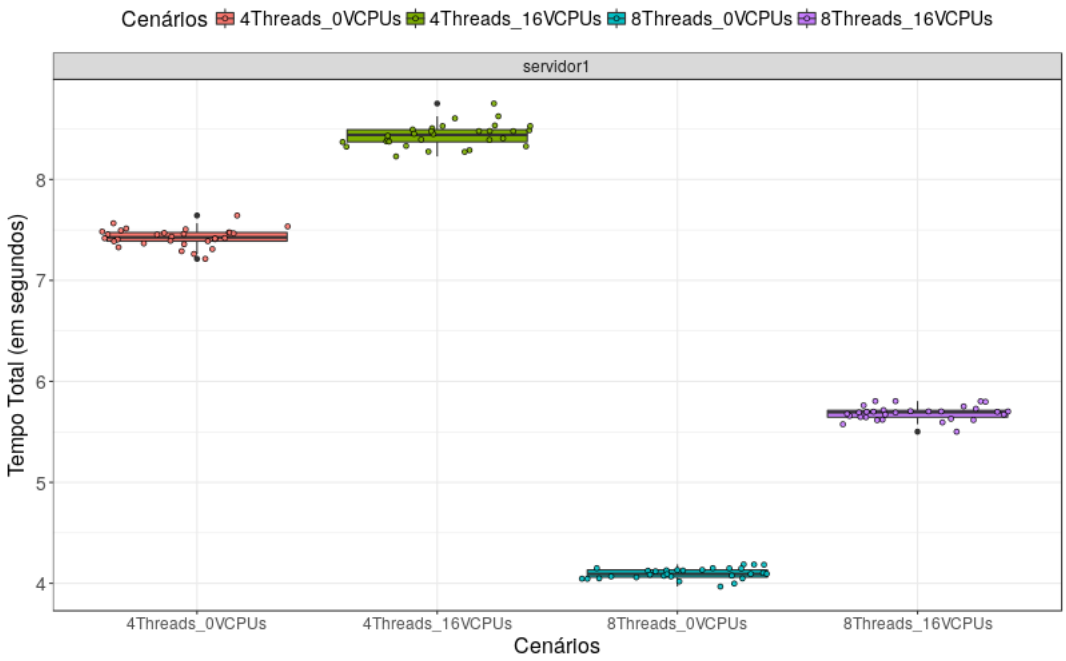


Figura 4.7: *Boxplot* do tempo total de execução do servidor1 em cada cenário

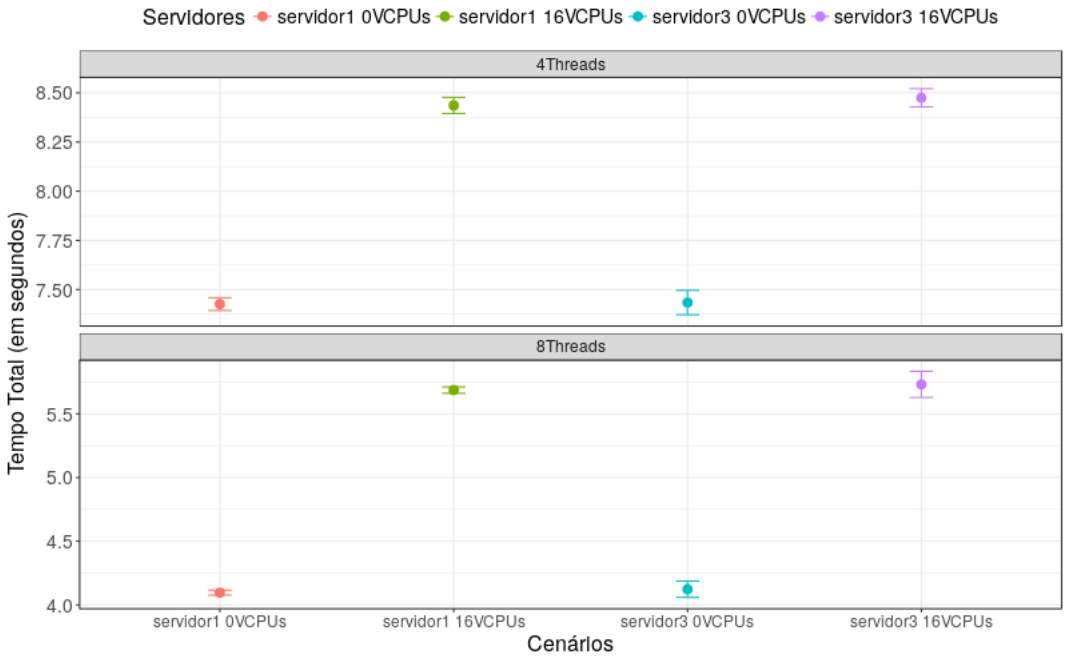


Figura 4.8: Intervalo de Confiança para Tempo Total de execução com 95% de confiança para cada cenário (servidor3/servidor1)

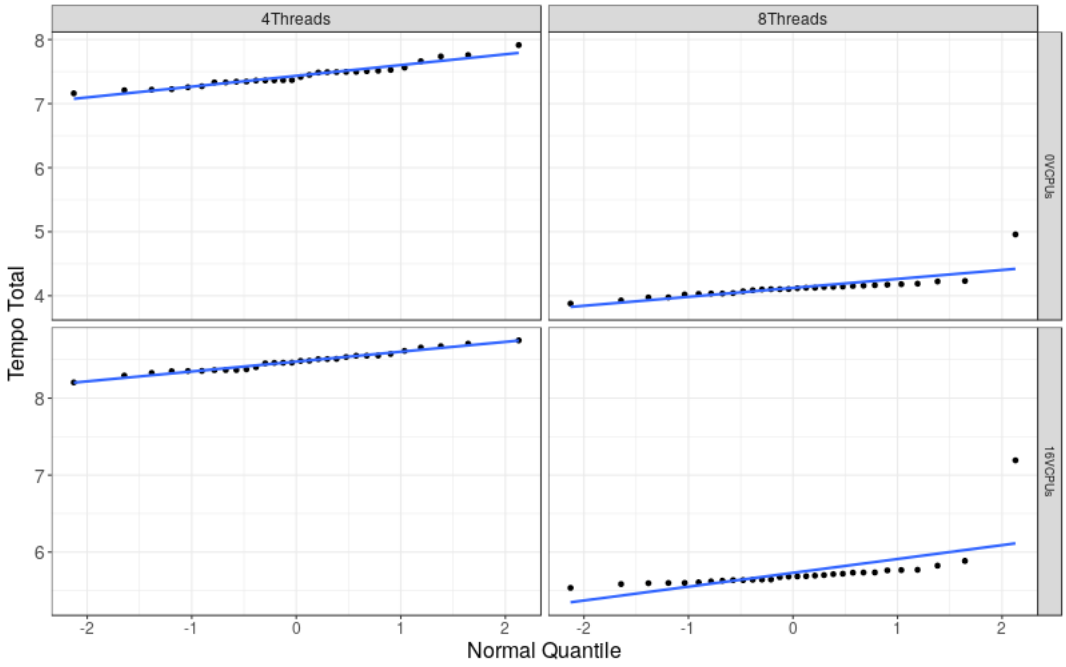


Figura 4.9: Normal QQ-Plot para o servidor3 separado por cenários

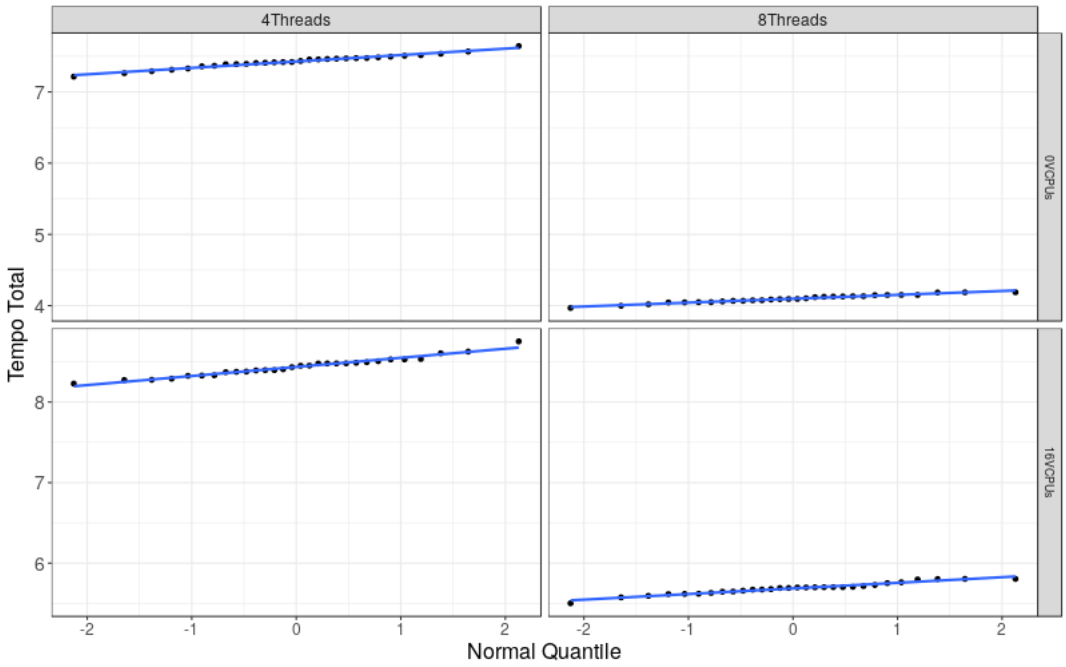


Figura 4.10: Normal QQ-Plot para o servidor1 separado por cenários

**Resultados**

Com base nos resultados obtidos acima, podemos concluir que o *Sysbench* consegue representar o desempenho que os servidores podem vir a apresentar em diferentes condições de carga de trabalho, podendo assim ser utilizado neste trabalho. Os *outliers* foram mantidos visto que, as métricas estão sendo coletadas em servidores físicos que possuem diversos processos e serviços executando sobre os quais não se tem controle.

## 4.3 Experimentos

Foi realizado um experimento com o objetivo de investigar se o uso das heurísticas consegue melhorar o desempenho de um sistema com servidores que estão sobrecarregados. A Tabela 4.7 mostra a configuração inicial dos servidores antes de se iniciar qualquer cenário de avaliação, foram estabelecidos três cenários para avaliar cada heurística, a diferença em cada cenário apresenta-se no CAP aplicado às máquinas virtuais nos servidores. O CAP neste caso reflete a configuração inicial das máquinas e afeta o desempenho dos servidores. Neste capítulo analisamos o CAP em 75%, as outras análises estão nos Apêndices A e B. O fluxo de execução para os cenários é relatado abaixo:

1. Selecionar o CAP inicial [25%, 50%, 75%] ;
2. Primeira rodada do balanceador de carga;
3. Ação - Criação e remoção de máquinas virtuais;
   (a) Criação de uma máquina virtual com 2 *vcpu*s no servidor servidor1 com o mesmo valor de CAP inicial;
   (b) Remoção de duas máquinas virtuais com 1 *vcpu* no servidor servidor2;
4. Segunda rodada do balanceador de carga;
5. Ação - Mudar o CAP de todas as máquinas virtuais dos servidores para 100%;
6. Terceira rodada do balanceador de carga.

Tabela 4.7: Configuração Inicial dos servidores para o experimento

| Servidores | Número de VMs | Número de CPUs alocadas |
|---|---|---|
| servidor1 | 3 | 6 |
| servidor2 | 7 | 8 |

### 4.3.1 Análises para CAP inicial em 75%

Neste cenário todas as VMs começam com 75% de CAP, abaixo observamos 5 execuções de experimento para cada heurística. Avaliamos as métricas que determinam que o servidor está sobrecarregado de acordo com cada heurística, além da utilização de CPU e o tempo de execução do *Sysbench* para comparação de desempenho. O *cpu_ratio* utilizado para este experimento foi de **0.5** é representado pela linha horizontal pontilhada nas figuras abaixo. A Tabela 4.8 apresenta o tempo em que as migrações aconteceram, durante o uso de cada heurística em cada execução, nos gráficos a linha preta vertical tracejada representa estes horários.

Tabela 4.8: Tempo em que as migrações ocorreram (CAP inicial 75%)

| | **Execução** | | | | |
|---|---|---|---|---|---|
| **Heurísticas** | 1 | 2 | 3 | 4 | 5 |
| BalanceInstancesOS | 4 e 28 | 4 e 29 | 4 e 26 | 4 e 32 | 4 e 30 |
| CPUCapAware | 6 e 51 | 4 e 49 | 4 e 48 | 4 e 47 | 4 e 47 |
| SysbenchPerfCPUCap | 4 e 48 | 4 e 52 | 4 e 56 | 4 e 52 | 4 e 52 |

Para a heurística *BalanceInstancesOS* as migrações ocorreram durante a primeira e segunda execução do balanceador de carga, para as outras heurísticas foi durante a primeira e terceira rodada. O motivo para a heurística *BalanceInstancesOS* atuar na segunda rodada por considerar apenas a alocação dos recursos. Da mesma forma as outras duas heurísticas por considerarem a utilização de recursos, atuam apenas na terceira rodada, visto que, previamente há um aumento no CAP das VMs.

Na Figura 4.11 podemos observar o *host_used_cpu_ratio* para a heurística *BalanceIns-*

*tancesOS* nas cinco execuções do balanceador, podemos identificar que o limiar do *cpu_ratio* é extrapolado levando o balanceador a tomar decissões de migração durante a primeira e segunda rodada, na primeira inicialmente o servidor servidor2 já se encontra sobrecarregado, na segunda a criação de VMs no servidor servidor1 o torna sobrecarregado. As variações que podemos observar nas execuções 4 e 5 são devido a algumas VMs não terem apresentado métricas no *Monasca*.

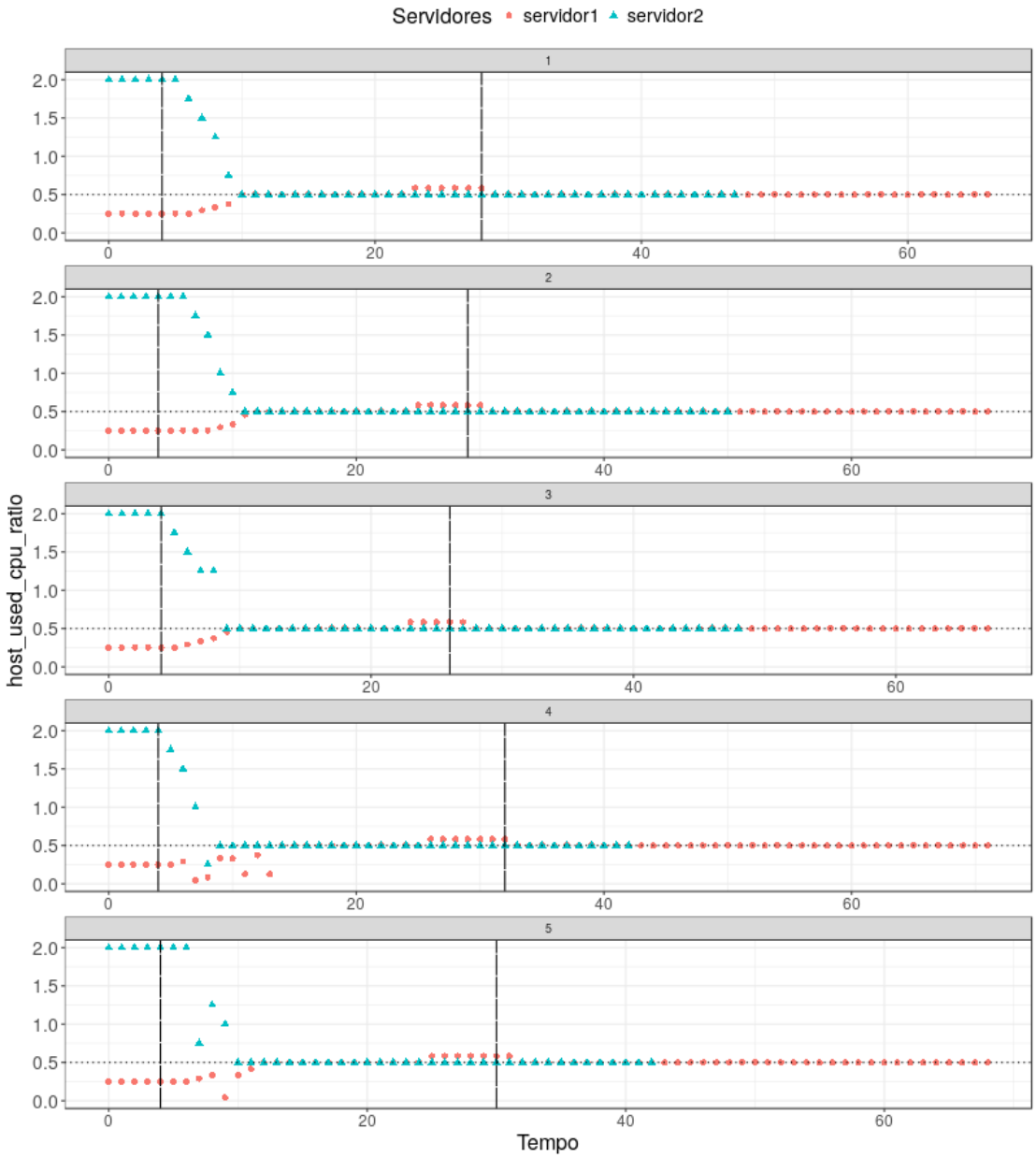


Figura 4.11: *host_used_cpu_ratio* dos servidores durante as execuções com Heurística *BalanceInstancesOS* e CAP inicial em 75%

As Figuras 4.12 e 4.13 representam o *host_total_consumption* para as heurísticas *CPUCapAware* e *SysbenchPerfCPUCap* respectivamente, o qual é responsável por desencadear as migrações durante a primeira rodada do balanceador de carga. As oscilações que ocorrem em algumas destas execuções (ex.: execução 1 para *CPUCapAware* e execução 2 *SysbenchPerfCPUCap*) antes da primeira rodada do balanceador nas heurísticas se deve ao fato de algumas VMs não terem provido métricas no *Monasca*.

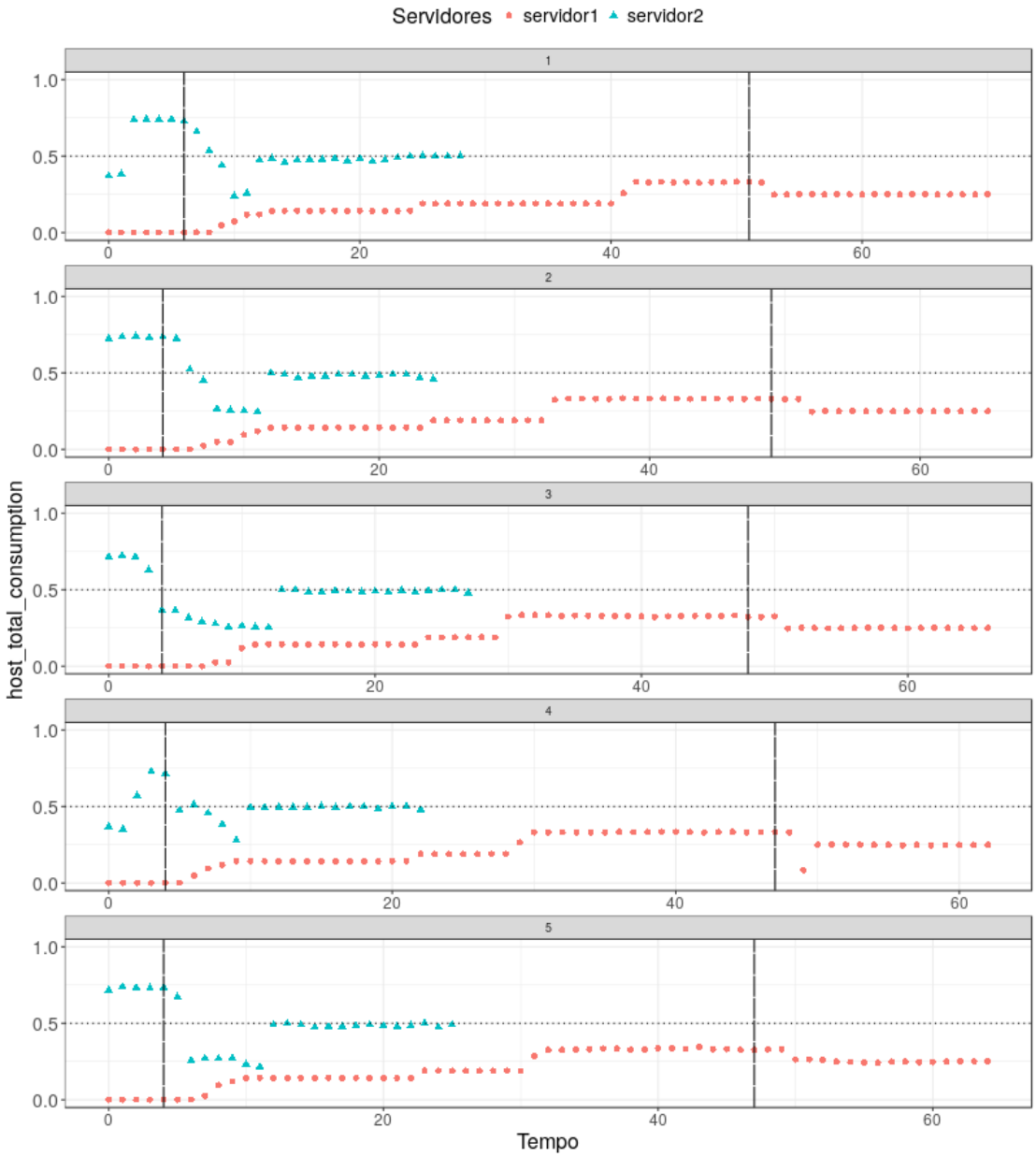


Figura 4.12: *host_total_consumption* dos servidores durante as execuções com Heurística *CPUCapAware* e CAP inicial em 75%

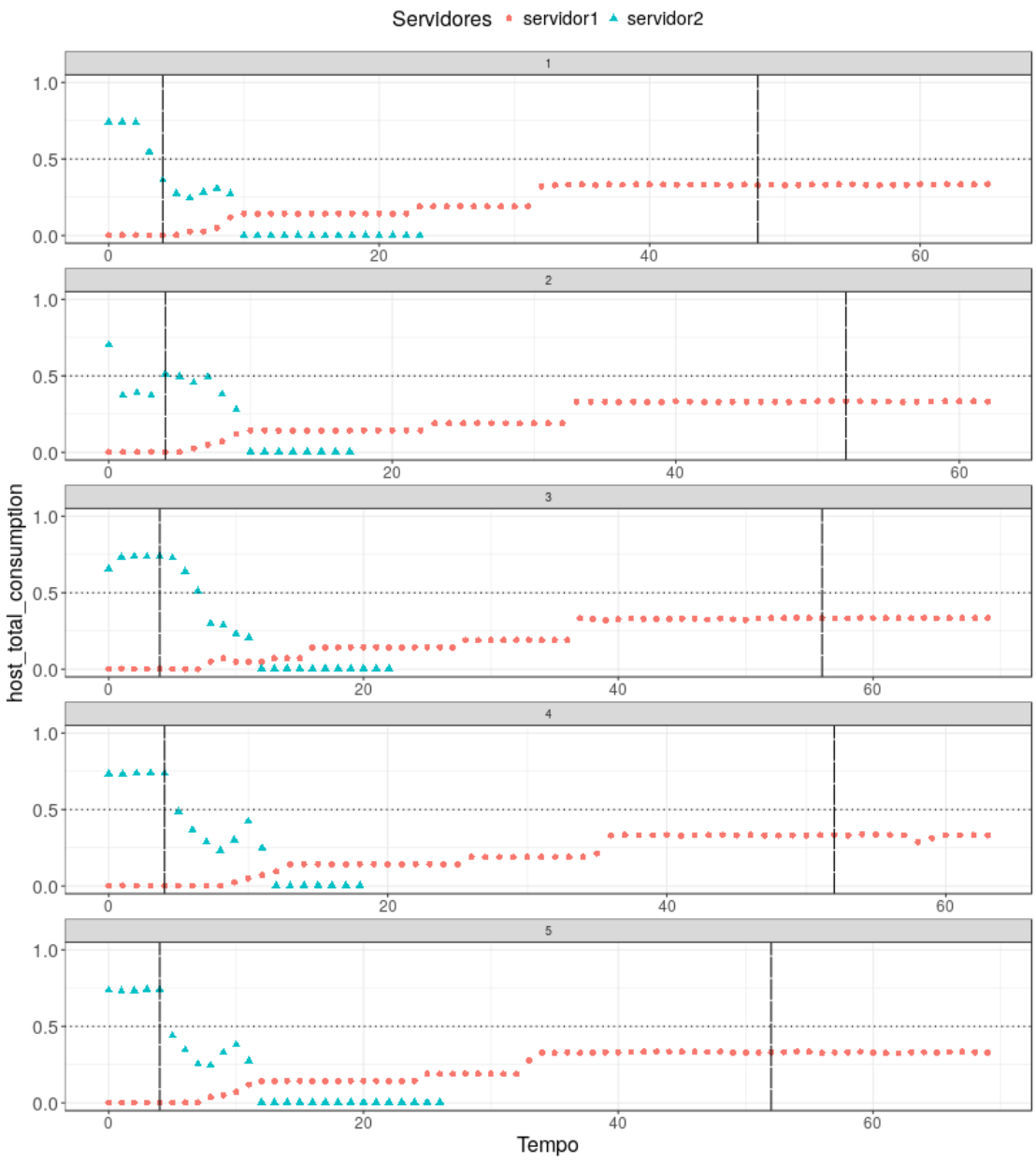


Figura 4.13: *host_total_consumption* dos servidores durante as execuções com Heurística *SysbenchPerfCPUCap* e CAP inicial em 75%

Comparando estas figuras podemos perceber que após as migrações durante a terceira rodada do balanceador, o *host_total_consumption* do servidor servidor2 que recebeu as VMs apresenta diferença, isto acontece pois na heurística *CPUCapAware* migra VMs com maior utilização, já a *SysbenchPerfCPUCap* compara o desempenho do servidor1 que estava sobrecarregado e o servidor servidor2 apresenta desempenho inferior e por isso as VMs migradas

são as que possuiam menor utilização.

As Figuras 4.14 e 4.15 representam o *host_total_cap* para as heurísticas *CPUCapAware* e *SysbenchPerfCPUCap* respectivamente, podemos perceber que a aproximadamente no meio de cada execução o limiar do *cpu_ratio* foi excedido no servidor servidor1, que foi causada pelo aumento do CAP das VMs para 100%, fazendo com que a terceira rodada do balanceador desencadeasse a migração de VMs para o servidor servidor2.

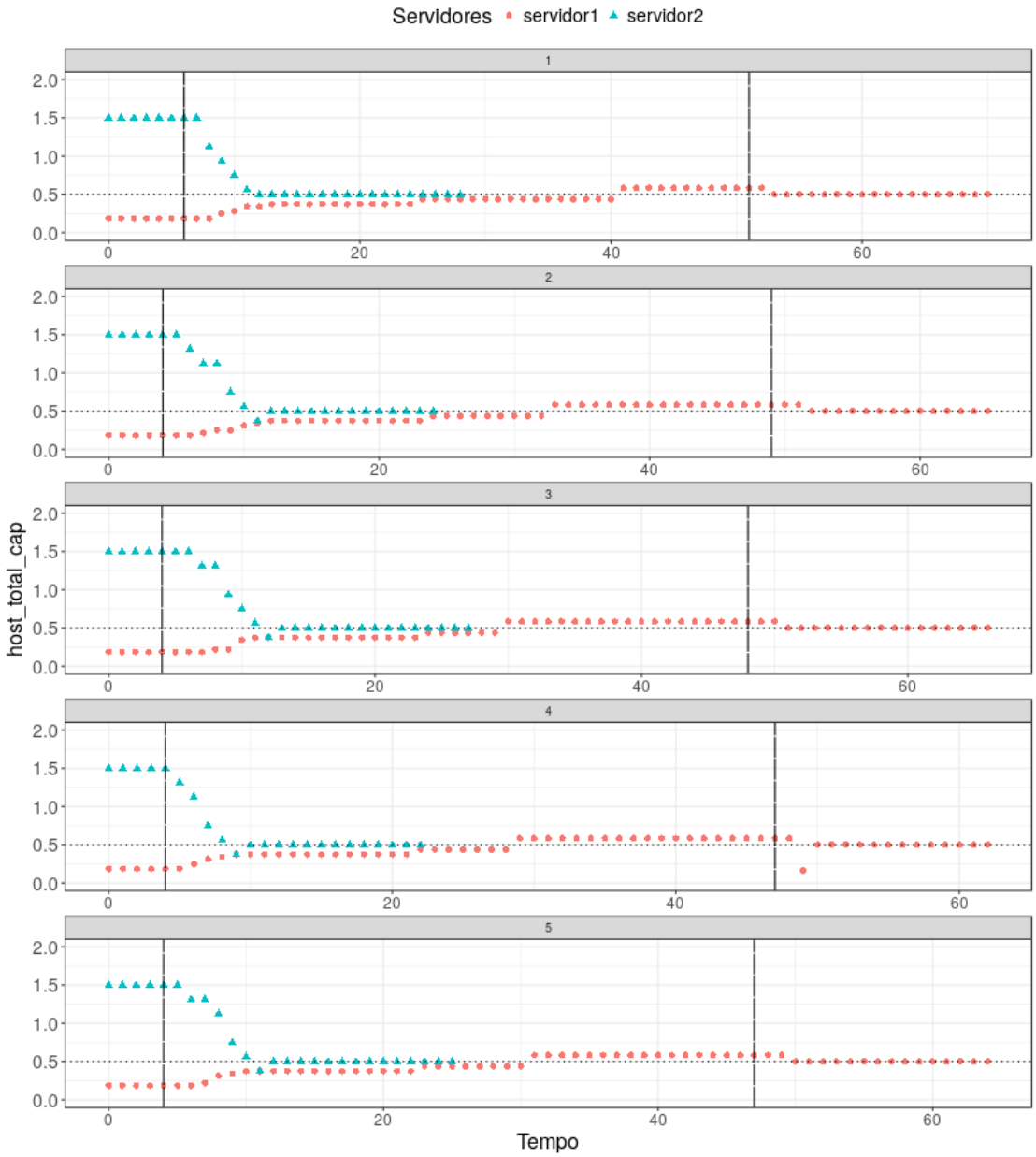


Figura 4.14: *host_total_cap* dos servidores durante as execuções com Heurística *CPUCapAware* e CAP inicial em 75%

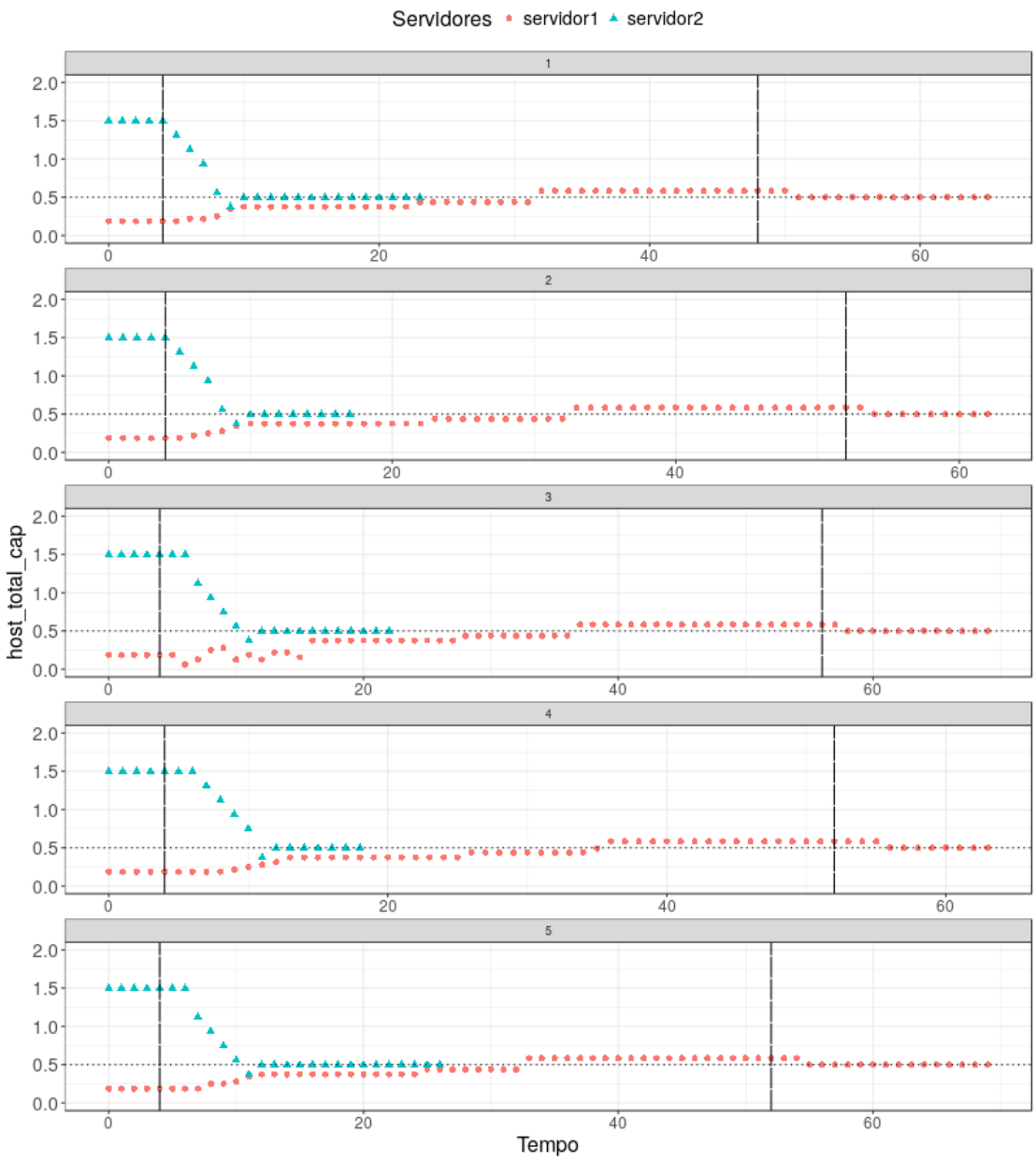


Figura 4.15: *host_total_cap* dos servidores durante as execuções com Heurística *Sysbench-PerfCPUCap* e CAP inicial em 75%

Analisando os gráficos de *host_total_consumption* (Figuras 4.12 e 4.13), *host_total_cap* (Figuras 4.14 e 4.15) e *host_used_cpu_ratio* (Figura 4.11) podemos ver o limite de *cpu_ratio* foi aplicado pelo balanceador em cada execução para todas as heurísticas, também é importante observar que há momentos em que o servidor servidor2 não apresenta métricas, isto ocorre devido a ação de remover VMs após a primeira rodada do balanceador, deixando-o

vazio.

A utilização de CPU dos servidores também foi coletada, esta informação é apresentada nas Figuras 4.16 a 4.18 abaixo, a linha azul e vermelha representam respectivamente os servidores servidor2 e servidor1, já linha vertical tracejada preta representa o tempo no qual as migrações foram iniciadas em cada execução das heurísticas.

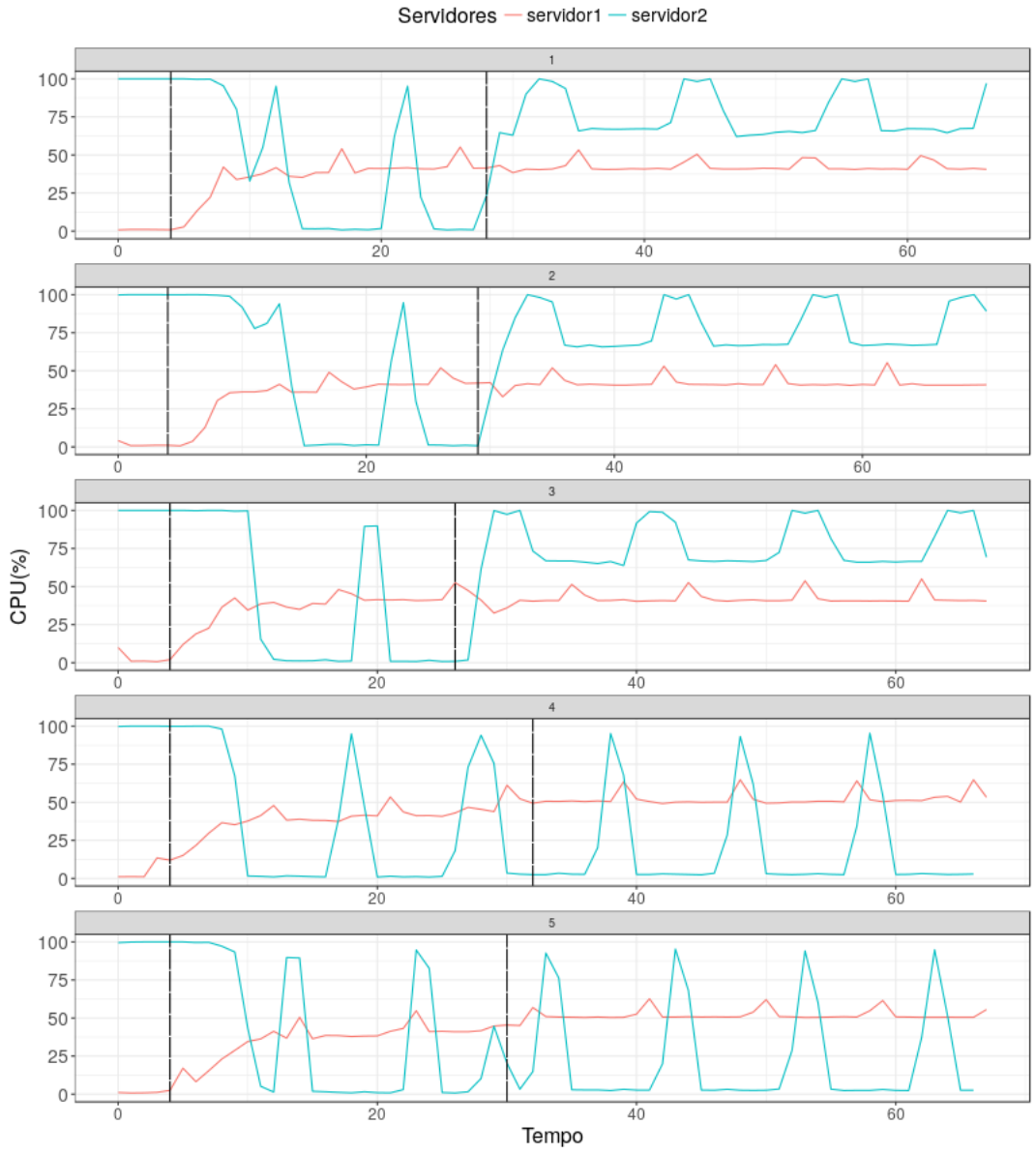


Figura 4.16: Utilização de CPU dos servidores durante as execuções com Heurística *BalanceInstancesOS* e CAP inicial em 75%

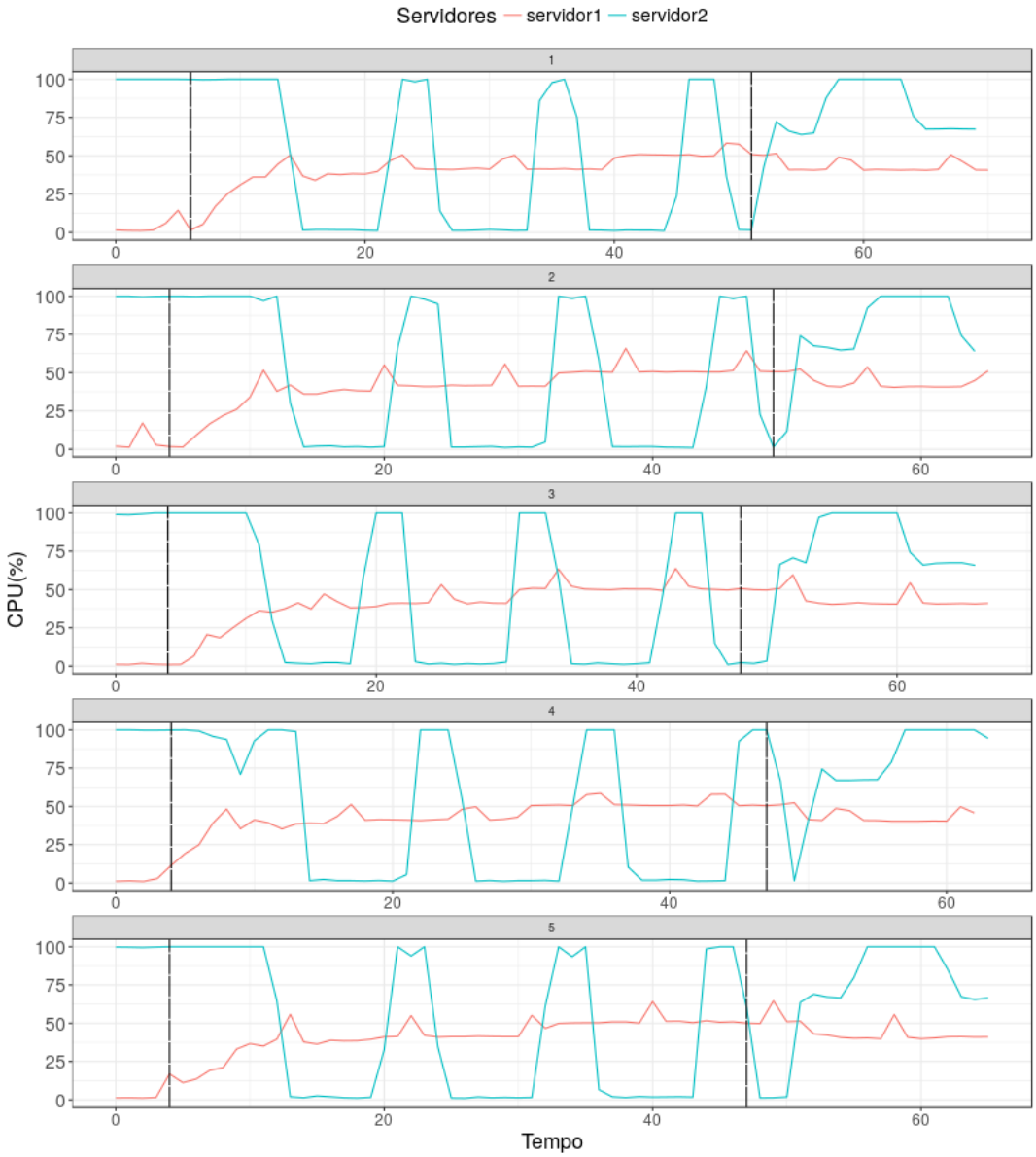


Figura 4.17: Utilização de CPU dos servidores durante as execuções com Heurística *CPU-CapAware* e CAP inicial em 75%

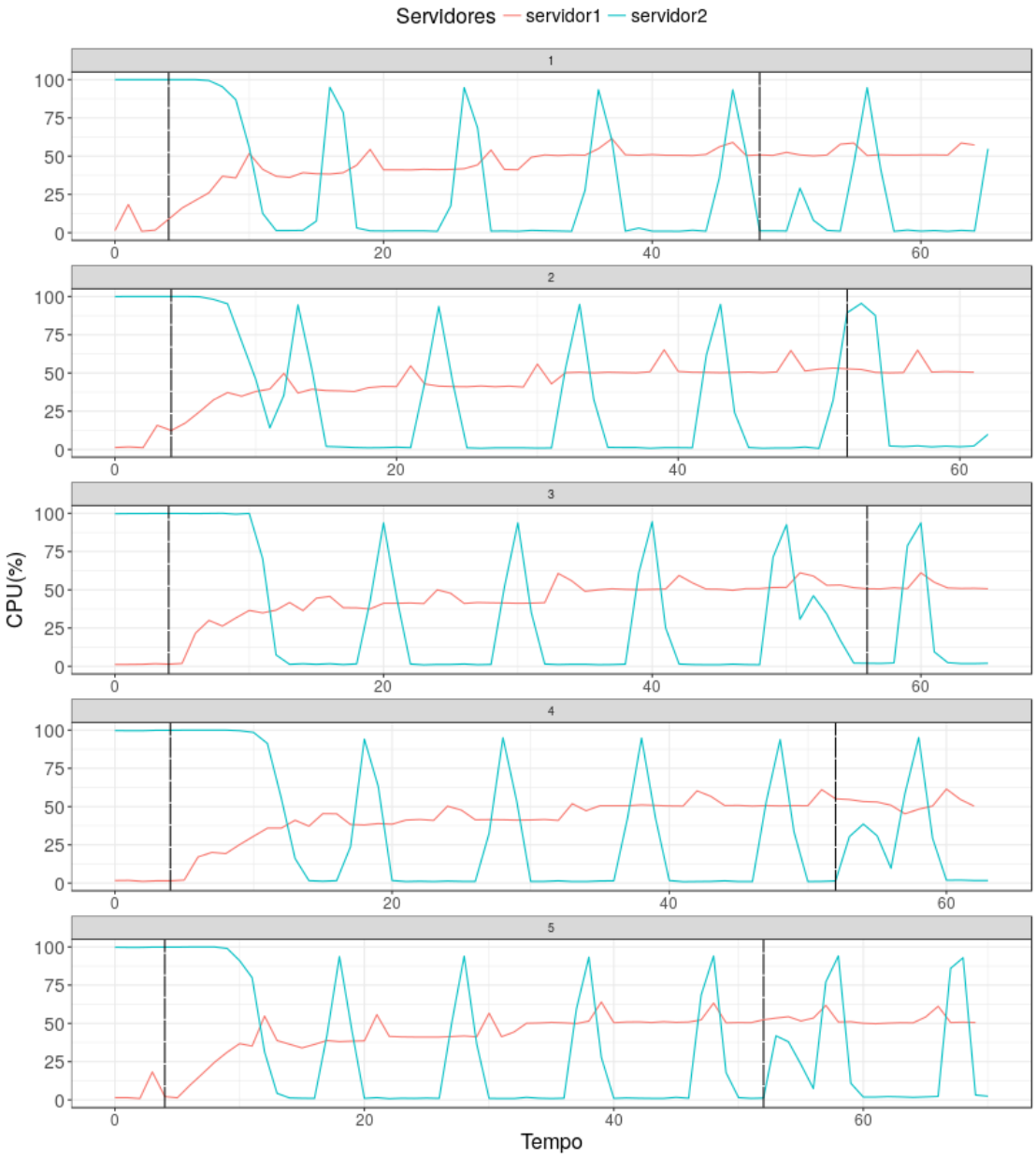


Figura 4.18: Utilização de CPU dos servidores durante as execuções com Heurística *SysbenchPerfCPUCap* e CAP inicial em 75%

O padrão que pode ser observado nestes gráficos é diminuição da utilização nos servidores que estão sobrecarregados quando o balanceador de carga é executado e o aumento de utilização nos servidores que estão recebendo as VMs que foram selecionadas para serem migradas. Além disso, o primeiro pico de utilização é causado pela remoção e criação de VMs, também há picos de utilização que são devidos a execução do *Sysbench* para coletar as

métricas de desempenho, o impacto para servidor2 é mais visível, visto que, possui apenas 4 CPUs, em contraste com 24 CPUs do servidor1.

Para comparar se houve melhoria de desempenho nos servidores, analisamos os tempos de execução do *Sysbench* para os servidores que estavam sobrecarregados, comparando antes e depois da rodada do balanceador onde houve migração de VMs. Antes de escolher o teste para comparar o desempenho, verificamos a normalidade dos dados utilizando o teste de normalidade de *Shapiro-Wilk* e também analisamos os intervalos de confiança, como não houve casos em que, para uma dada heurística e rodada do balanceador um mesmo servidor apresentasse normalidade tanto quando estava sobrecarregado e quando não estava, conforme pode ser visto na Tabela 4.9, sendo assim foi escolhido o teste de *Wilcoxon* para comparar o desempenho.

Tabela 4.9: Resultados do Teste de Normalidade *Shapiro-Wilk* para o tempo de execução do *Sysbench* de acordo com as heurísticas quando houve migrações (CAP inicial 75%)

| Heurística | Rodada do Balanceador | Servidor/Condição | p-value |
|---|---|---|---|
| *BalanceInstancesOS* | #1 | *servidor2/Sobrecarregado* | 9,901e-08 |
| | | *servidor2/Não Sobrecarregado* | **0,953** |
| | #2 | *servidor1/Sobrecarregado* | 0,00155 |
| | | *servidor1/Não Sobrecarregado* | **0.3559** |
| *CPUCapAware* | #1 | *servidor2/Sobrecarregado* | **0,565** |
| | | *servidor2/Não Sobrecarregado* | 0,006373 |
| | #3 | *servidor1/Sobrecarregado* | 0,02259 |
| | | *servidor1/Não Sobrecarregado* | 6,493e-08 |
| *SysbenchPerfCPUCap* | #1 | *servidor2/Sobrecarregado* | 8,824e-07 |
| | | *servidor2/Não Sobrecarregado* | 4,778e-09 |
| | #3 | *servidor1/Sobrecarregado* | 1,761e-05 |
| | | *servidor1/Não Sobrecarregado* | **0,06753** |

Nas Figuras 4.19 a 4.21 podemos observar os intervalos de confiança para as heurísticas *BalanceInstancesOS*, *CPUCapAware* e *SysbenchPerfCPUCap* respectivamente, neles apresentamos os intervalos de quando cada servidor estava sobrecarregado e após não estar mais sobrecarregado. É interessante observar que para todas as heurísticas a diferença existente nos valores do servidor servidor2 é bem perceptível e que há uma diminuição no tempo de

execução do *Sybench* em torno de 50%, quando ele passa a não está sobrecarregado. A única sobreposição de intervalos se deu na heurística *SysbenchPerfCPUCap* com o servidor servidor1 que pode ser observado na Figura 4.21, isto acontece devido ao fato da migração que ocorreu ter sido de uma VM que possuía o menor valor de consumo dentre todas as outras no servidor não afetando o desempenho do mesmo.

Aplicamos os teste de *Wilcoxon* para os dados e chegamos aos resultados obtidos na Tabela 4.10. Os cenários apresentam *p-value* menor que *0.05* refutando a hipótese de que seguem uma mesma distribuição, com exceção do cenário da heurística *SysbenchPerfCPU-Cap* com o servidor servidor1 sobrecarregado o qual o *p-value* foi maior que **0,05** o qual faz sentido, visto que, o intervalo de confiança também apresentou sobreposição e pode ser explicado pelo migração de uma VM que possuia o menor consumo, não afetando o desempenho do servidor.

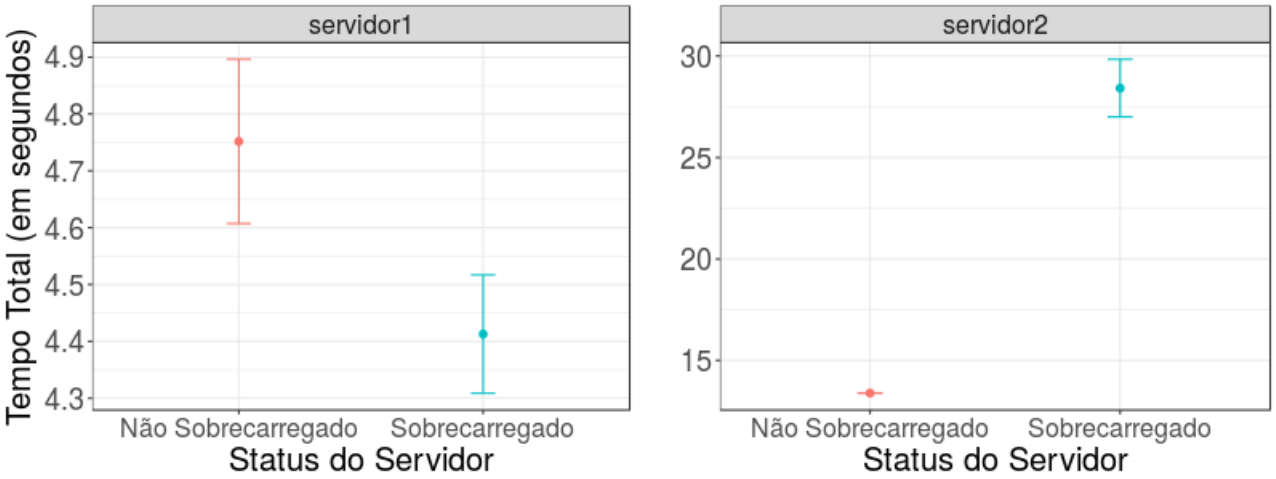


Figura 4.19: Intervalo de Confiança 95% para *BalanceInstancesOS* com CAP inicial em 75% comparando o tempo do *Sysbench* quando os servidores estavam sobrecarregados e no momento que não estavam

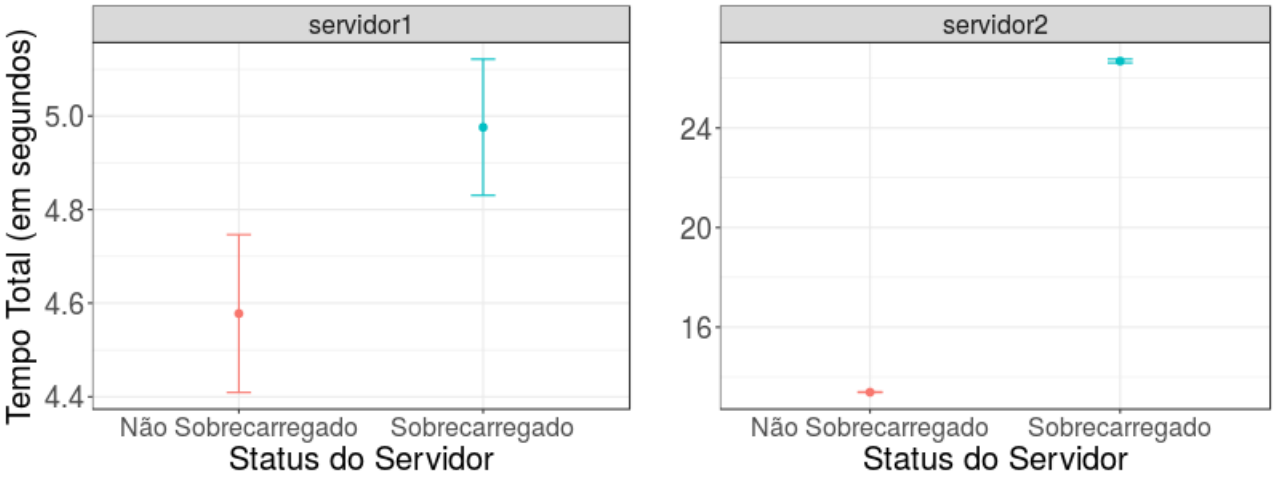


Figura 4.20: Intervalo de Confiança 95% para *CPUCapAware* com CAP inicial em 75% comparando o tempo do *Sysbench* quando os servidores estavam sobrecarregados e no momento que não estavam

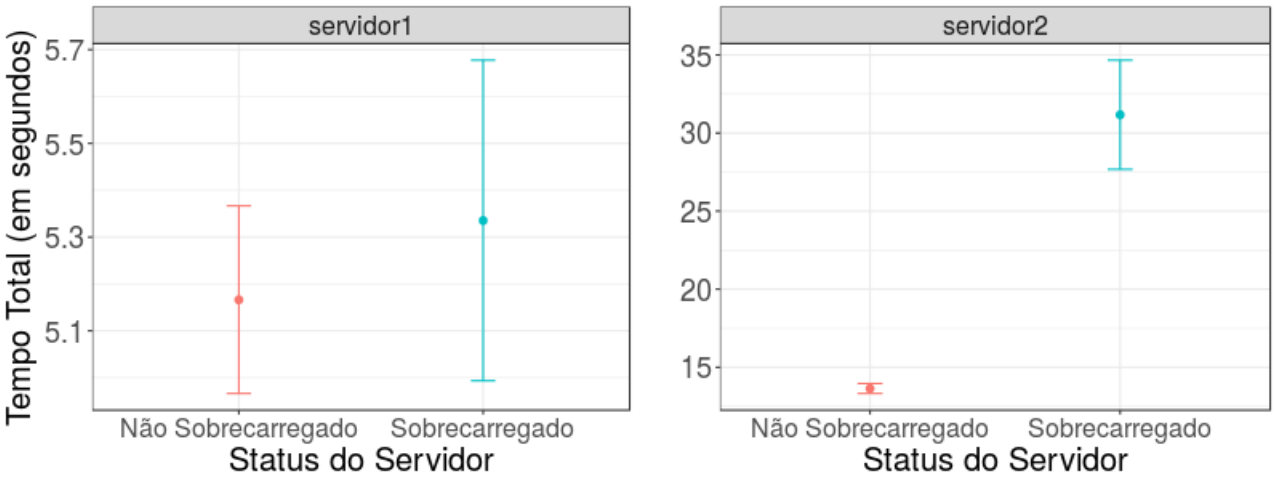


Figura 4.21: Intervalo de Confiança 95% para *SysbenchPerfCPUCap* com CAP inicial em 75% comparando o tempo do *Sysbench* quando os servidores estavam sobrecarregados e no momento que não estavam

Tabela 4.10: Resultados do Teste de*Wilcoxon* para cada Heurística comparando o desempenho (CAP inicial 75%)

| Heurística | Rodada do Balanceador | Servidor sobrecarregado | p-value |
|---|---|---|---|
| *BalanceInstancesOS* | #1 | servidor2 | 5.96e-08 |
| | #2 | servidor1 | 0,0004895 |
| *CPUCapAware* | #1 | servidor2 | 5.96e-08 |
| | #3 | servidor1 | 0,0004297 |
| *SysbenchPerfCPUCap* | #1 | servidor2 | 5.96e-08 |
| | #3 | servidor1 | **0,9368** |

# Capítulo 5

# Conclusão

Neste capítulo apresentamos a conclusão desta dissertação, na Seção 5.1, retomamos qual o problema que se desejou resolver e discorre-se sobre os resultados obitidos no Capítulo 4. Na Seção 5.2 são apontadas limitações deste trabalho e possíveis trabalhos futuros.

## 5.1 Sumário

Este trabalho se ateve ao objetivo de propor e analisar heurísticas, que auxiliassem no balanceamento de carga de servidores físicos em uma infraestrutura de nuvem que se encontra sobrecarregada, de forma a prover melhoria no desempenho dos servidores por meio da realocação de máquinas virtuais por meio da migração. A pergunta de pesquisa foi, se há ganho de desempenho com relação à atividades intensas em CPU quando há servidores sobrecarregados infraestrutura.

Para responder a esta pergunta, foram propostas três diferentes heurísticas e implementadas em um balanceador de carga para se realizar os experimentos. Para o cenários com 75% de CAP apresentados no Capítulo 4 desta dissertação, os resultados encontrados para todas as heurísticas mostram que é possível diminuir a utilização de CPU dos servidores sobrecarregados e apenas no caso da heurística *SysbenchPerfCPUCap* com o servidor *c4-compute12*, onde não foi possível perceber melhoria de desempenho quando o servidor estava sobrecarregado e deixou de estar. Para os outros casos o ganho de desempeno foi observado.

Dentre as três heurísticas propostas, a heurística *SysbenchPerfCPUCap* consegue se comportar bem nos cenários em que as outras heurísticas não são indicadas, conforme explicado

na Secção 3.4

As avaliações para 25% e 50% de CAP são detalhadas no Apêndice A. Para 25% de CAP foi não possível verificar o ganho de desempenho nos servidores apenas no caso da heurísticas *CPUCapAware* com o servidor *c4-compute22* e *SysbenchPerfCPUCap* com o servidor *c4-compute22*, já para 50% de CAP os resultados permitem verificar o ganho de desempenho para todas as heurísticas.

## 5.2 Trabalhos Futuros

Após os resultados que foram obtidos com as heurísticas e a ferramenta desenvolvida nesta dissertação, nos limitamos a considerar apenas CPU como o principal recursos e métricas relacionadas, além disso, a infraestrutura de nuvem utilizada foi OpenStack e a ferramenta consegue lidar apenas com migrações que considerem o compartilhamento de de disco, desta forma podemos elencar alguns trabalhos futuros tais como:

- Criação de heurísticas que considerem métricas de recursos como rede, memória e disco.
- Criação de heurísticas que façam uso de combinação de métricas de diferentes tipos de recursos.
- Implementar suporte para outras infraestruturas de nuvem na ferramenta de balanceamento de carga.
- Avaliar impacto das migrações do balanceador em aplicações reais, as quais tem como principal recurso CPU.
- Implementar suporte para que a ferramenta de balanceamento tenha conhecimento de aplicações para auxiliar na decisão das migrações.

# Bibliografia


[1] Peter Mell and Timothy Grance. Sp 800-145. The NIST Definition of Cloud Computing. Technical report, Gaithersburg, MD, United States, 2011.

[2] Miguel Grinberg. OpenStack Orchestration in Depth, Part IV: Scaling. https://developer.rackspace.com/blog/openstack-orchestration-in-depth-part-4-scaling/, 2015.

[3] Marian Turowski and Alexander Lenk. Vertical Scaling Capability of OpenStack - Survey of Guest Operating Systems, hypervisors, and the cloud management platform. In *ICSOC Workshops*, 2014.

[4] libvirt: Domain XML format. http://libvirt.org/formatdomain.html#elementsCPUTuning.

[5] OpenStack Open Source Cloud Computing Software. https://www.openstack.org/software. Accessado: 01 de Junho 2017.

[6] The World Runs OpenStack. https://www.openstack.org/user-stories/. Acessado: 01 de Junho 2017.

[7] Keystone. https://www.openstack.org/software/releases/ocata/components/keystone. Acessado: 02 de Junho de 2017.

[8] Nova. https://www.openstack.org/software/releases/ocata/components/nova. Acessado: 02 de Junho de 2017.

[9] Monasca. https://www.openstack.org/software/releases/ocata/components/monasca. Acessado: 02 de Junho de 2017.

[10] Rahul Ghosh and Vijay K. Naik. Biting Off Safely More Than You Can Chew: Predictive Analytics for Resource Over-Commit in IaaS Cloud. In *Proceedings of the 2012 IEEE Fifth International Conference on Cloud Computing*, CLOUD '12, pages 25–32, Washington, DC, USA, 2012. IEEE Computer Society.

[11] Charles Reiss, Alexey Tumanov, Gregory R Ganger, Randy H Katz, and Michael A Kozuch. Towards understanding heterogeneous clouds at scale: Google trace analysis. *Intel Science and Technology Center for Cloud Computing, Tech. Rep*, 84, 2012.

[12] Ehsan Arianyan, Hassan Taheri, and Saeed Sharifian. Novel Energy and SLA Efficient Resource Management Heuristics for Consolidation of Virtual Machines in Cloud Data Centers. *Comput. Electr. Eng.*, 47(C):222–240, October 2015.

[13] Saad Mustafa, Babar Nazir, Amir Hayat, Sajjad A Madani, et al. Resource management in cloud computing: Taxonomy, prospects, and challenges. *Computers & Electrical Engineering*, 47:186–203, 2015.

[14] Welcome to Python.org. `https://www.python.org/`. Acessado: 01 de Janeiro de 2017.

[15] OpenStack Identity (Keystone) Client. `https://github.com/openstack/python-keystoneclient/`. Acessado: 01 de Janeiro de 2017.

[16] OpenStack Compute (Nova) Client. `https://github.com/openstack/python-novaclient`. Acessado: 01 de Janeiro de 2017.

[17] Python client for Monasca REST API. `https://github.com/openstack/python-monascaclient`. Acessado: 01 de Janeiro de 2017.

[18] R: The R Project for Statistical Computing. `https://www.r-project.org/`. Acessado: 20 de Fevereiro de 2017.

[19] Ubuntu Manpage: sysbench - A modular, cross-platform and multi-threaded benchmark tool. `http://manpages.ubuntu.com/manpages/xenial/man1/sysbench.1.html`.

[20] Brendan Gregg. *Systems Performance: Enterprise and the Cloud*. Prentice Hall Press, Upper Saddle River, NJ, USA, 1st edition, 2013.

[21] 33.4. Overcommitting Resources. `https://access.redhat.com/documentation/en-US/Red_Hat_Enterprise_Linux/5/html/Virtualization/sect-Virtualization-Tips_and_tricks-Overcommitting_with_KVM.html`.

[22] David Wang. Meeting green computing challenges. In *Electronics Packaging Technology Conference, 2008. EPTC 2008. 10th*, pages 121–126. IEEE, 2008.

[23] Fahimeh Farahnakian, Adnan Ashraf, Tapio Pahikkala, Pasi Liljeberg, Juha Plosila, Ivan Porres, and Hannu Tenhunen. Using ant colony system to consolidate vms for green cloud computing. *IEEE Transactions on Services Computing*, 8(2):187–198, 2015.

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

[25] F. Xu, F. Liu, L. Liu, H. Jin, B. Li, and B. Li. iaware: Making live migration of virtual machines interference-aware in the cloud. *IEEE Transactions on Computers*, 63(12):3012–3025, Dec. 2014.

[26] Hugo Eiji Tibana Carvalho. VOLTAIC : UM SISTEMA DE GERÊNCIA AUTOMATIZADA DE RECURSOS DE NÓS VIRTUAIS PARA COMPUTAÇÃO EM NUVENS. Master's thesis, Universidade Federal do Rio de Janeiro, 2012.

[27] C. P. ALKMIN. Gerenciamento de recursos computacionais em plataformas de Computação em Nuvem. Master's thesis, Instituto de Matemática e Estatística, Universidade de São Paulo, 2014.

[28] SimMyCloud. `https://www.ime.usp.br/~cassiop/simmycloud/`.

[29] Deborah Maria Vieira Magalhães. Uma Estratégia de migração dinâmica de máquinas virtuais para economia de energia em ambientes computacionais virtualizados. Master's thesis, Universidade Federal do Ceará, 2012.

[30] OpenStack User Survey - April 2017. https://www.openstack.org/assets/survey/April2017SurveyReport.pdf.

# Apêndice A

# Experimentos complementares

Este apêndice contêm as análises e resultados para os outros dois cenários de experimentos apresentados que foram citados no Capítulo 4 na Seção 4.3. Para ambos os cenários abaixo, foram avaliadas as métricas que determinam se o servidor está sobrecarregado de acordo com cada heurística, a utilização de CPU dos servidores e o tempo de execução do *Sysbench* para comparação de desempenho. O limiar do *cpu_ratio* foi de **0,5**.

## A.1 Avaliação para CAP inicial em 25%

A Tabela A.1 apresenta o tempo em que as migrações iniciaram em cada heurística para cada execução da mesma, nos gráficos abaixo estes tempos são demarcados pelas linhas verticais tracejadas pretas. É interessante observar que apenas a heurística *BalanceInstancesOS* apresentou dois momentos de migração de VMs, isto se deve ao fato dela não levar em consideração o consumo que as VMs possuem nos servidores.

Tabela A.1: Tempo em que as migrações ocorreram (CAP inicial 25%)

| | **Execução** | | | | |
|---|---|---|---|---|---|
| **Heurísticas** | 1 | 2 | 3 | 4 | 5 |
| *BalanceInstancesOS* | 4 e 28 | 3 e 27 | 8 e 34 | 4 e 28 | 6 e 30 |
| *CPUCapAware* | 49 | 48 | 54 | 50 | 50 |
| *SysbenchPerfCPUCap* | 52 | 53 | 53 | 50 | 50 |

Na Figura A.1 podemos observar o *host_used_cpu_ratio* para a heurística *BalanceIns-*

*tancesOS* nas cinco execuções do balanceador, podemos identificar que o limiar do *cpu_ratio* é extrapolado levando o balanceador a tomar decisões de migração durante a primeira e segunda rodada, na primeira inicialmente o servidor2 já se encontra sobrecarregado, na segunda a criação de VMs no servidor1 o torna sobrecarregado. Também é importante observar que após a primeira rodada do balanceador, o servidor2 não apresenta métricas, isto ocorre devido a ação de remover VMs que deixa-o vazio.

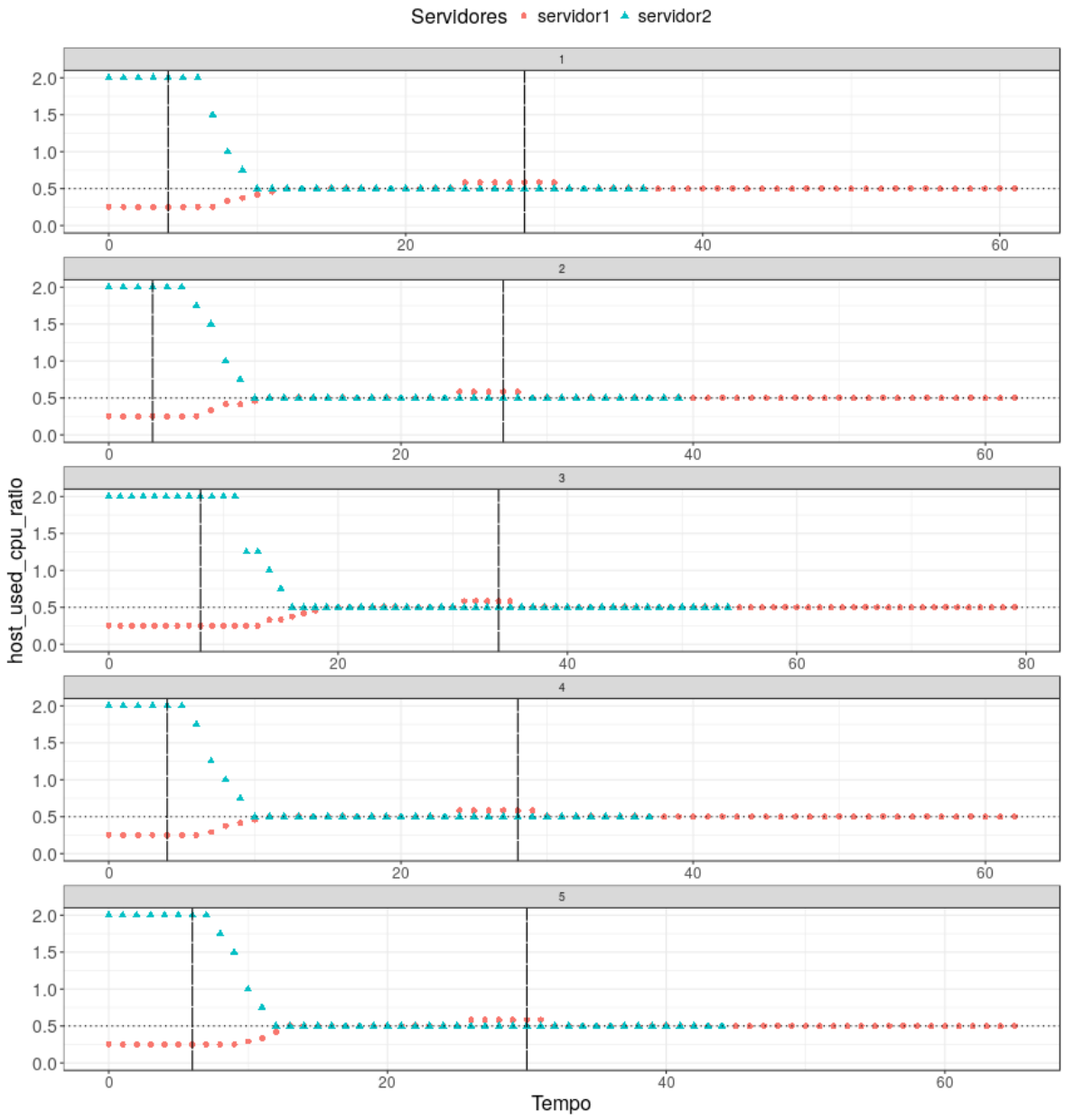


Figura A.1: *host_used_cpu_ratio* dos servidores durante as execuções com a Heurística *BalanceInstancesOS* com CAP inicial em 25%

As Figuras A.2 e A.3 representam o *host_total_consumption* para as heurísticas *CPUCapAware* e *SysbenchPerfCPUCap* respectivamente, o qual é responsável por desencadear as migrações durante a terceira rodada do balanceador de carga, já que antes houve o aumento do CAP para 100% em todas as VMs dos servidores, causando assim a sobrecarga no servidor2.

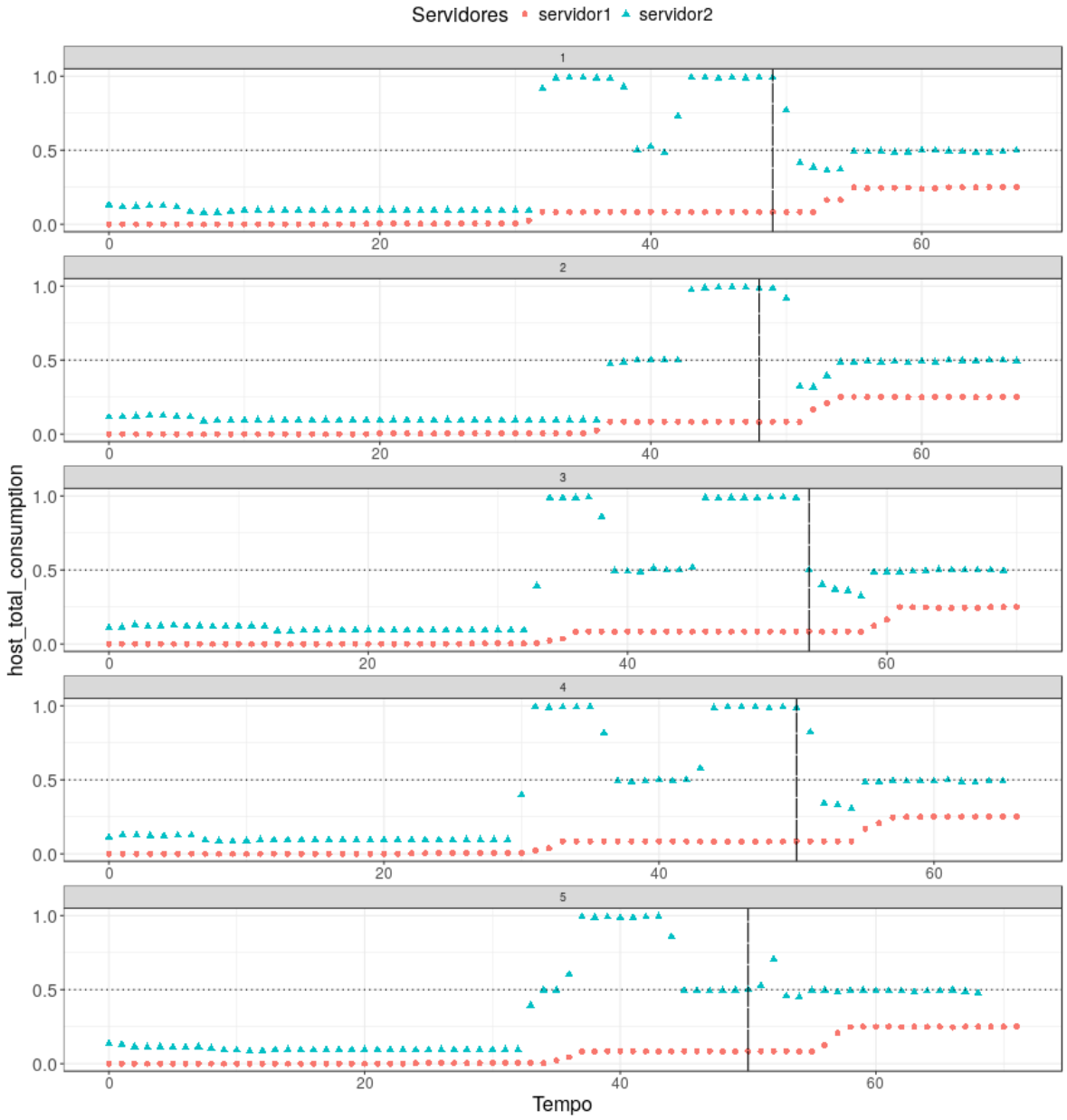


Figura A.2: *host_total_consumption* dos servidores durante as execuções com a Heurística *CPUCapAware* com CAP inicial em 25%

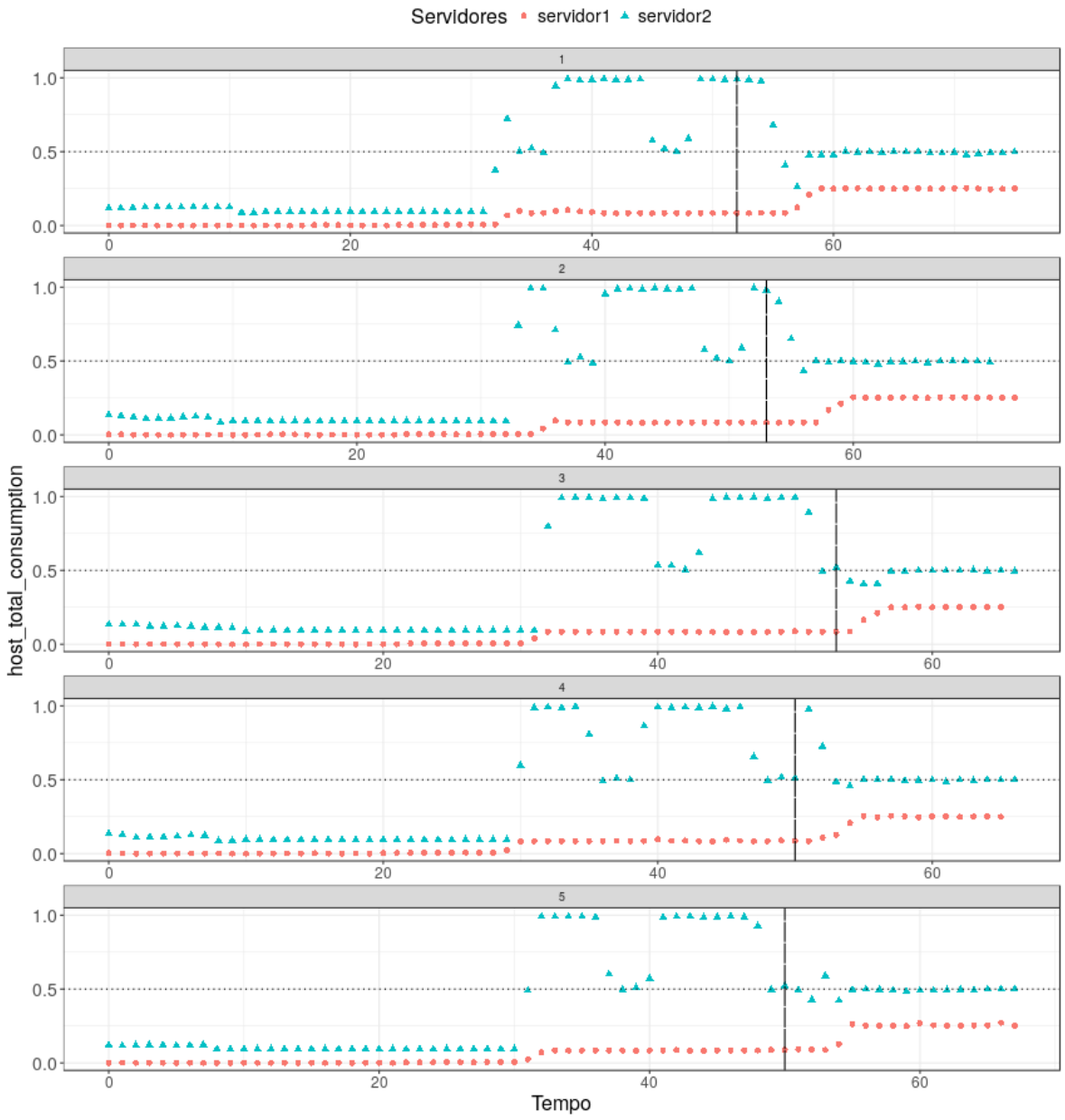


Figura A.3: *host_total_consumption* dos servidores durante as execuções com a Heurística *SysbenchPerfCPUCap* com CAP inicial em 25%

As variações que ocorrem são devido ao *Monasca* não ter provido métricas naquele tempo para algumas VMs. Os gráficos para ambas possuem o mesmo padrão visto que as VMs migradas em ambas as heurísticas foram as que, apresentavam maior consumo.

Nas Figuras A.4 e A.5 podemos avaliar o *host_total_cap* para as heurísticas *CPU-CapAware* e *SysbenchPerfCPUCap* respectivamente, que deixa mais claro que o servidor2

realmente estava sobrecarregado já que não leva em consideração a utilização das VMs.

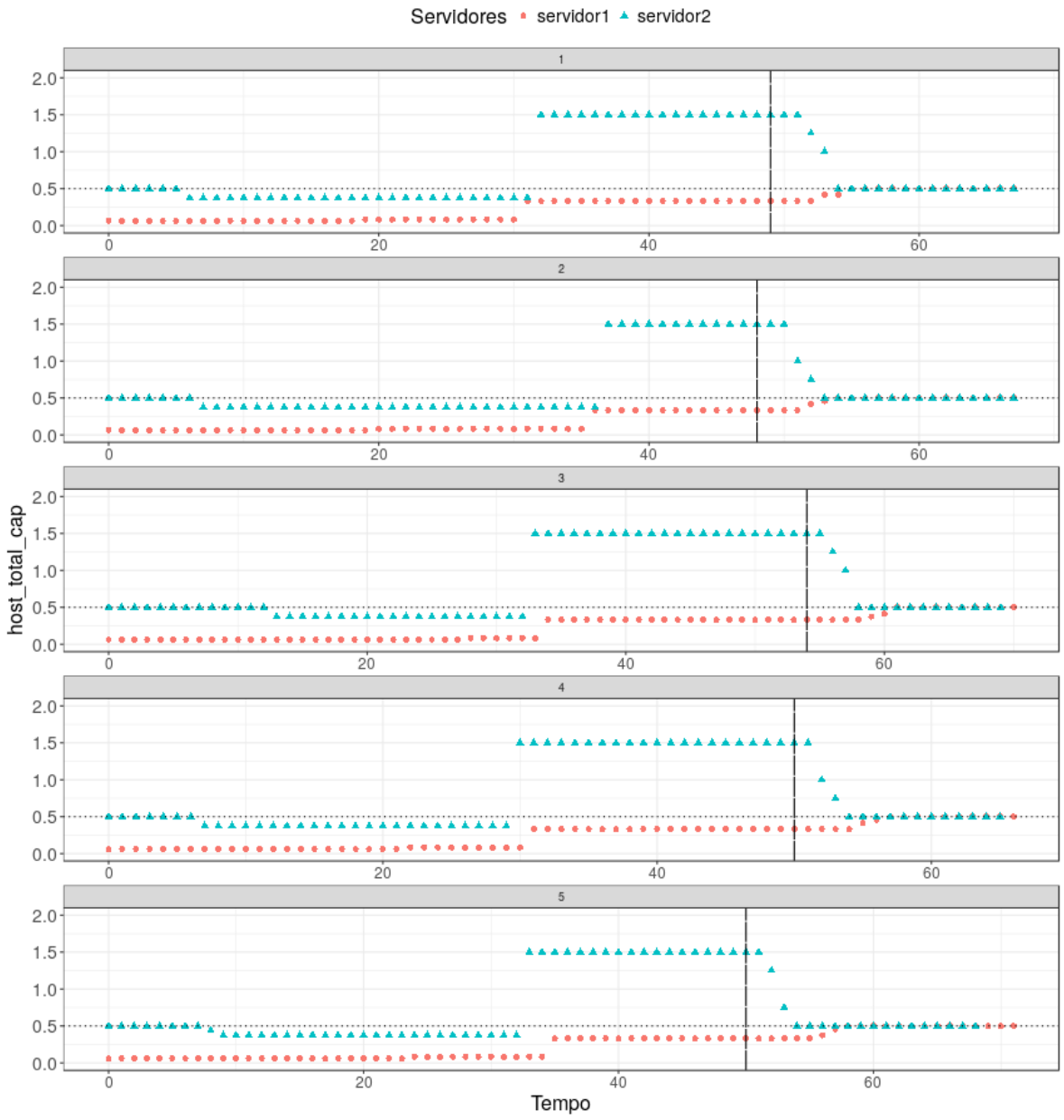


Figura A.4: *host_total_cap* dos servidores durante as execuções com a Heurística *CPUCapAware* com CAP inicial em 25%

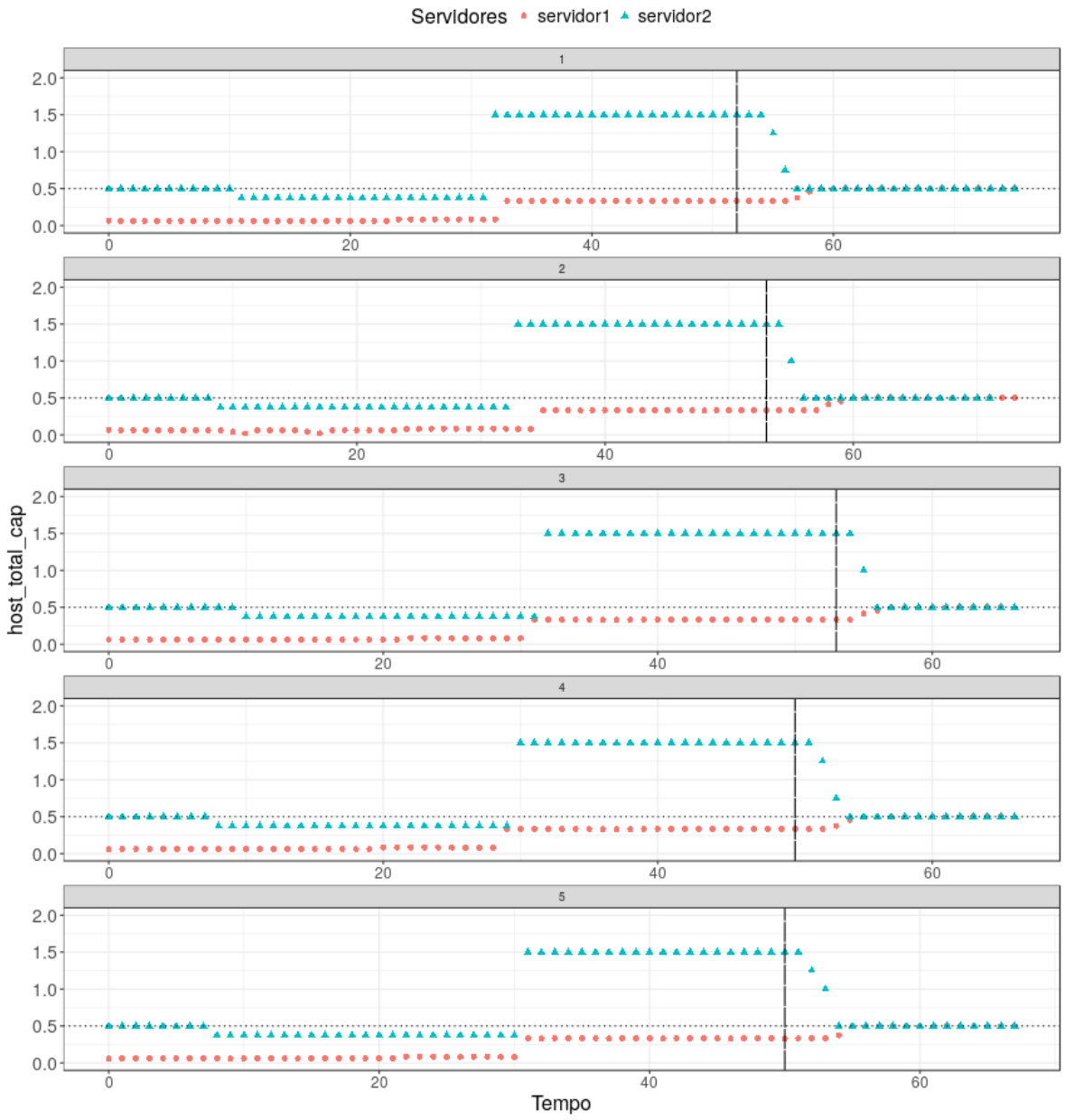


Figura A.5: *host_total_cap* dos servidores durante as execuções com a Heurística *Sysbench-PerfCPUCap* com CAP inicial em 25%

A utilização de CPU dos servidores é apresentada nas Figuras A.6 a A.8, nas quais a linha vermelha e azul representam respectivamente os servidores servidor2 e servidor1.

O padrão que pode ser observado para a heurística *BalanceInstancesOS* na Figura A.6 é diminuição da utilização no servidor servidor2 após a primeira rodada do balanceador, e os picos de utilização que ocorrem entre as duas linhas verticais e após a última linha vertical

representam a execução do *Sysbench*. Para o servidor servidor1 podemos observar que há um crescimento gradual na sua utilização devido as migrações que ocorreram e a mudança de CAP para 100% que ocorre após a última linha vertical.

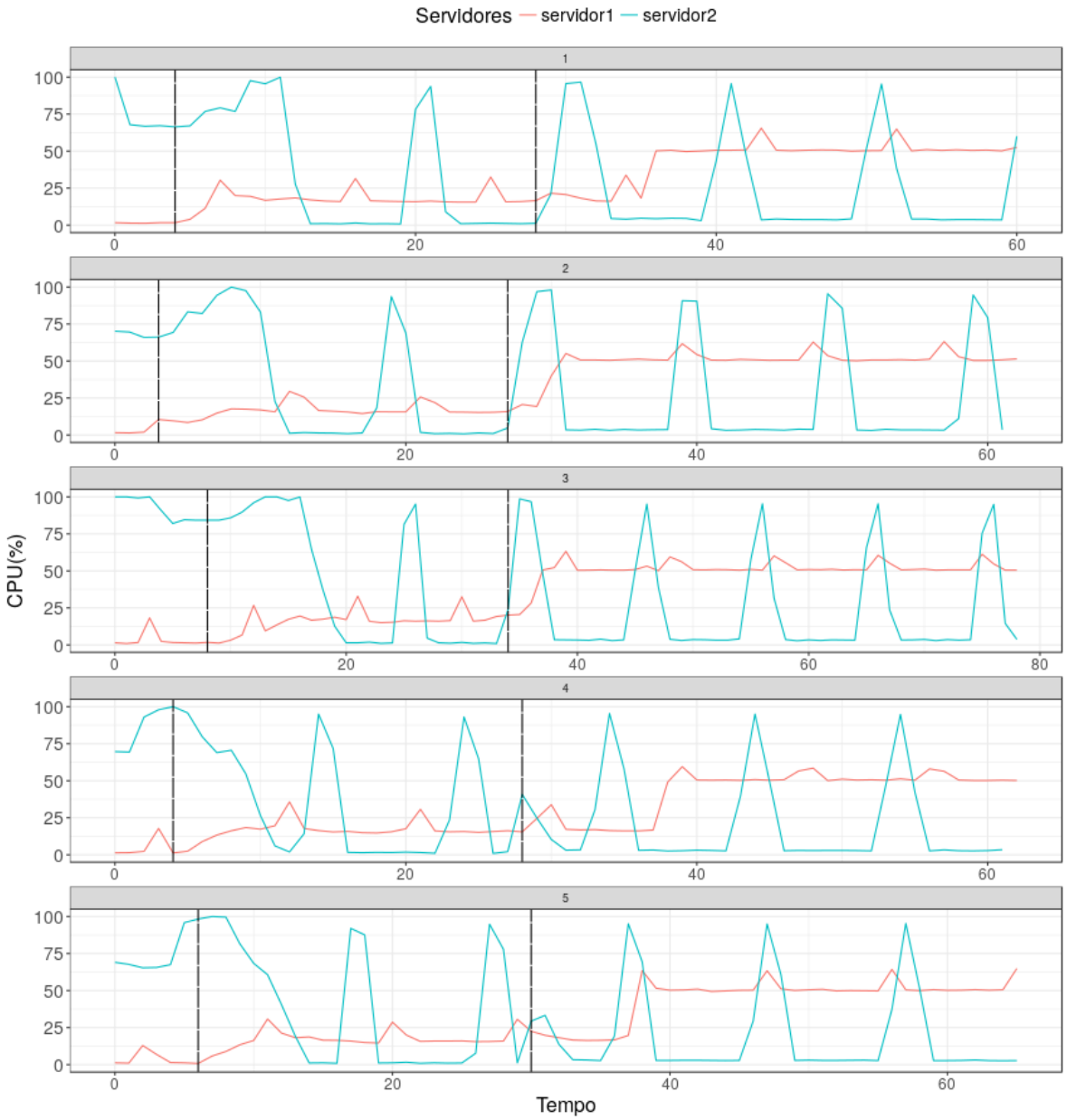


Figura A.6: Utilização de CPU dos servidores durante as execuções com a Heurística *BalanceInstancesOS* com CAP inicial em 25%

Para as outras heurísticas, *CPUCapAware* e *SysbenchPerfCPUCap*, podemos observar a utilização de CPU nas Figuras A.7 e A.8 respectivamente, ambas apresentam o mesmo pa-

drão de oscilações na utilização de ambos os servidores causadas pela execução do *Sysbench*, quando ocorre a mudança do CAP para 100% a utilização do servidor2 se mantém constante e durante a terceira rodada do balanceador há a migração de VMs para o servidor servidor1, ocasionando uma queda na utilização de servidor2 e um aumento em servidor1.

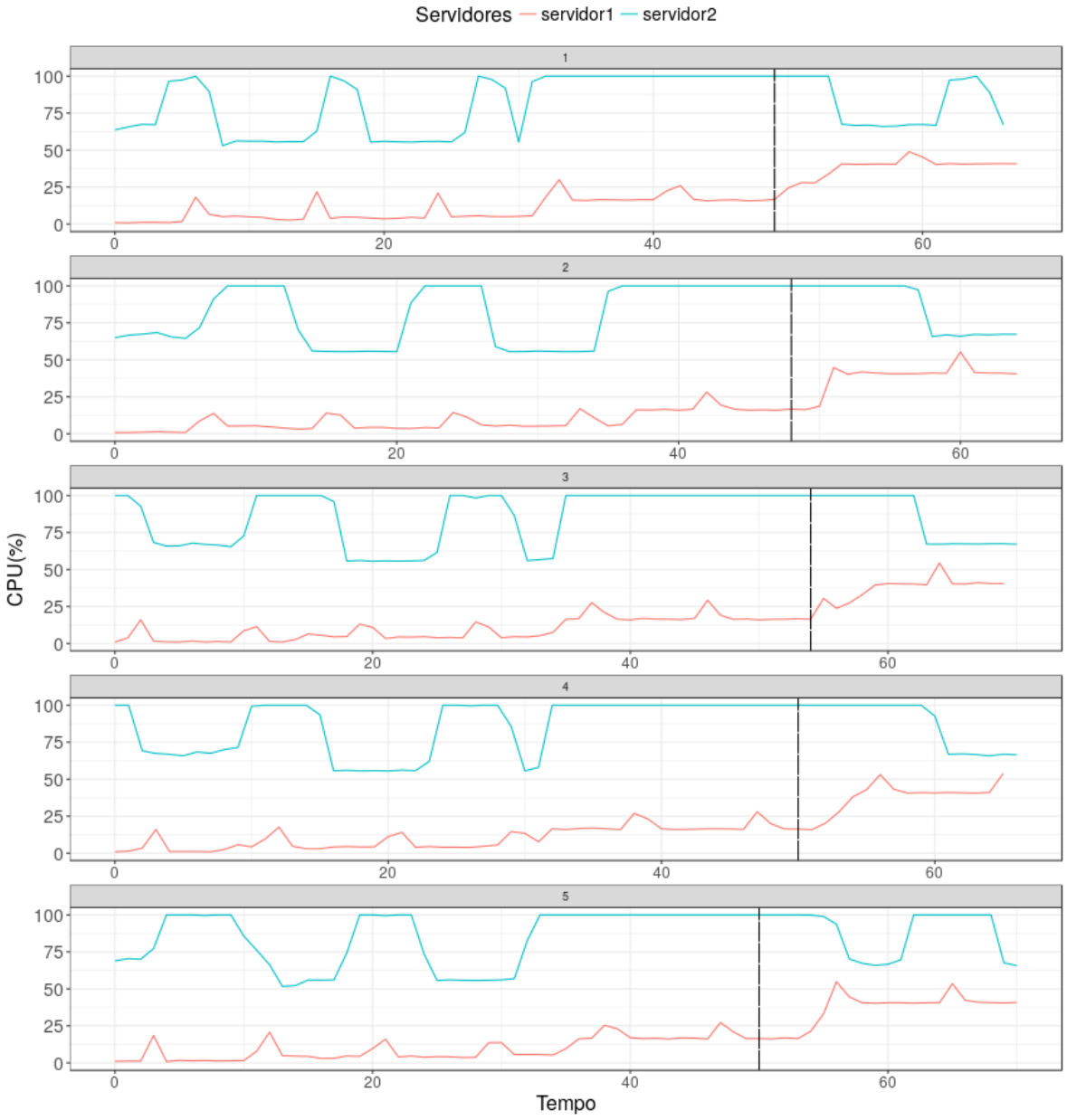


Figura A.7: Utilização de CPU dos servidores durante as execuções com a Heurística *CPU-CapAware* com CAP inicial em 25%

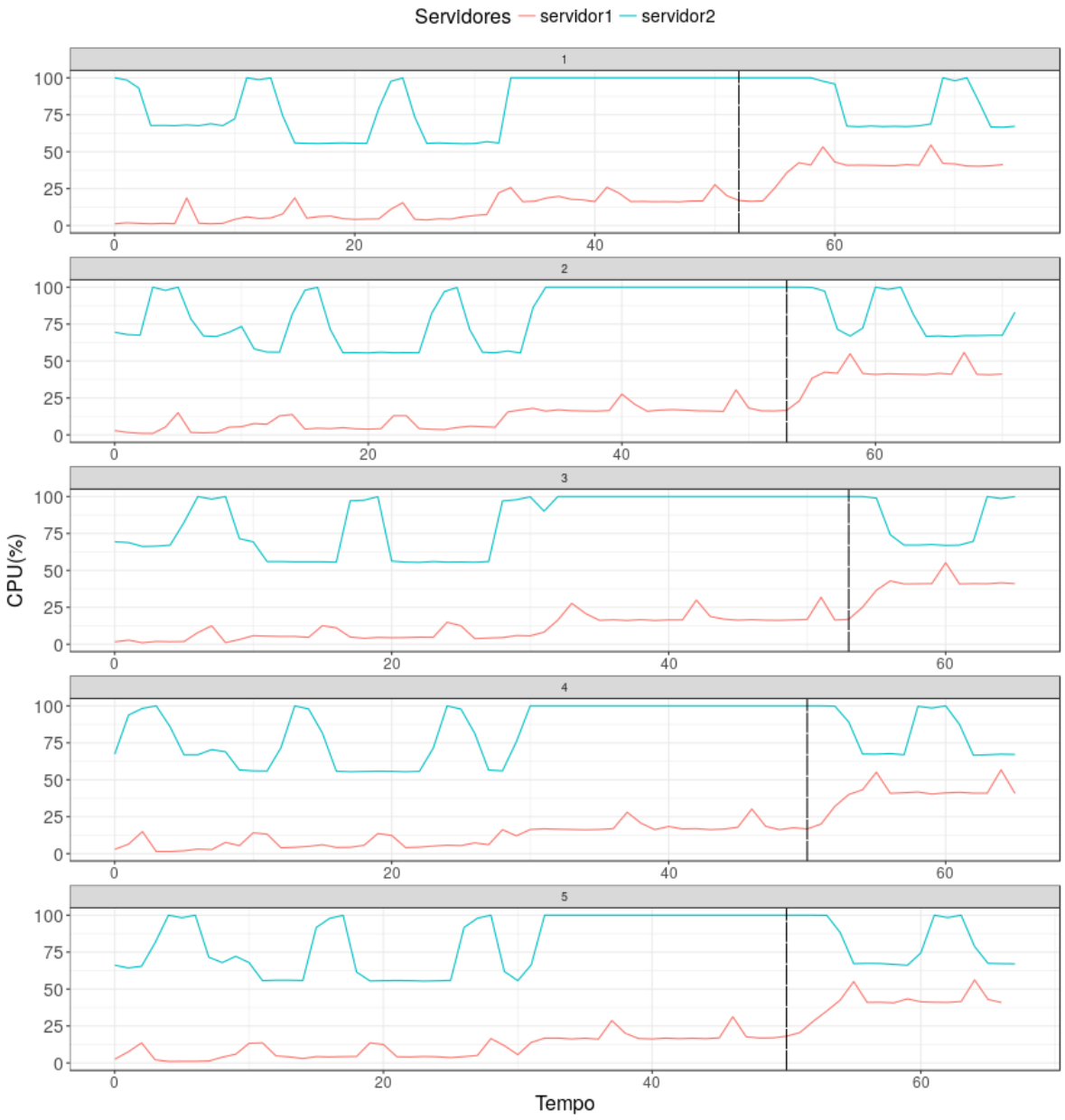


Figura A.8: Utilização de CPU dos servidores durante as execuções com a Heurística *SysbenchPerfCPUCap* com CAP inicial em 25%

Para avaliar se houve melhoria de desempenho nos servidores inicialmente verificamos a normalidade dos dados do tempo de execução para assim pode escolher qual teste estatístico deveria ser utilizado e os intervalos de confiança. Os resultados do teste de normalidade de *Shapiro-Wilk* se encontram na Tabela A.2, dentre os 4 possíveis cenários dois apresentaram *p-value* maior que **0,05** indicando que são normalmente distribuídos, para estes foi utilizado

o teste *t-student* e para os outros o teste de *Wilcoxon*.

Tabela A.2: Resultados do Teste de Normalidade *Shapiro-Wilk* para o tempo de execução do *Sysbench* de acordo com as heurísticas quando houve migrações (CAP inicial 25%)

| Heurística | Rodada do Balanceador | Servidor/Condição | *p-value* |
|---|---|---|---|
| *BalanceInstancesOS* | #1 | *servidor2/Sobrecarregado* | **0,1749** |
| | | *servidor2/Não Sobrecarregado* | **0,1614** |
| | #2 | *servidor1/Sobrecarregado* | 5,2e-06 |
| | | *servidor1/Não Sobrecarregado* | **0,01141** |
| *CPUCapAware* | #3 | *servidor2/Sobrecarregado* | **0,32** |
| | | *servidor2/Não Sobrecarregado* | **0,2173** |
| *SysbenchPerfCPUCap* | #3 | *servidor2/Sobrecarregado* | 3,371e-06 |
| | | *servidor2/Não Sobrecarregado* | 3,806e-06 |

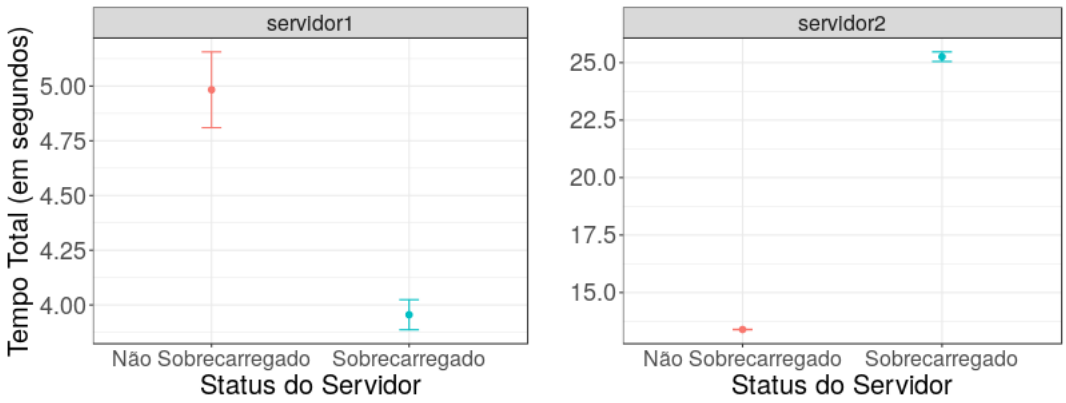


Figura A.9: Intervalo de Confiança 95% para *BalanceInstancesOS* com CAP inicial em 25% comparando o tempo do *Sysbench* quando os servidores estavam sobrecarregados e quando não estavam

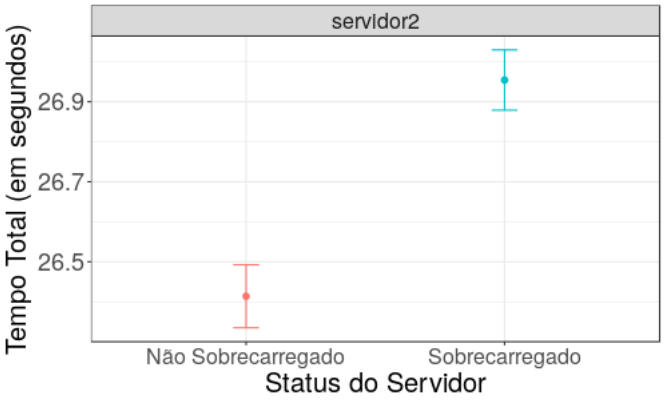


Figura A.10: Intervalo de Confiança 95% para *CPUCapAware* com CAP inicial em 25% comparando o tempo do *Sysbench* quando os servidores estavam sobrecarregados e quando não estavam

As Figuras A.9 a A.11 apresentam os intervalos de confiança para as heurísticas *BalanceInstancesOS*, *CPUCapAware* e *SysbenchPerfCPUCap* respectivamente, neles apresentamos os intervalos de quando cada servidor estava sobrecarregado e após não estar mais sobrecarregado.

É interessante observar que para a heurística *BalanceInstancesOS* os intervalos de confiança na Figura A.9 há uma diferença perceptível nas escalas dos valores de quando o servidor2 estava sobrecarregado e quando não estava mais, já para servidor1 esta diferença existe porém é menor que 1, sendo assim não há sobreposição dos intervalos levando a crer que há diferença de desempenho. O intervalo de confiança para o servidor2 com a heurística *CPUCapAware* na Figura A.10 mostra que os intervalos são bem próximos mas não há sobreposição. Analisando o intervalo da heurística *SysbenchPerfCPUCap* na Figura A.11, podemos observar que há uma sobreposição na qual a média está inclusa indicando que não há diferença para média do tempo *Sysbench*.

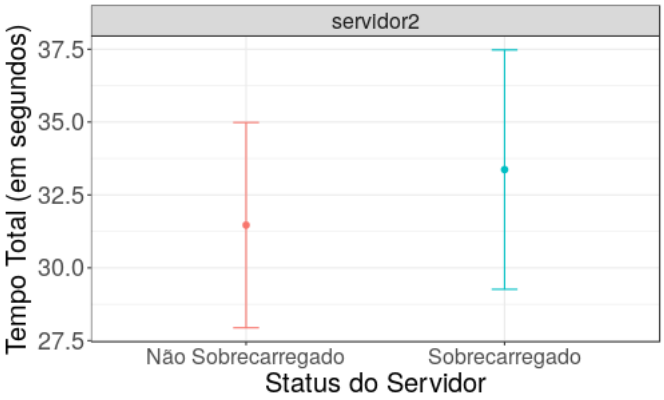


Figura A.11: Intervalo de Confiança 95% para *SysbenchPerfCPUCap* com CAP inicial em 25% comparando o tempo do *Sysbench* quando os servidores estavam sobrecarregados e quando não estavam

Na Tabela A.3 apresentamos os *p-value* para cada cenário e qual o teste que foi aplicado. Os resultados condizem com as avaliações para os intervalos de confiança, visto que, o cenário da heurística *SysbenchPerfCPUCap* foi o único que não refutou a hipótese nula já que apresentou *p-value* igual a 0,22 levando a crer a que vem da mesma distribuição segundo o teste de *Wilcoxon*.

Tabela A.3: Resultados do Testes de Hipótese para cada Heurística comparando o desempenho (CAP inicial 25%)

| Heurística | Teste | Rodada do Balanceador | Servidor sobrecarregado | *p-value* |
|---|---|---|---|---|
| *BalanceInstancesOS* | *t student* | #1 | servidor2 | 2,2e-16 |
| | *Wilcoxon* | #2 | servidor1 | 5,96e-08 |
| *CPUCapAware* | *t student* | #3 | servidor2 | 6,696e-11 |
| *SysbenchPerfCPUCap* | *Wilcoxon* | #3 | servidor2 | **0,22** |

## A.2 Avaliação para CAP inicial em 50%

A Tabela A.4 apresenta o tempo em que as migrações iniciaram em cada heurística para cada execução da mesma, nos gráficos abaixo estes tempos são demarcados pelas linhas verticais tracejadas pretas. É importante ressaltar que apenas a heurística *BalanceInstancesOS* apresenta dois momentos de migração de VMs devido ao fato dela não levar em consideração o consumo que as VMs possuem nos servidores.

Tabela A.4: Tempo em que as migrações ocorreram (CAP inicial 50%)

| | **Execução** | | | | |
|---|---|---|---|---|---|
| **Heurísticas** | 1 | 2 | 3 | 4 | 5 |
| *BalanceInstancesOS* | 6 e 31 | 4 e 28 | 3 e 28 | 5 e 29 | 4 e 29 |
| *CPUCapAware* | 5 | 4 | 6 | 4 | 6 |
| *SysbenchPerfCPUCap* | 6 | 5 | 5 | 6 | 5 |

Na Figura A.12 podemos observar o *host_used_cpu_ratio* para a heurística *BalanceInstancesOS* nas cinco execuções do balanceador, podemos identificar que o limiar do *cpu_ratio* é extrapolado levando o balanceador a tomar decisões de migração durante a primeira e segunda rodada, na primeira inicialmente o servidor2 já se encontra sobrecarregado devido as configurações iniciais do experimento, na segunda rodada a criação de VMs no servidor1 que ocorre antes o torna sobrecarregado. Também é importante observar que após a primeira rodada do balanceador, o servidor2 não apresenta métricas, isto ocorre devido a ação de remover VMs que deixa-o vazio.

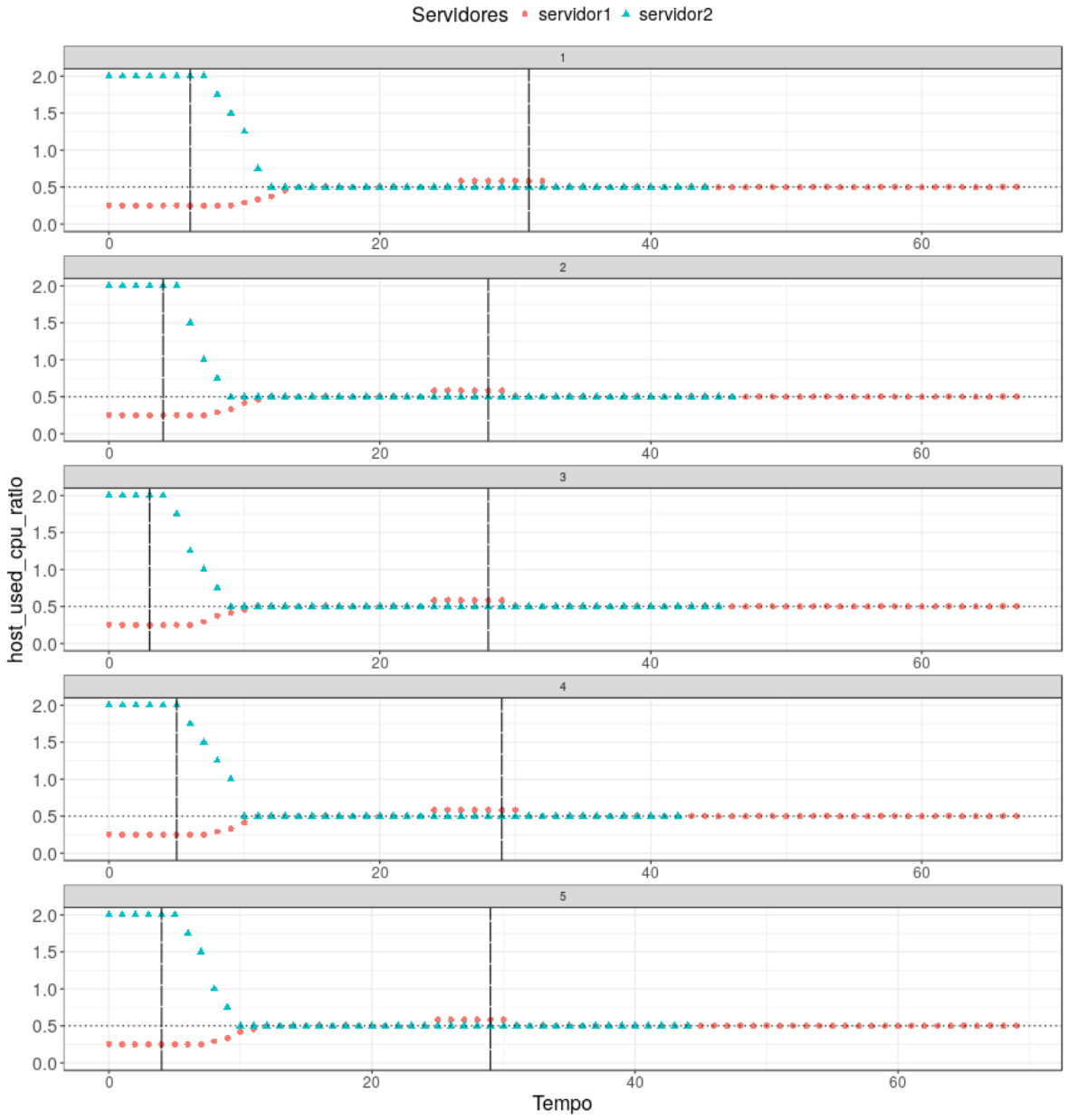


Figura A.12: *host_used_cpu_ratio* dos servidores durante as com *BalanceInstancesOS*

As Figuras A.13 e A.14 representam o *host_total_consumption* para as heurísticas *CPU-CapAware* e *SysbenchPerfCPUCap* respectivamente, porém não considerando-a não se faz necessário a execução de migrações visto que o limiar de 0,5 de *cpu_ratio* não foi extrapolado. Podemos observar o decrescimento do *host_total_consumption* que indica a migração das VMs afim deixar o servidor2 sem estar sobrecarregado. Ao efetuar a mudança de CAP para 100% podemos observar também o aumento do *host_total_consumption* a partir do

meio de cada gráfico de execução.

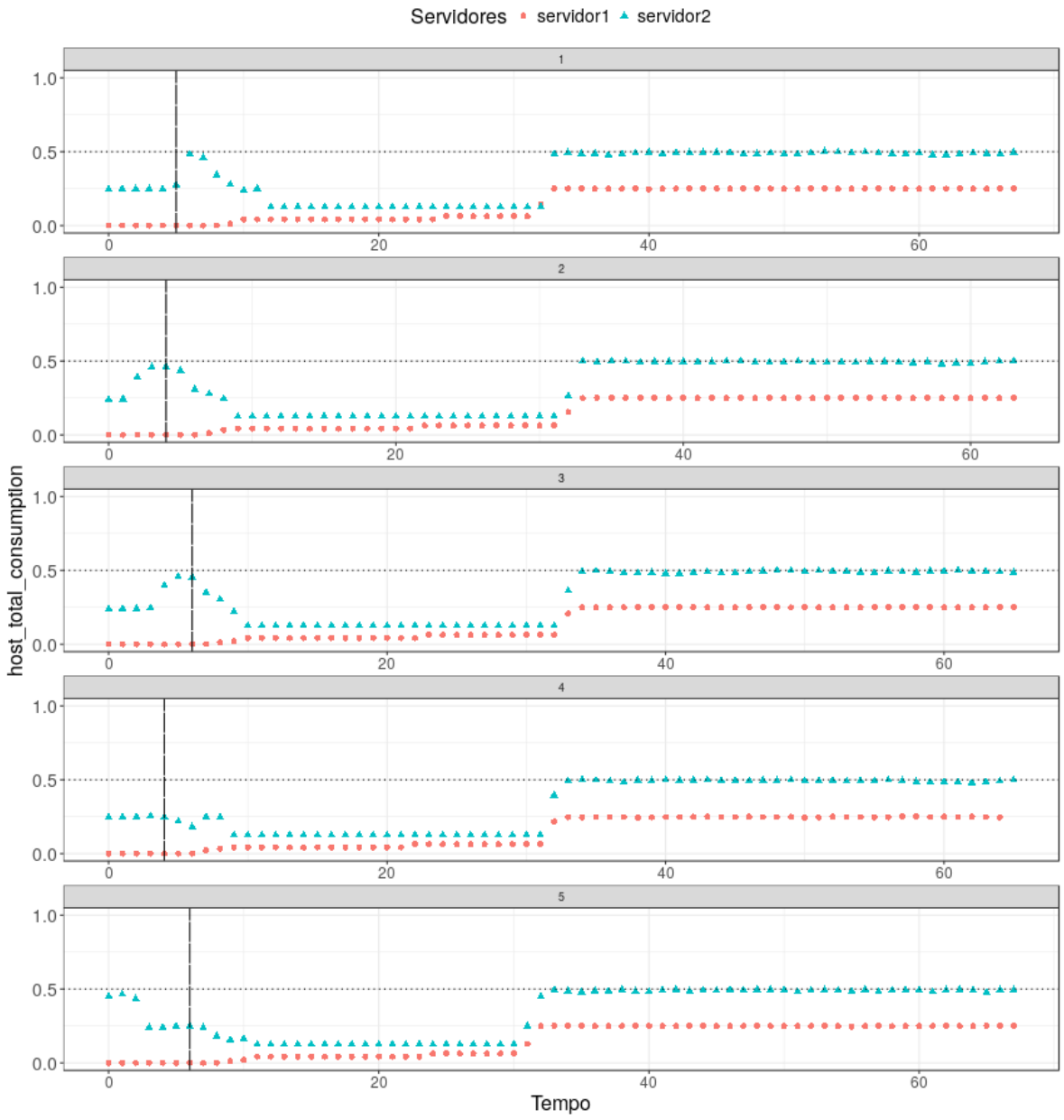


Figura A.13: *host_total_consumption* dos servidores durante as com *CPUCapAware*

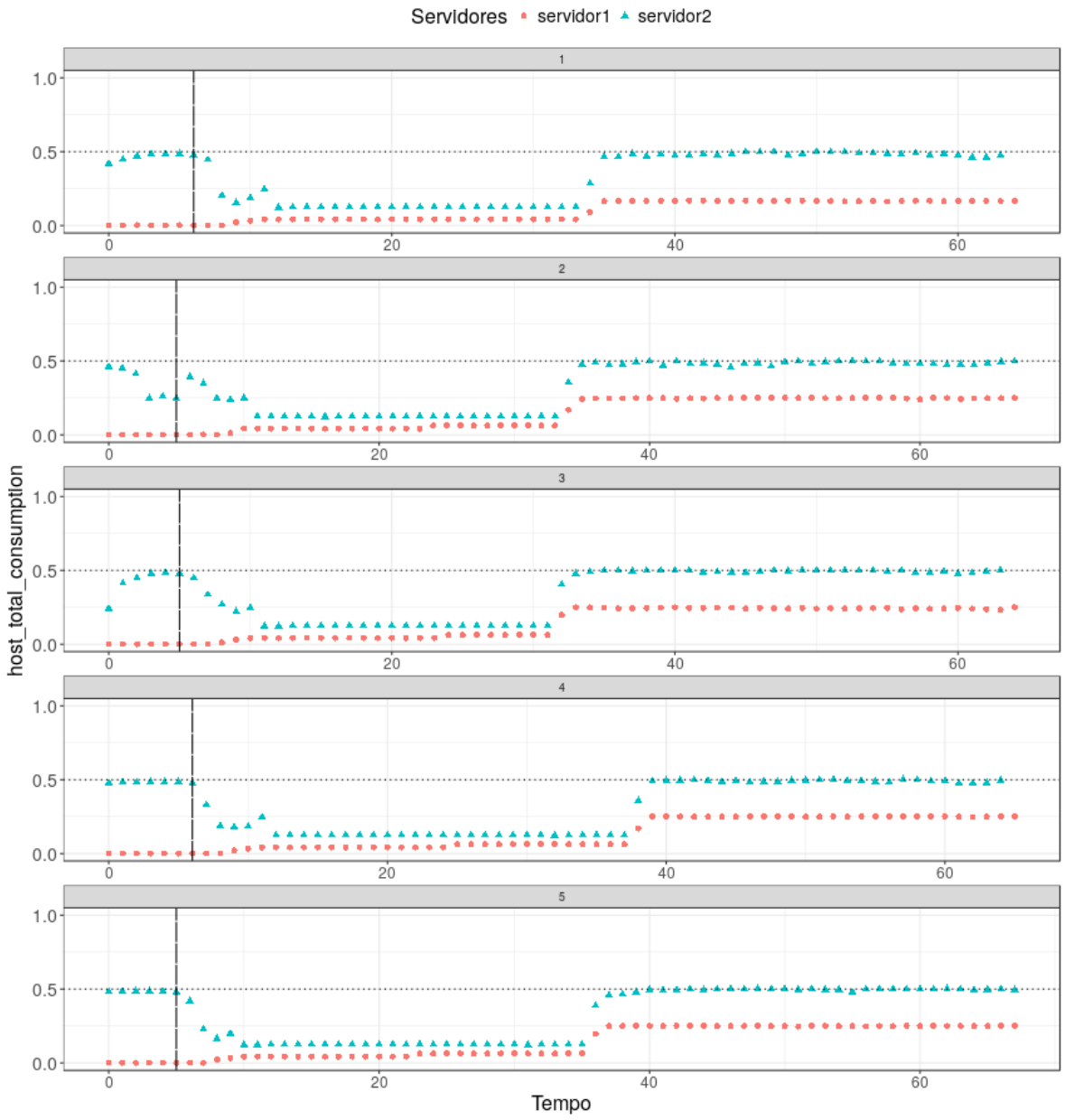


Figura A.14: *host_total_consumption* dos servidores durante as com *SysbenchPerfCPUCap*

O *host_total_cap* das heurísticas *CPUCapAware* e *SysbenchPerfCPUCap* pode ser visto respectivamente nas Figuras A.15 e A.16, nelas podemos observar que o limite de *cpu_ratio* realmente foi violado e que após a primeira rodada ter efetuado as migrações das VMs do servidor2 para servidor1 o limiar de 0,5 não foi mais violado.

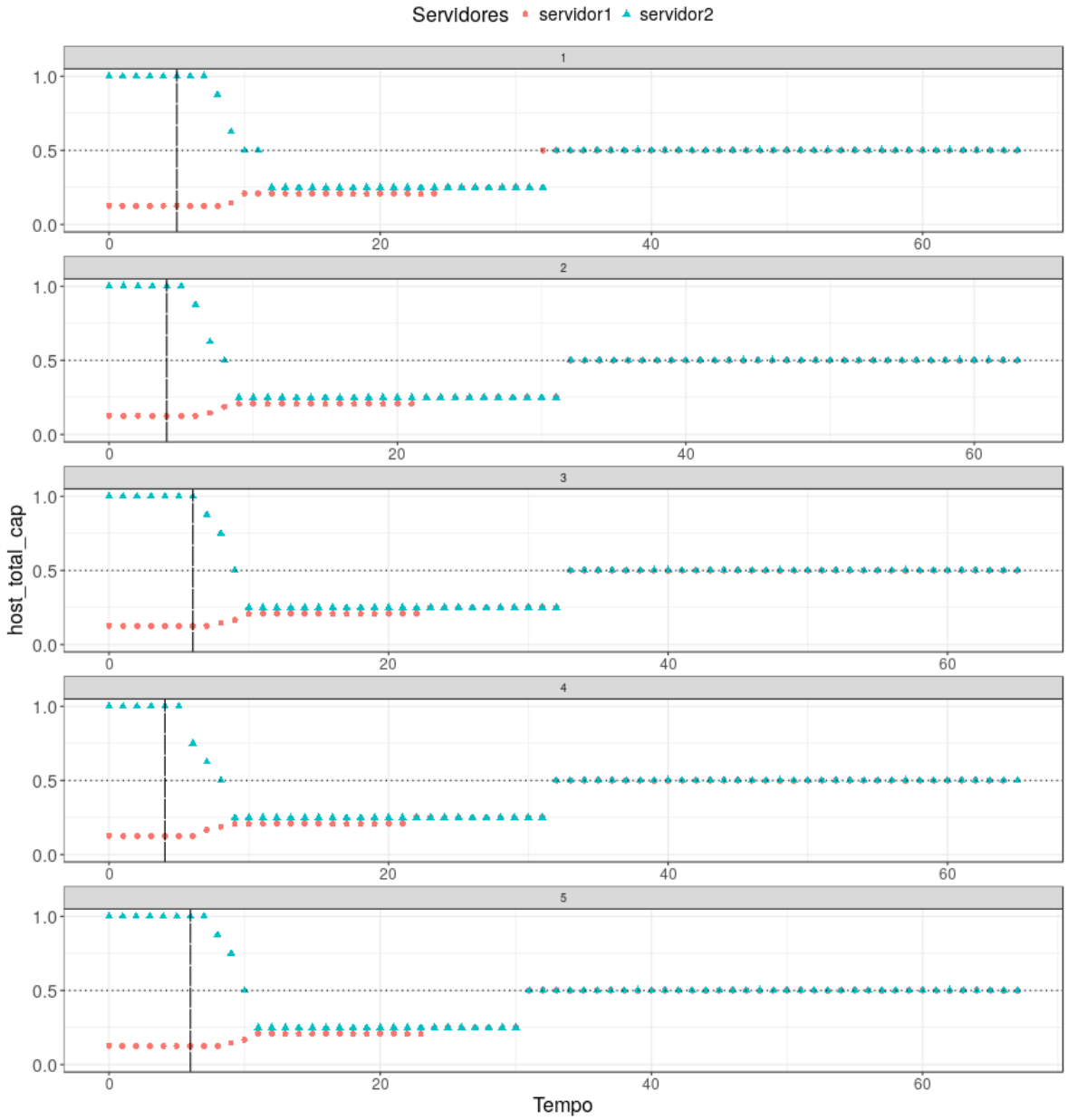


Figura A.15: *host_total_cap* dos servidores durante as com *CPUCapAware*

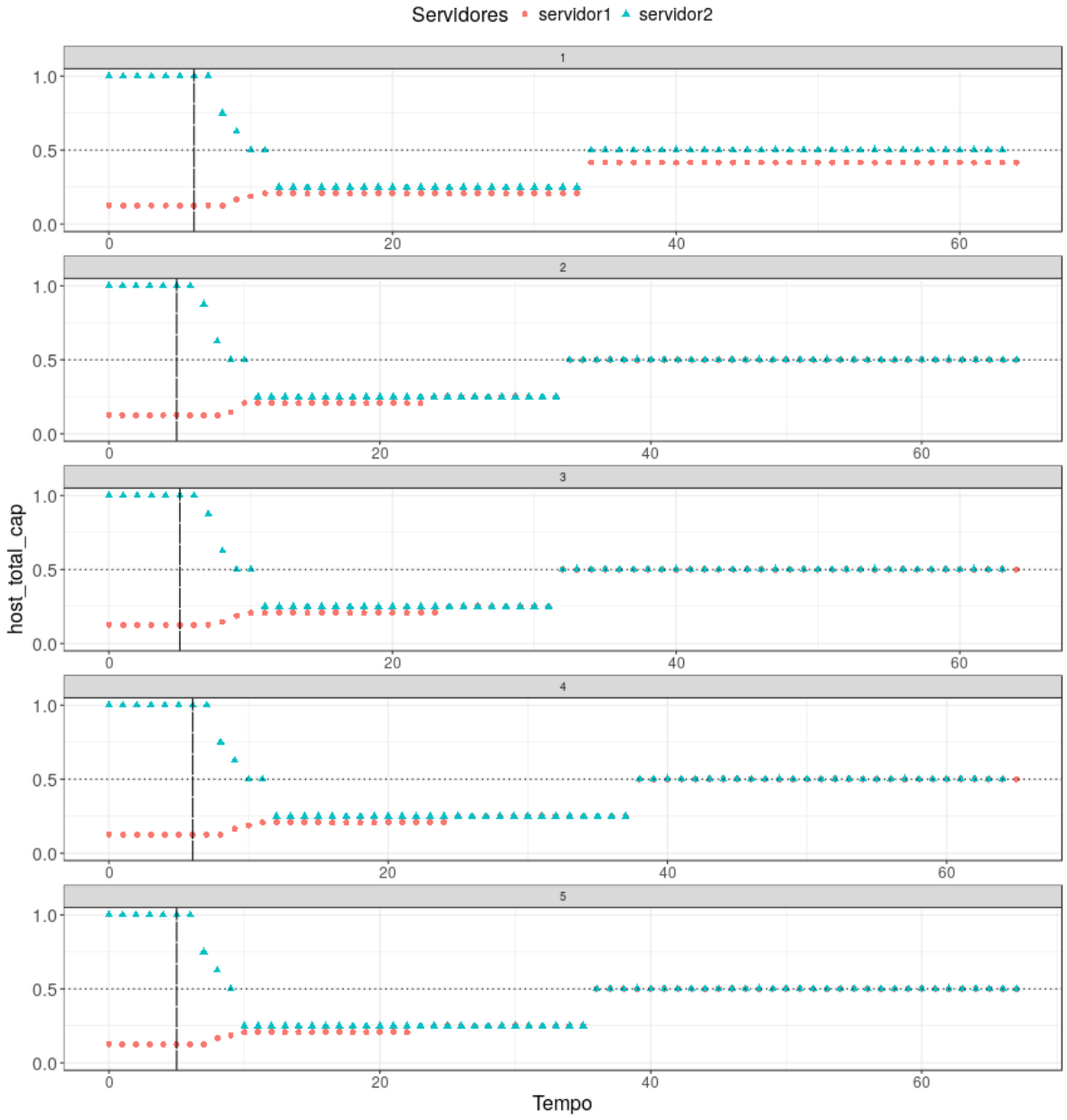


Figura A.16: *host_total_cap* dos servidores durante as com *SysbenchPerfCPUCap*

A utilização da CPU dos servidores foi coletada e podemos ver os dados nas Figuras A.17 a A.19. O padrão comum que podemos ver nos gráficos é a diminuição da utilização no servidor2 que está inicialmente sobrecarregado quando o balanceador de carga inicia sua execução e o aumento no servidor1 que está recebendo as VMs que foram migradas. A mudança de CAP para 100% afeta a utilização de todos os servidores, podemos observar este comportamento da metade em diante em cada gráfico, os picos para que encontramos para

os servidores são devidos a execução do *Sysbench* para coleta de métricas de desempenho.

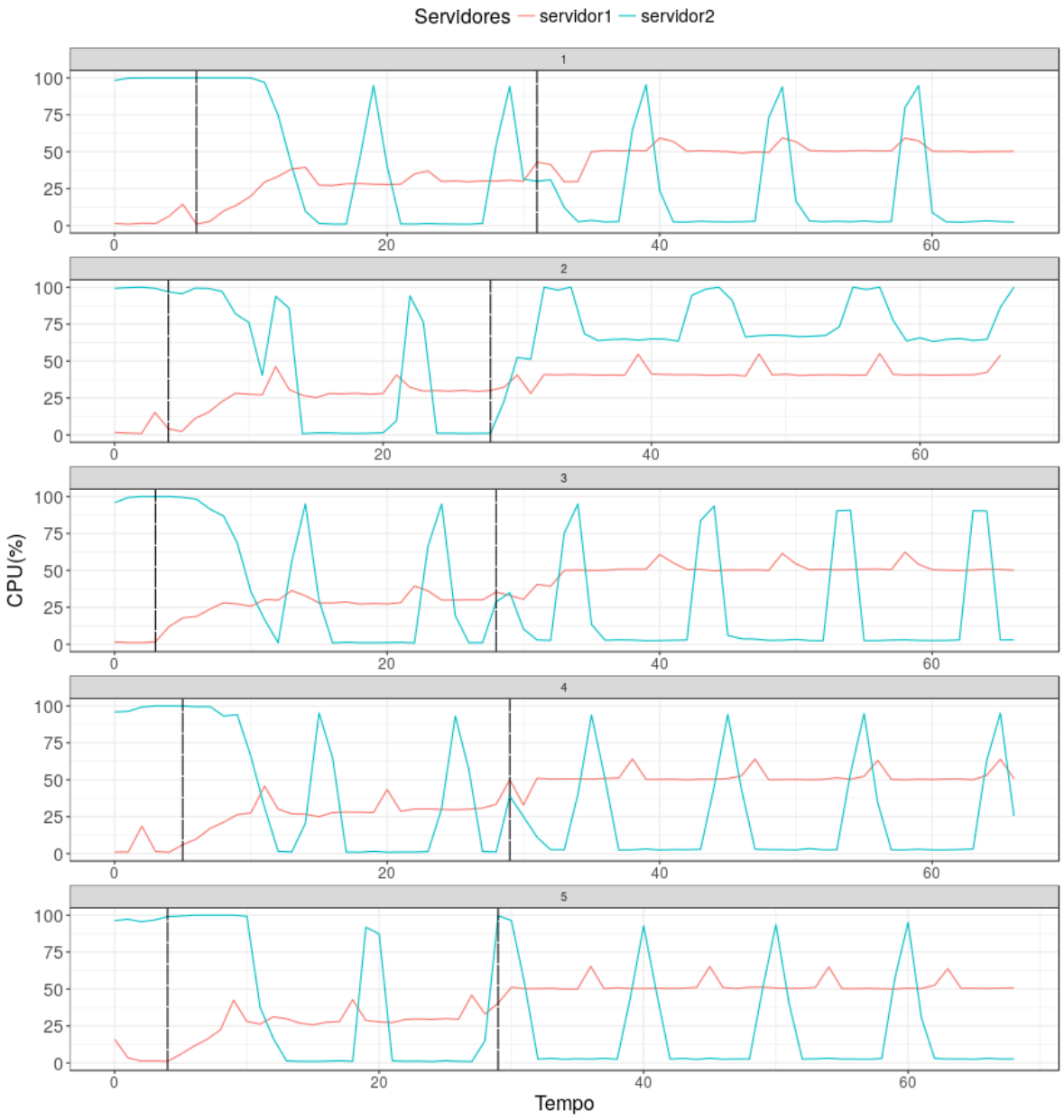


Figura A.17: Utilização de CPU dos servidores durante as com *BalanceInstancesOS*

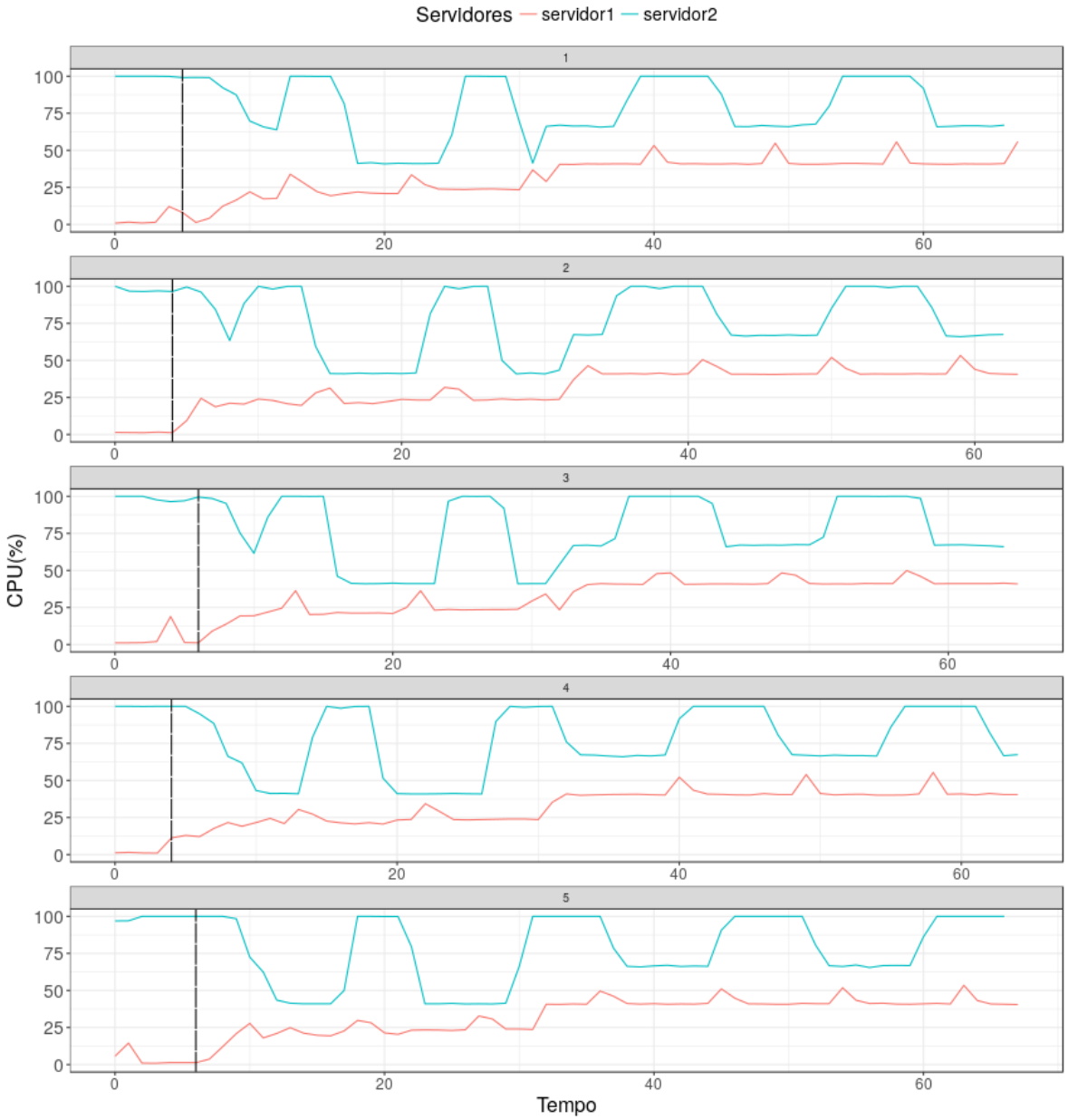


Figura A.18: Utilização de CPU dos servidores durante as com *CPUCapAware*

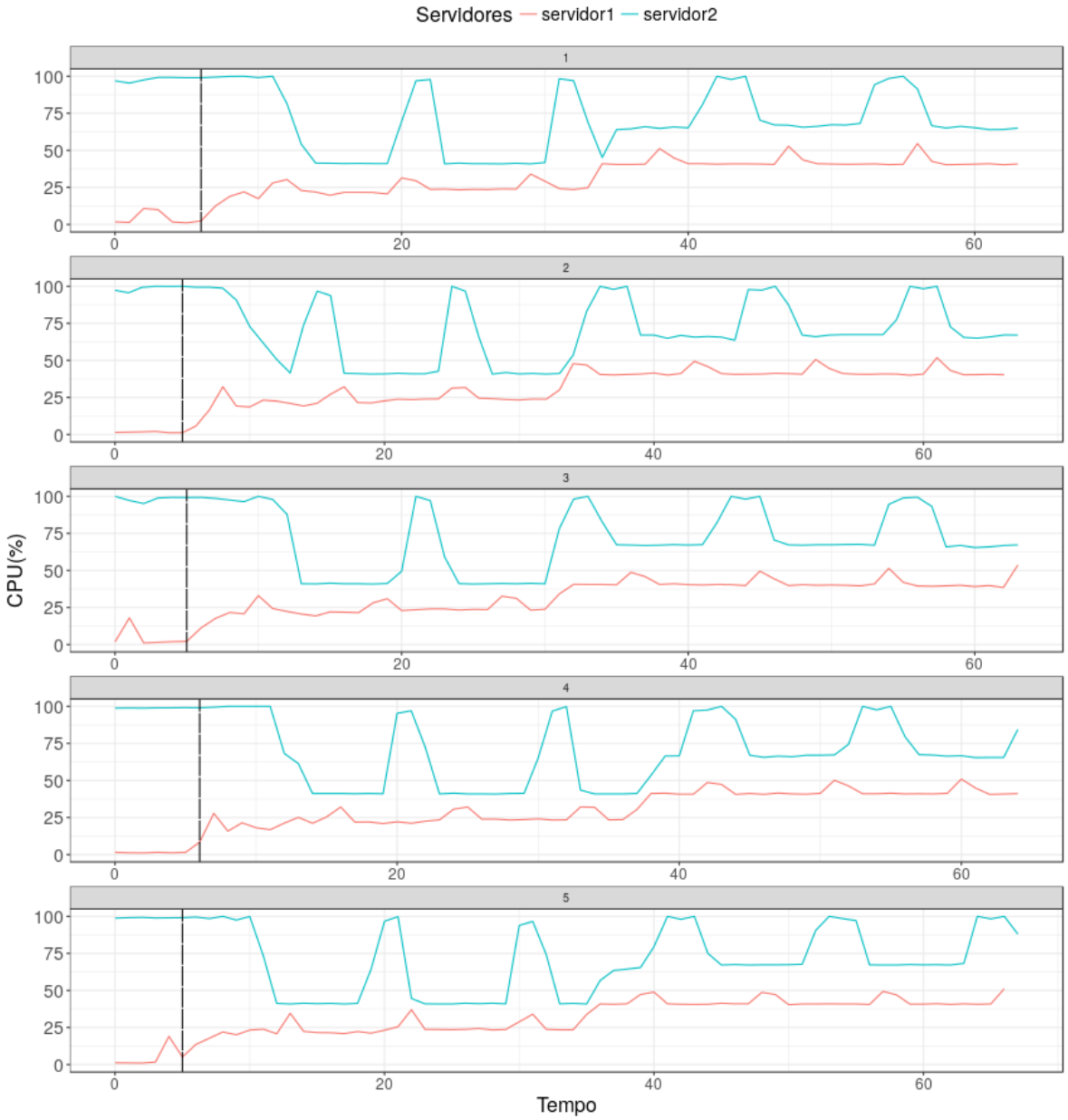


Figura A.19: Utilização de CPU dos servidores durante as com *SysbenchPerfCPUCap*

Para avaliar se houve melhoria de desempenho nos servidores inicialmente verificamos a normalidade dos dados do tempo de execução para assim poder escolher qual teste estatístico deveria ser utilizado e os intervalos de confiança. Os resultados do teste de normalidade de *Shapiro-Wilk* se encontram na Tabela A.5, dentre os 4 possíveis cenários apenas um apresentou *p-value* maior que *0,05* que foi para a heurística *BalanceInstancesOS* com o servidor2 quando estava sobrecarregado e quando não estava, para este cenário utilizamos o

teste *t-student* e para os outros foi escolhido o teste de *Wilcoxon*

Tabela A.5: Resultados do Teste de Normalidade *Shapiro-Wilk* para o tempo de execução do *Sysbench* de acordo com as heurísticas quando houve migrações (CAP inicial 50%)

| Heurística | Rodada do Balanceador | Servidor/Condição | *p-value* |
|---|---|---|---|
| *BalanceInstancesOS* | #1 | *servidor2/Sobrecarregado* | **0,5419** |
| | | *servidor2/ Não Sobrecarregado* | **0,2948** |
| | #2 | *servidor1 / Sobrecarregado* | 8,057e-06 |
| | | *servidor1/ Não Sobrecarregado* | 2,137e-06 |
| *CPUCapAware* | #1 | *servidor2/Sobrecarregado* | 1,619e-05 |
| | | *servidor2/ Não Sobrecarregado* | **0,1288** |
| *SysbenchPerfCPUCap* | #1 | *servidor2/Sobrecarregado* | 5,356e-06 |
| | | *servidor2/ Não Sobrecarregado* | 1,358e-05 |

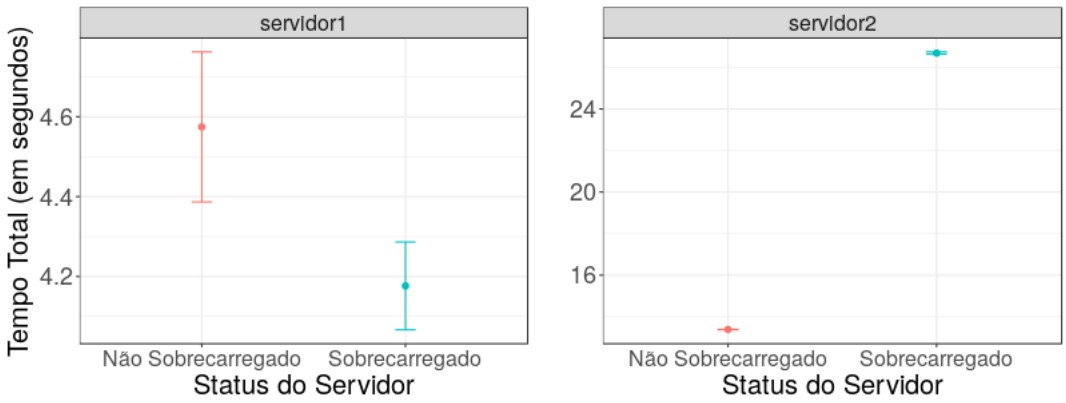


Figura A.20: Intervalo de Confiança 95% para *BalanceInstancesOS* comparando o tempo do *Sysbench* quando os servidores estavam sobrecarregados e quando não estavam

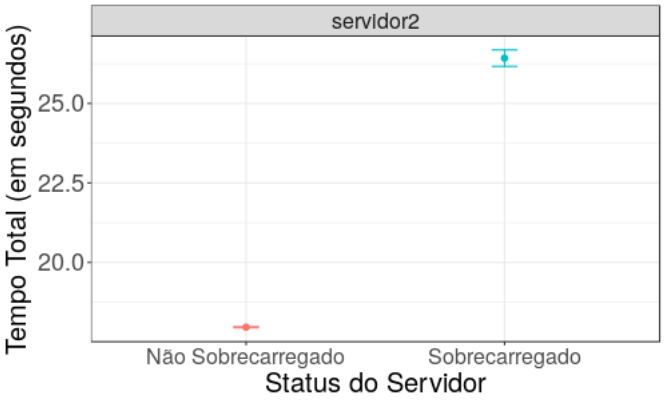


Figura A.21: Intervalo de Confiança 95% para *CPUCapAware* comparando o tempo do *Sysbench* quando os servidores estavam sobrecarregados e quando não estavam

As Figuras A.20 a A.22 apresentam os intervalos de confiança para as heurísticas *BalanceInstancesOS*, *CPUCapAware* e *SysbenchPerfCPUCap* respectivamente, neles apresentamos os intervalos de quando cada servidor estava sobrecarregado e após não estar mais sobrecarregado.

Todos os intervalos de confiança não apresentam sobreposição, para o servidor2 a diferença das escalas é perceptível, já para o servidor1 a diferença nas escalas é pequena variando apenas valores decimais, indicando que há diferença de desempenho em todos os cenários avaliados.

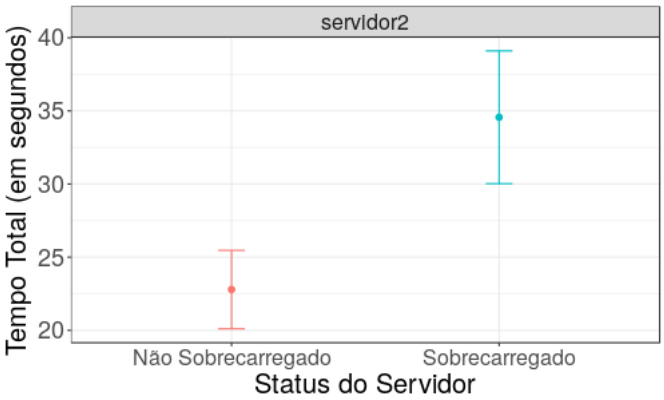


Figura A.22: Intervalo de Confiança 95% para *SysbenchPerfCPUCap* comparando o tempo do *Sysbench* quando os servidores estavam sobrecarregados e quando não estavam

Na Tabela A.6 apresentamos os *p-value* para cada cenário e qual o teste que foi aplicado,

todos os cenários apresentaram *p-value* menor que *0,05* refutando a hipótese de que seguiam a mesma distribuição (*Wilcoxon*) ou que possuiam a mesma média (*t-student*), confirmando as análises anteriores dos intervalos de confiança.

Tabela A.6: Resultados do Testes de Hipótese para cada Heurística comparando o desempenho (CAP 50%)

| Heurística | Teste | Rodada do Balanceador | Servidor sobrecarregado | *p-value* |
|---|---|---|---|---|
| *BalanceInstancesOS* | *t student* | #1 | servidor2 | 2,2e-16 |
| | *Wilcoxon* | #2 | servidor1 | 0,0006313 |
| *CPUCapAware* | *Wilcoxon* | #1 | servidor2 | 5,96e-08 |
| *SysbenchPerfCPUCap* | *Wilcoxon* | #1 | servidor2 | 1,508e-05 |

# Apêndice B

# Guia do Balanceador de Carga

Este apêndice irá ajudá-lo a instalar, configurar e executar o balanceador de carga em uma máquina, além de como criar novas heurísticas. Certifique-se de que a máquina que você está instalando o serviço tenha acesso aos *Hypervisors* KVM nos servidores de infraestrutura e aos seguintes serviços OpenStack: *Keystone*, *Nova* e *Monasca*. A versão atual apenas suporta a infraestrutura OpenStack e considera armazenamento compartilhado para o processo de migração de VMs.

## B.1 Instalação

Você pode instalar o balanceador de carga em qualquer máquina física ou virtual, a configuração mínima é descrita abaixo.

- Sistema Operacional: Ubuntu 14.04
- CPU: 1 núcleo
- Memória: 2GB

Os passos de instalção são descritos abaixo:

1. Atualize a sua máquina

   ```
   $ sudo apt-get update && sudo apt-get upgrade
   ```

2. Instale os pacotes necessários (dependências do python, *git* e *pip*)

```
$ sudo apt-get install python-setuptools python-dev build-essential
$ sudo easy_install pip
$ sudo apt-get install git
$ sudo apt-get install libssl-dev
```

3. clone o repositório do balanceador de carga

```
$ git clone https://github.com/bigsea-ufcg/bigsea-loadbalancer.git
```

4. Acesse a pasta do balanceador e rode o *script* de instalação.

```
$ cd bigsea-loadbalancer/
# Alguns requisitos precisam de sudo.
$ sudo ./install.sh
```

## B.2 Configuração

O arquivo de configuração para o balanceador de carga atualmente composto das seguintes seções: *monitoring*, *heuristic*, *infrastructure*, *openstack*. A Figura B.1 apresenta um exemplo deste arquivo e a Tabela B.1 oferece a descrição de cada parâmetro.

Tabela B.1: Descrição dos parâmetros de cada seção do arquivo de configuração

| Seção | Parâmetro | Descrição |
|---|---|---|
| monitoring | username | nome do usuário para acessar o serviço do *Monasca* |
| | password | senha do usuário para acessar o serviço do *Monasca* |
| | project_name | nome do projeto ao qual o usuário está associado |
| | auth_url | url de autenticação |
| | monasca_api_version | versão da API do Monasca |
| heuristic | cpu_ratio | valor com casa decimal representando a proporção do número de CPUs que os servidores utilizaram (fator de *overcommit*) |
| | wait_rounds | número de rodadas que uma máquina virtual que já foi migrada deve esperar antes que possa ser migrada novamente, cada rodada representa uma execução do balanceador. |
| | module | o nome do arquivo onde a heurística foi implementada (sem o .py) |
| | class | o nome da classe da heurística contida no módulo definido |
| | period | número de segundos que se esperar antes de executar a heurística novamente |
| infrastructure | user | nome do usuário com acesso aos servidores |
| | hosts | lista com o nome completo dos servidores separados por vírgula. (ex: compute.mylab.edu.br) |
| | key | localização da chave para acessar os servidores) |
| | provider | nome do provedor de infraestrutura (OpenStack) |
| openstack | username | nome do usuário para acessar os serviços da Nuvem |
| | password | senha do usuário para acessar os serviços da Nuvem |
| | user_domain_name | nome do domínio da nuvem ao qual o usuário faz parte |
| | project_name | nome do projeto ao qual o usuário está associado |
| | project_domain_name | nome do domínio da nuvem ao qual o projeto está associado |
| | auth_url | url de autenticação |

```
[monitoring]
username=<@username>
password=<@password>
project_name=<@project_name>
auth_url=<@auth_url>
monasca_api_version=2_0

[heuristic]
# A float value that represents the ratio of number of cores in the hosts. (overcommit factor)
cpu_ratio=0.5
# An integer that represent the number of rounds that a instance need to wait before be migrated again
# Each round represents an execution of the loadbalancer
wait_rounds= 1
# The filename for the module that is located in /loadbalancer/service/heuristic/
# without .py extension
module=<module_name>
# The class name that is inside the given module, this class should implement BasicHeuristic
class=<class_name>
#Number of seconds before execute the heuristic again
period=<value>

[infrastructure]
# The user that have access to each host
user=<username>
#List of full hostnames of servers that the loadbalancer will manage (separated by comma).
#e.g compute1.mylab.edu.br
hosts=<host>,<host2>
#The key used to access the hosts
key=<key_path>
#The type of IaaS provider on your infrastructure e.g OpenStack, OpenNebula
provider=OpenStack

[openstack]
username=<@username>
password=<@password>
user_domain_name=<@user_domain_name>
project_name=<@project_name>
project_domain_name=<@project_domain_name>
auth_url=<@auth_url>
```

Figura B.1: Exemplo de arquivo de configuração do balanceador de carga

As Figuras B.2 a B.4 representam como deve ser configurada a seção *heuristic* para utilizar as heurísticas *BalanceInstancesOS*, *CPUCapAware* e *SysbenchPerfCPUCap* respectivamente.

```
[heuristic]
# The filename for the module that is located in /loadbalancer/service/heuristic/
# without .py extension
module=instances
# The class name that is inside the given module, this class should implement BasicHeuristic
class=BalanceInstancesOS
#Number of seconds before execute the heuristic again
period=600
# A float value that represents the ratio of number of CPUs in the hosts.
cpu_ratio=1
# An integer that represent the number of rounds that a instance need to wait before be migrated again
# Each round represents an execution of the loadbalancer
wait_rounds=1
```

Figura B.2: Configuração da seção *heuristic* para utilizar a heurística *BalanceInstancesOS*

```
# Section to configure all heuristic information.
[heuristic]
# The filename for the module that is located in /loadbalancer/service/heuristic/
# without .py extension
module=<module_name>
# The class name that is inside the given module, this class should implement BasicHeuristic
class=<class_name>
#Number of seconds before execute the heuristic again
period=<value>
# A float value that represents the ratio of number of CPUS in the hosts. (overcommit factor)
cpu_ratio=0.5
# An integer that represent the number of rounds that an instance need to wait before be migrated again
# Each round represents an execution of the loadbalancer
wait_rounds= 1
```

Figura B.3: Configuração da seção *heuristic* para utilizar a heurística *CPUCapAware*

```
[heuristic]
# The filename for the module that is located in /loadbalancer/service/heuristic/
# without .py extension
module=benchmark_performance
# The class name that is inside the given module, this class should implement BasicHeuristic
class=SysbenchPerfCPUCap
#Number of seconds before execute the heuristic again
period=600
# A float value that represents the ratio o number of cores in the hosts.
cpu_ratio=1
# An integer that represent the number of rounds that a instance need to wait before be migrated again
# Each round represents an execution of the loadbalancer
wait_rounds=1
```

Figura B.4: Configuração da seção *heuristic* para utilizar a heurística *SysbenchPerfCPUCap*

## B.3 Execução

Para executar o balanceador de carga, você deve acessar o diretório do repositório e definir a variável de ambiente *PYTHONPATH*.

```
$ cd bigsea-loadbalancer/
$ export PYTHONPATH=":"`pwd`
```

Para executar utilizando o arquivo de configuração com nome padrão

```
$ python loadbalancer/cli/main.py
```

Utilizando arquivo de configuração sem nome padrão

```
$ python loadbalancer/cli/main.py -conf load_balancer.cfg
```

## B.4 Criação de novas Heurísticas

Para criar uma nova heurística você deve seguir os passos abaixo:

1. Criar um arquivo python no diretório loadbalancer/service/heuristic

2. Neste arquivo crie uma classe que herda a classe *BaseOptimizer*

```
from loadbalancer.service.heuristic.base import BaseHeuristic
...

class MyNewHeuristic(BaseHeuristic):
     def __init__ (self , **kwargs):
          ...
          ...

     def collect_information(self):
          ...
          return ...

     def decision(self):
          ...
          ...
```

3. Sobrescreva os métodos *request_instances* e *decision* na sua classe.