DIEGO GIACOMELLI CARDOSO

Dimensionamento Vertical Automático de Recursos em Nuvens Computacionais

Vitória-ES

2019

DIEGO GIACOMELLI CARDOSO

Dimensionamento Vertical Automático de Recursos em Nuvens Computacionais

Dissertação de mestrado submetida ao Programa de Pós-Graduação em Informática da Universidade Federal do Espírito Santo como requisito parcial para a obtenção do título de Mestre em Informática. Área de concentração: Ciência da Computação.

Universidade Federal do Espírito Santo – UFES

Programa de Pós-Graduação em Informática – PPGI

Orientador: Rodolfo da Silva Villaça

Coorientador: Renato Elias Nunes de Moraes

Vitória-ES

2019

Ficha catalográfica disponibilizada pelo Sistema Integrado de Bibliotecas - SIBI/UFES e elaborada pelo autor

---

Cardoso, Diego Giacomelli, 1988-
C268d Dimensionamento Vertical Automático de Recursos em Nuvens Computacionais / Diego Giacomelli Cardoso. - 2019.
65 f. : il.

Orientador: Rodolfo da Silva Villaça.
Coorientador: Renato Elias Nunes de Moraes.
Dissertação (Mestrado em Informática) - Universidade Federal do Espírito Santo, Centro Tecnológico.

1. Dimensionamento Vertical. 2. Computação em Nuvem. 3. Gerenciamento de Recursos, Elasticidade. I. Villaça, Rodolfo da Silva. II. Moraes, Renato Elias Nunes de. III. Universidade Federal do Espírito Santo. Centro Tecnológico. IV. Título.

CDU: 004

DIMENSIONAMENTO VERTICAL AUTOMÁTICO DE RECURSOS EM NUVENS COMPUTACIONAIS

Diego Giacomelli Cardoso

Dissertação submetida ao Programa de Pós-Graduação em Informática da Universidade Federal do Espírito Santo como requisito parcial para a obtenção do grau de Mestre em Informática.

Aprovada em 25 de fevereiro de 2019:

Prof. Dr. Rodolfo da Silva Villaça
Orientador

Prof. Dr. Renato Elias Nunes de Moraes
Coorientador

Prof. Dr. Vinícius Fernandes Soares Mota
Membro Interno

Prof. Dr. Antônio Augusto de Aragão Rocha
Membro Externo

UNIVERSIDADE FEDERAL DO ESPÍRITO SANTO
Vitória-ES, 25 de fevereiro de 2019.

Agradecimentos

Gostaria de agradecer à minha família e meus amigos pelo apoio incondicional nesta jornada. Obrigado por estarem ao meu lado, muitas vezes em momentos difíceis, momentos em que quis jogar tudo para o alto. Obrigado por sempre acreditarem em mim, por me darem forças para terminar este trabalho.

Aos meus orientadores Rodolfo da Silva Villaça e Renato Elias Nunes de Moraes um grande obrigado por me orientarem neste trabalho. Obrigado por me guiarem, mesmo eu sendo teimoso na implementação dos códigos. Aos membros da banca examinadora, Vinícius Fernandes Soares Mota e Antônio Augusto de Aragão Rocha, um grande obrigado por participarem da avaliação do meu trabalho. Agradeço a todos da banca por ajudarem a encontrar pontos de falha nas argumentações do trabalho, bem como apontar pontos de melhorias.

Aos amigos do laboratório NERDS, por trilharem este caminho comigo, compartilhando alegrias e tristezas, sucessos e dificuldades. Foi uma honra poder trabalhar com vocês. Aos colegas do projeto FUTEBOL, obrigado pela oportunidade de vivenciar uma experiência única de trabalho em cooperação atravessando o oceano Atlântico.

Aos colegas da Vale e da Tata Consultancy Services, um obrigado por me acolherem na reta final da escrita desta dissertação, momento bastante estressante para qualquer mestrando.

"O sucesso é ir de fracasso em fracasso sem perder entusiasmo."
(Winston Churchill)

Resumo

Computação em nuvem é um paradigma capaz de oferecer recursos de processamento, armazenamento, rede, entre outros, para diferentes tipos de aplicações. A computação em nuvem tornou-se popular por permitir o compartilhamento do uso de recursos computacionais por diferentes aplicações. Isso torna um atrativo, pois nuvens permitem que aplicações operem sob demanda, reduzindo custos operacionais por parte dos provedores destas aplicações. Máquinas virtuais fornecidas por nuvens englobam ou hospedam aplicações. Com o uso por essas aplicações, as máquinas virtuais podem necessitar de mais poder de processamento ou recursos de memória. Em caso de saturação, situação aonde a máquina virtual possui gargalos em algum de seus recursos, pode ocorrer perda de desempenho nas aplicações. Com isso, tem-se o desafio de como gerenciar os recursos de uma nuvem a fim de atender demandas de diferentes máquinas virtuais, melhorando a alocação destes recursos e evitando a sua saturação. Neste contexto este trabalho propõe um protótipo de dimensionador de recursos de nuvem, que entrega recursos sob demanda para máquinas virtuais saturadas, ou remove recursos subutilizados de máquinas virtuais, sem que ocorra a interrupção da própria máquina virtual. Para avaliar o protótipo proposto, são feitos três estudos de caso, sendo o primeiro com um gerador de carga sintético, o segundo com um servidor *web* sob alta demanda e o terceiro o controle remoto de um robô em um espaço inteligente.

Palavras-chave: Dimensionamento Vertical, Computação em Nuvem, Gerenciamento de Recursos, Elasticidade.

Abstract

Cloud computing is a paradigm capable of providing processing, storage, network and other resources, for different types of applications. Cloud computing becomes popular by allowing the sharing of resource usage by different applications. This makes it attractive because clouds allow applications to operate on demand, reducing operating costs by application providers. Virtual machines provided by clouds encompass or host applications. With the use by these applications, virtual machines may require more processing power or memory resources. In case of saturation, situation where the virtual machine has bottlenecks in some of its resources, there may be loss of performance in the application. Therefore, there is a challenge of how to manage the resources of a cloud in order to meet the demands of different virtual machines and improving the allocation of these resources, in order to avoid the saturation of resource use. In this context, this work proposes a prototype of cloud scaler, which delivers on-demand resources to saturated virtual machines, or removes underutilized resources from virtual machines, without interrupting the virtual machine itself. To evaluate the proposed prototype, three case studies are made, the first with a synthetic load generator, the second with a web server under heavy load, and the third is the remote control of a robot in an intelligent space.

Keywords: *Vertical Scaling, Cloud Computing, Resources Management, Elasticity*

Lista de ilustrações

Lista de abreviaturas e siglas

VM	Virtual Machine
CPU	Central Processing Unit
API	Application Programming Interface
SDK	Software Development Kit
QoS	Quality of Service
IaaS	Infrastructure-as-a-Service
FPS	Frames Per Second
JSON	JavaScript Object Notation

Sumário

1 Introdução

Computação em Nuvem é um paradigma capaz de oferecer recursos de processamento, armazenamento, rede, dentre outros, para diversos tipos de aplicações, como por exemplo redes sociais, comércio eletrônico, serviços multimídia orientados por assinatura, tais como Spotify® e Netflix™ (QU; CALHEIROS; BUYYA, 2018; YAZDANOV; FETZER, 2013). Outras aplicações incluem serviços de hospedagem (UOL, 2019) e um facilitador que contribui para a Indústria 4.0 (THAMES; SCHAEFER, 2016).

A Figura 1 mostra o conceito de uma nuvem computacional genérica. Como dito, uma nuvem pode conter diferentes tipos de aplicações e serviços e podem ser acessadas de diferentes fontes, como computadores, celulares, *tablets*, entre outros.

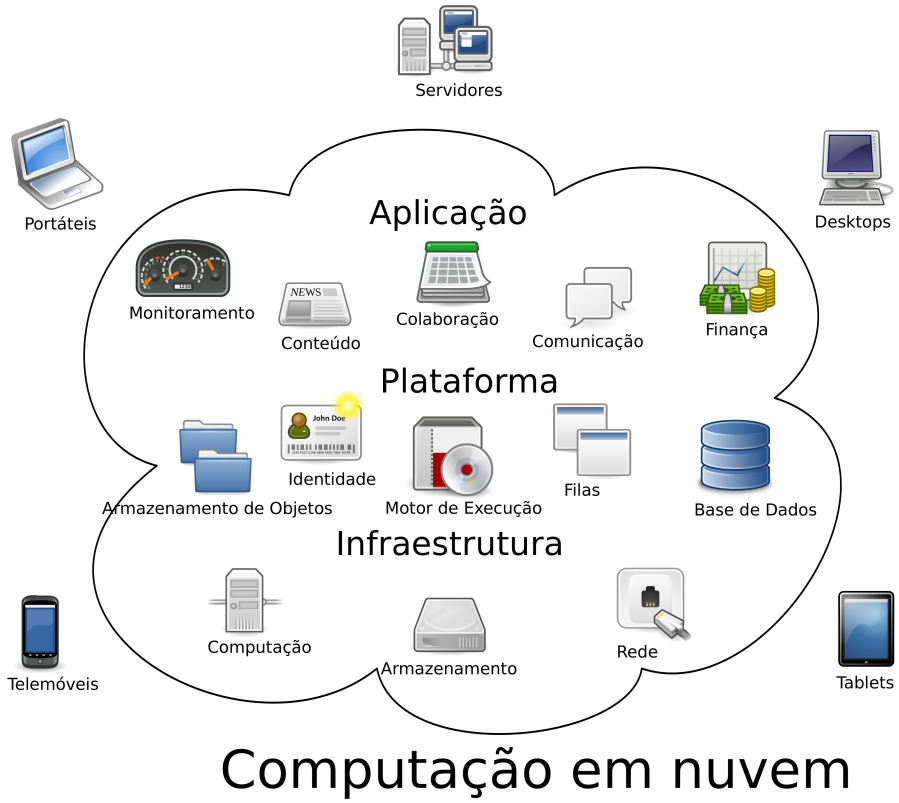


Figura 1 – Conceito de uma nuvem.

Créditos: http://upload.wikimedia.org/wikipedia/commons/b/b5/Cloud_computing.svg

Computação em nuvem possui um potencial para transformar a indústria de TI (Tecnologia da Informação), transformando *software* em serviços e moldando a forma como *hardware* é desenhado e vendido (THAMES; SCHAEFER, 2016). Não é necessário para desenvolvedores arcar com custos de compra de *hardware* para implantação de seus serviços

e produtos para Internet (THAMES; SCHAEFER, 2016). O uso de nuvens computacionais permite que empresas, ou provedoras de aplicações, dimensionem dinamicamente seus produtos e serviços, sem a preocupação de ter que prever o quanto de recursos serão necessários (THAMES; SCHAEFER, 2016) para as suas aplicações.

Um dos principais objetivos da computação em nuvem é melhorar o uso de recursos computacionais. Tais recursos podem ser compartilhados e combinados para se obter um melhor desempenho na execução das aplicações e serviços, bem como na resolução de problemas computacionais de larga escala. A computação em nuvem lida com diferentes aspectos, tais como virtualização, escalabilidade, interoperabilidade e qualidade de serviço(JADEJA; MODI, 2012). Isso faz com que o usuário final de uma aplicação ou serviço hospedado em uma nuvem não necessite ter conhecimentos específicos sobre a localização física da nuvem e nem das configurações do sistema que entrega os serviços (JADEJA; MODI, 2012). Aplicações podem ser entregues como serviços pela Internet. Como resultado, existe também uma redução de custos (JADEJA; MODI, 2012), pois nuvens permitem que processamento e armazenamento de dados sejam movidos de computadores pessoais e computadores portáteis para a nuvem (DIKAIAKOS et al., 2009).

Em geral, as aplicações são hospedadas em máquinas virtuais (ou *Virtual Machines* - VM) e máquinas virtuais são instanciadas por nuvens. E uma característica inerente às nuvens, tanto públicas quanto privadas, é a elasticidade. A elasticidade consiste em aumentar ou reduzir a disponibilidade de recursos computacionais em uma VM de maneira dinâmica, de acordo com alguma política de alocação que considere a demanda de recursos feita por uma VM (QU; CALHEIROS; BUYYA, 2018; ARABNEJAD et al., 2016; XU et al., 2007).

Um exemplo de elasticidade consiste em, dada uma VM contendo uma aplicação que demanda uma certa quantidade de recursos (processador e memória) em função de uma carga trabalho. Em horários de pico, a aplicação necessita de mais processador e memória. A nuvem então deve ser capaz de disponibilizar recursos extras para esta VM. A elasticidade permite que esta nova demanda seja automaticamente suprida pela nuvem a fim de se evitar gargalos de uso por parte dos usuários, consequentemente melhorando o desempenho das aplicações hospedadas na VM.

A elasticidade é um atrativo para que provedores de aplicações movam suas aplicações para dentro de nuvens (QU; CALHEIROS; BUYYA, 2018) pois nuvens reduzem custos de aplicações por meio do compartilhamento de recursos. O compartilhamento de recursos gera um desafio pois é difícil prever a quantidade de recursos que cada VM irá exigir. Simplesmente entregar o máximo de recursos para uma máquina virtual se torna inviável, pois em um modelo de computação em nuvem é interessante que exista uma quantidade de recursos disponíveis para serem alocados à maior variedade de máquinas virtuais ou aplicações possíveis, de maneira que esses recursos não fiquem alocados exclusivamente

para algumas aplicações.

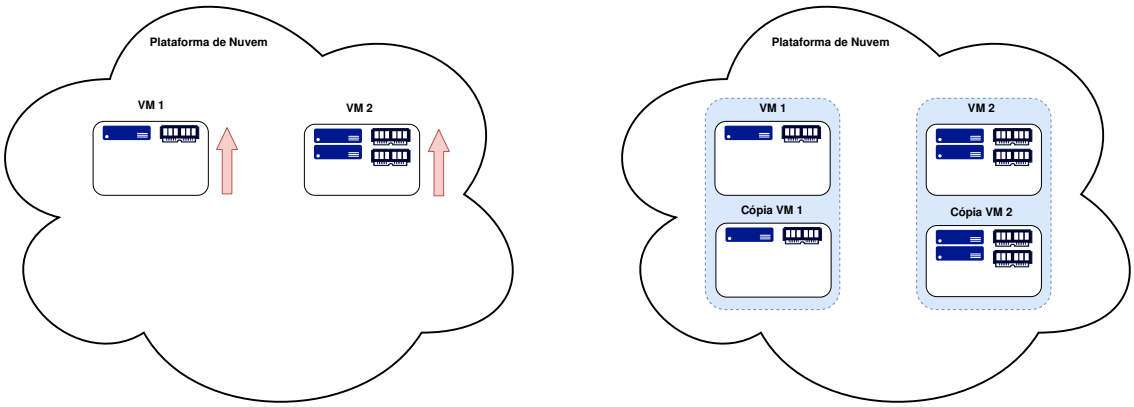
Por outro lado, manter máquinas virtuais com o mínimo de recursos possível também é inviável, pois o mínimo pode ser insuficiente para atender as demandas que as aplicações possam exigir. Com isso, uma nuvem deve ser capaz de alocar recursos para cada VM de tal maneira que seja garantido que não existam gargalos de utilização nas máquinas virtuais (XU et al., 2007), consequentemente garantindo o melhor desempenho das aplicações hospedadas em cada VM. Uma abordagem para garantir a melhor utilização do uso dos recursos da nuvem é o dimensionamento automático (ou *auto-scaling*).

O processo de *auto-scaling* dimensiona automaticamente a quantidade de recursos de uma máquina virtual com o objetivo de se manter o uso dos recursos da VM dentro de parâmetros classificados como aceitáveis (ARABNEJAD et al., 2016) evitando gargalos de utilização.

O dimensionamento de recursos pode ser classificado como horizontal ou vertical (ARABNEJAD et al., 2016). No dimensionamento horizontal (*horizontal scaling*) o dimensionamento consiste na instanciação de uma nova VM ou remoção de uma VM pré-existente. A Figura 2 ilustra como funciona o dimensionamento horizontal. Na Figura 2(a), tem-se duas máquinas virtuais saturadas, necessitando de mais recursos (processador e memória). Quando se dimensiona horizontalmente, é feita uma cópia da VM e o resultado pode ser visto na Figura 2(b). A cópia mantém os serviços e aplicações da VM original e, quando não se torna mais necessária, essa cópia é removida pelo dimensionador de recursos de nuvem.

Nuvens comerciais como Google (GOOGLE, 2019) e Amazon (AMAZON, 2018) oferecem suporte para dimensionamento horizontal. Porém, para que seja possível o dimensionamento horizontal é necessário que a aplicação dentro da VM seja preparada para tal, pois alguns pré-requisitos devem ser atendidos. O primeiro é o balanceamento de carga. A premissa é que ao ser criada uma nova VM, ela deve receber parte da carga de trabalho da VM original, caso contrário, não haverá utilidade para a sua criação. A entidade que faz essa redistribuição de carga de trabalho, é chamado de balanceador de carga. Outro pré-requisito é a não dependência de estado. A dependência de estado pode ser vista, por exemplo, como dados armazenados na máquina virtual. Ao duplicar a VM, esses dados podem tornar-se inconsistentes, causando efeitos negativos na aplicação como um todo.

O dimensionamento vertical (*vertical scaling*) funciona de maneira diferente. O dimensionamento vertical envolve a disponibilização de novos recursos sem a realização de cópias da VM original. Um exemplo pode ser visto na Figura 3. Na Figura 3(a) tem-se duas VMs (VM 1 e VM 2) necessitando de mais recursos, em outras palavras, saturadas, e uma VM com subutilização de recursos (VM 3). Na Figura 3(b) tem-se o resultado do dimensionamento vertical, onde as máquinas virtuais VM 1 e VM 2 recebem mais recursos



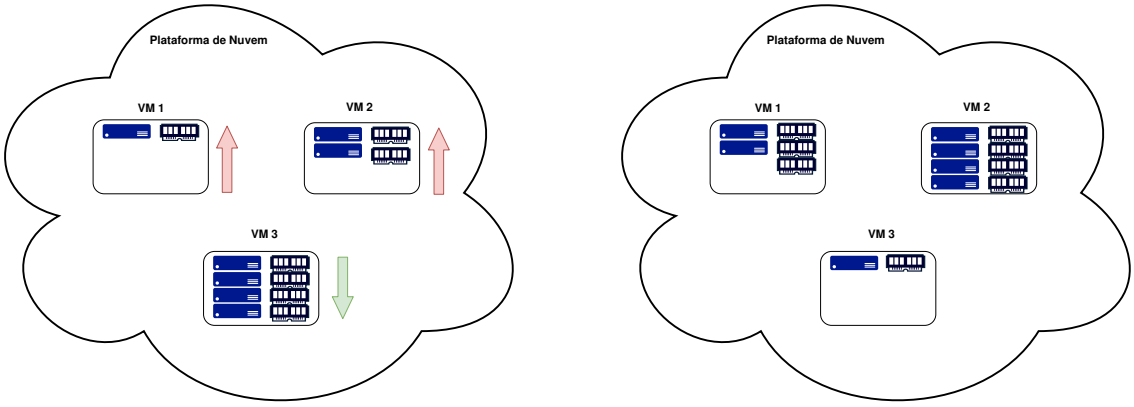
(a) Máquinas Virtuais VM 1 e VM 2 necessitam de mais processador e memória.

(b) No dimensionamento horizontal é feita uma cópia da VM.

Figura 2 – Exemplo de dimensionamento horizontal.

(processador e memória), enquanto que a máquina virtual VM 3 tem recursos removidos, mas nenhuma outra máquina virtual é criada.

O dimensionamento vertical é mais flexível (DAWOUD; TAKOUNA; MEINEL, 2012) em sua execução por não requerer a instanciação de um balanceador de carga nem arranjos especiais de replicação de dados por parte da aplicação. O dimensionamento depende apenas da nuvem possuir recursos disponíveis e ocorre em unidades de recurso, como CPU e memória, sem que haja a disponibilização de uma máquina virtual completa. Porém o dimensionamento vertical é limitado à máquina física que hospeda VM (QU; CALHEIROS; BUYYA, 2018). Múltiplas VMs, normalmente, são hospedadas na mesma máquina física, competindo por recursos, o que limita o ainda mais a potencial capacidade para dimensioná-las (QU; CALHEIROS; BUYYA, 2018).



(a) Máquinas Virtuais VM 1, VM 2 necessitam de mais processador e memória. VM 3 possui recursos subutilizados.

(b) No dimensionamento vertical recursos são entregues/removidos diretamente para/das as VMs.

Figura 3 – Exemplo de dimensionamento vertical.

O dimensionamento vertical ainda pode ser subdividido em 2 tipos: o dimensio-

namento a quente e o a frio. No dimensionamento a frio, a troca de recursos, isto é, o aumento ou redução de memória e/ou CPU, é feita seguindo um conjunto de passos de desligar a VM, realizar a troca e finalmente religar a VM. No dimensionamento a quente, a troca é feita com a máquina virtual funcionando, sem a necessidade de desligá-la ou reiniciá-la.

Nuvens comerciais como Amazon (AMAZON, 2019) e Google (GOOGLE, 2019) oferecem dimensionamento horizontal e vertical a frio, mas não vertical a quente. Neste contexto, este trabalho tem como objetivo propor um dimensionador vertical para utilização em nuvens computacionais. O dimensionador proposto deve entregar recursos dinamicamente às VMs, adicionando ou removendo processador e memória com a máquina virtual em funcionamento, sem a necessidade de reinicialização, ou seja, sem afetar a disponibilidade dos serviços hospedados na VM automaticamente dimensionada. Um protótipo do dimensionador vertical será implementado em uma nuvem real, no ambiente OpenStack, e será capaz de gerenciar recursos de CPU e memória. A entrega (ou remoção) de recursos pelo dimensionador será feita de tal maneira que não será necessário desligar/reiniciar as máquinas virtuais, ou seja, sem interromper os serviços e aplicações hospedados nestas VMs.

O processo geral do dimensionador consiste em analisar as máquinas virtuais existentes e em funcionamento na nuvem. Para cada máquina virtual, o dimensionador deve realizar um ciclo de coleta, análise e execução em função das métricas de utilização dos recursos computacionais alocados em cada VM. Na fase de coleta o dimensionador obtém a utilização dos recursos de uma máquina virtual. Na fase de análise o dimensionador verifica se a VM está saturada, ou seja, verifica se o uso de recursos está dentro dos limites superior e inferior previamente determinados pelos donos das aplicações. Na fase de execução o dimensionador adiciona recursos, ou seja, adiciona mais CPU e memória, caso a VM esteja saturada, ou remove recursos no caso de subutilização.

Os objetivos específicos deste trabalho são:

- Definir e utilizar APIs (*Application Programming Interface*) e SDKs (*Software Development Kit*) padronizadas para cada nuvem, assim facilitando a portabilidade do dimensionador;
- Atuar em diferentes virtualizadores também facilitando a portabilidade do dimensionador, o que pode ser alcançado por meio do uso da biblioteca libvirt (LIBVIRT, 2019);
- Usar um barramento de difusão de mensagens, como o RabbitMQ (PIVOTAL, 2019), que permita que o dimensionador possa ser dividido em diferentes micro-serviços desacoplados, dando mais flexibilidade na sua implementação;

- Permitir a configuração dos parâmetros de utilização do dimensionador sem que seja necessário conhecer os detalhes de sua implementação e instalação na nuvem. Esse objetivo é alcançado por meio do uso de um modelo cliente-servidor e a implementação de um cliente para configuração do dimensionador na nuvem.

A avaliação do protótipo do escalonador consiste em analisar seu desempenho em 3 (três) cenários distintos:

1. Avaliar seu comportamento em um consumidor sintético de recursos. Este consumidor cria uma carga de uso no processador e na memória. O objetivo deste cenário é avaliar diferentes possibilidades de uso e mostrar que os mecanismos de dimensionamento funcionam;

2. Avaliar os benefícios de alocação dinâmica de recursos para um serviço *web*. Neste estudo de caso, tem-se um servidor *web* que atende um certo número clientes com uma disponibilidade aceitável. Em um certo momento uma grande quantidade de clientes começa a acessar o servidor *web*, reduzindo sua disponibilidade. O dimensionador deve agir afim de mitigar a queda da disponibilidade do serviço *web* oferecido na nuvem;

3. Controle de um robô em um espaço inteligente. O robô deve seguir uma determinada trajetória pré-configurada e os serviços associados ao robô, tais como detecção e localização, processamento de imagens, controlador, entre outros estão hospedados na nuvem. Para reduzir o erro de trajetória do robô, os serviços do espaço inteligente demandam mais poder de processamento e o dimensionador deve ser capaz de atender tais demandas dentro de um limite operacional. Espera-se que o uso do dimensionador tenha um impacto positivo no desempenho dos serviços utilizados pelo espaço inteligente, ou seja, espera-se reduzir o erro de trajetória do robô.

Por fim essa dissertação divide-se da seguinte forma: No Capítulo 2 é feita uma revisão bibliográfica da literatura, com discussão de trabalhos relacionados. No Capítulo 3 é apresentada a implementação do dimensionador e suas restrições de funcionamento. No Capítulo 4 é feita a avaliação do escalonador com base nos três estudos de caso propostos. O Capítulo 5 conclui o trabalho e apresenta trabalhos futuros relacionados a essa dissertação.

2 Trabalhos Relacionados

Dimensionamento de recursos pode ser feito de maneira horizontal ou vertical. Cada um deles possui suas vantagens e limitações (QU; CALHEIROS; BUYYA, 2018). Este capítulo discorre sobre diferentes trabalhos que abordam os diferentes tipos de dimensionamento. Além disso, neste capítulo é feita uma breve explicação sobre o serviço de dimensionamento de recursos do OpenStack, um ambiente de nuvem, de código aberto, com presença marcante da comunidade e empresas que apoiam ou utilizam a plataforma em seus negócios.

Dimensionar horizontalmente significa operar em número de máquinas virtuais. Em outras palavras, uma nova máquina virtual será instanciada se uma determinada VM estiver com elevado uso de recursos (saturada), ou uma VM será removida em caso de baixo uso de recursos. A ideia é que a carga de trabalho que uma VM saturada possui seja distribuída ente as máquinas virtuais criadas.

Nuvens comerciais como Amazon (AMAZON, 2018) e Google (GOOGLE, 2019) fornecem serviços de dimensionamento horizontal em suas respectivas plataformas de nuvens. Tipicamente, as configurações da nova VM, tais como quantidade de processador e memória, seguem as mesmas utilizadas na máquina virtual original analisada pelo dimensionador. Isto facilita na gerência da nuvem e do dimensionador além de ser um modelo bem aceito (QU; CALHEIROS; BUYYA, 2018).

Porém, não necessariamente uma nova máquina virtual precisa possuir as mesmas características da máquina virtual original (QU; CALHEIROS; BUYYA, 2018). Grozev *et. al.* (GROZEV; BUYYA, 2017) apresenta um método para avaliar o dimensionamento de uma VM utilizando a técnica de Memória Temporal Hierárquica, combinada uma rede neural artificial, para buscar uma configuração de VM que melhor atenda a demanda. Espera-se que no momento que ocorrer o dimensionamento horizontal, seja entregue uma nova VM mais específica para atender a nova demanda. Isto permite um dimensionamento mais fino e reduz o custo da aplicação (QU; CALHEIROS; BUYYA, 2018). Custo pode ser entendido como a quantidade de recurso financeiro utilizado para se manter uma aplicação dentro de uma máquina virtual em um ambiente de nuvem. Quem deve investir este recurso financeiro são os provedores de aplicações.

Outros trabalhos, como (FERNANDEZ; PIERRE; KIELMANN, 2014), apresentam uma estratégia que transforma todas as possíveis configurações de VMs em uma árvore e no momento do dimensionamento de recursos, busca-se na árvore, qual melhor configuração que atenda a carga de demanda de uma VM com o menor custo. Em (SHARMA; SHENOY; TOWSLEY, 2012) os autores utilizam algoritmos gulosos para obter um plano

de provisionamento de recursos considerando dimensionamento horizontal.

Apesar de existirem diferentes estratégias, o dimensionamento horizontal implica na criação de novas máquinas virtuais, o que torna o processo de dimensionamento lento (QU; CALHEIROS; BUYYA, 2018). Dawoud *et. al.* em seu trabalho (DAWOUD; TAKOUNA; MEINEL, 2012), realiza uma comparação entre dimensionamento horizontal e vertical. Na comparação, os autores demonstram que o dimensionamento vertical possui um desempenho melhor em relação ao dimensionamento horizontal.

Chen *et. al.* (CHEN; BAHSOON, 2017) apresenta uma heurística auto adaptável, considerando QoS (do inglês *Quality of Service*, ou Qualidade de Serviço) para ajustar seu próprio algoritmo, sem intervenção humana e dimensionando apenas CPU. Gong *et. al.* (GONG; GU; WILKES, 2010) apresenta um dimensionador vertical de CPU e memória, cuja análise é preditiva e entrega a quantidade de recursos ideal para uma determinada VM. Outra estratégia de dimensionamento é apresentada em (WANG; GUPTA; URGAONKAR, 2016), onde os autores apresentam um dimensionador vertical para CPU e memória. A tomada de decisão para aumentar/reduzir recursos vem de dentro da VM, considerando a medição de uso de recursos de dentro da própria máquina virtual.

Ambientes de nuvem também possuem seus dimensionadores nativos. Um exemplo de nuvem que implementa dimensionadores é o OpenStack (OPENSTACK, 2019), que é a ferramenta de estudo deste trabalho. O OpenStack é um *software* que controla um conjunto de recursos de computação, armazenamento e rede ao longo de um *data center* (OPENSTACK, 2019). O diagrama da visão geral do OpenStack pode ser visto na figura 4. Nela é possível notar que o OpenStack atua em diferentes tipos de serviços (rede, computação, acesso) para montar um ambiente de nuvem.

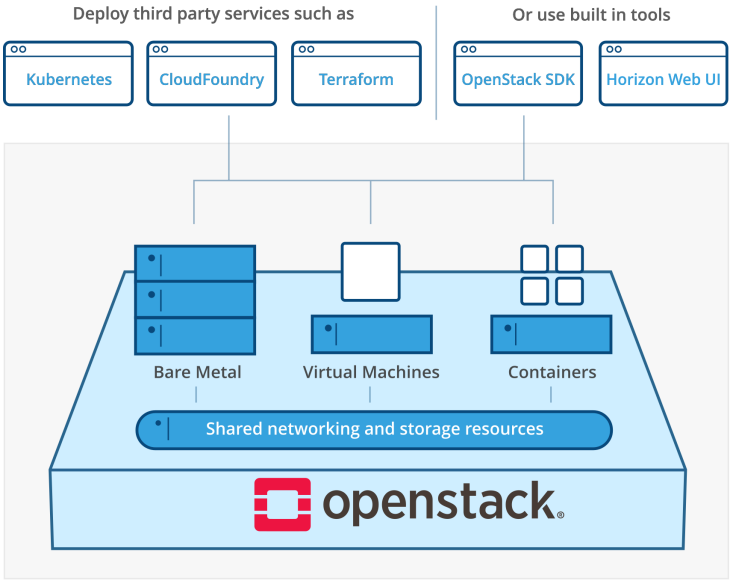


Figura 4 – Diagrama da visão geral do OpenStack.

Créditos: https://www.openstack.org/software/

	Vertical	Horizontal	A quente	Recursos	Virtualizador
(GROZEV; BUYYA, 2017)	não	sim	sim	CPU/mem	AWS
(FERNANDEZ; PIERRE; KIELMANN, 2014)	não	sim	sim	CPU	DAS-4/AWS
(SHARMA; SHENOY; TOWSLEY, 2012)	não	sim	sim	CPU	OpenNebula
(DAWOUD; TAKOUNA; MEINEL, 2012)	sim	não	sim	CPU	Xen
(CHEN; BAHSOON, 2017)	sim	não	sim	CPU	Xen
(GONG; GU; WILKES, 2010)	sim	não	sim	CPU/mem	Xen
(WANG; GUPTA; URGAONKAR, 2016)	sim	não	sim	CPU/mem	docker
Dimensionador Proposto	sim	não	sim	CPU/mem	OpenStack

Tabela 1 – Comparação de trabalhos de dimensionamento de recursos

O OpenStack opera sob o modelo (MELL; GRANCE et al., 2011) de Infraestrutura como Serviço (do inglês IaaS - *Infrastrucure-as-a-Service*). O que permite uma gerência por parte do usuário da nuvem em termos de serviços de processamento, armazenamento e rede. Além disso, o usuário da nuvem é capaz de implementar seu próprio sistema operacional.

Sendo uma nuvem, o OpenStack é munido com algumas ferramentas de dimensionamento de recursos. Uma delas é o horizontal, ou seja, o OpenStack é capaz de realizar dimensionamento horizontal. Outro dimensionamento realizado pelo OpenStack é o vertical, porém a frio. Isto significa que o OpenStack é capaz de realizar a troca de recursos verticalmente desligando a máquina virtual. Após realizar a troca dos recursos, o OpenStack religa a VM.

A Tabela 1 sumariza os trabalhos sobre dimensionamento de recursos de nuvem. A tabela classifica os trabalhos em dimensionamento vertical, horizontal, se é a quente ou não, quais tipos de recursos são usados no dimensionamento e qual tipo de virtualizador é utilizado. É possível perceber que trabalhos que tratam de dimensionamento vertical, focam em utilizar virtualizadores como Xen ou docker. Já os trabalhos que tratam de dimensionamento horizontal utilizam ambientes de nuvem. Como pode ser visto na tabela, a solução proposta neste trabalho é capaz de realizar dimensionamento vertical à quente em um ambiente de nuvem real.

Neste trabalho, é proposto um protótipo de dimensionador para nuvens. O protótipo baseia-se em regras para executar o dimensionamento. Significa que, se o nível de uso de recursos estiver fora de uma determinada faixa, o dimensionador atua, inserindo ou removendo recursos na VM. A inserção/remoção é feita de tal maneira que a máquina virtual continue funcionando, sem a necessidade de desligar ou reiniciar. Neste trabalho, formaliza-se um modelo de um dimensionador de recursos e o implementa para nuvens OpenStack.

3 Projeto e Implementação

Este capítulo trata do dimensionador e de detalhes sobre o protótipo que foi construído, tais como as tecnologias que foram utilizadas. O capítulo é subdivido como segue: no subcapítulo 3.1 é mostrada a arquitetura do dimensionador, no subcapítulo 3.2 são mostradas quais tecnologias foram utilizadas para construir o protótipo. Nos subcapítulos 3.3, 3.4 e 3.5 são mostrados os componentes do dimensionador. E no subcapítulo 3.6 é mostrado a estratégia adotada para tratamento das leituras de medidas feitas pelo protótipo de dimensionador.

É importante destacar aqui que o dimensionador tem por objetivo gerenciar recursos de uma ou mais máquinas virtuais. Para isso, o dimensionador considera o uso corrente de recursos, bem como os limites de uso estipulados para cada VM. Como Dawoud *et. al.* (DAWOUD; TAKOUNA; MEINEL, 2012) mostrou em seu trabalho, o dimensionamento vertical é limitado aos recursos disponíveis do hospedeiro físico, portanto é necessário que o dimensionador saiba o quanto ele pode inserir em uma VM. A remoção também é gerenciada, pois uma VM sem recursos, simplesmente não funciona.

3.1 Arquitetura do Dimensionador

A arquitetura do dimensionador proposto neste trabalho pode ser vista na Figura 5. O dimensionador é subdivido em três componentes: Cliente, Gerente e Agente. Cada componente se preocupa com uma função específica.

O componente Cliente atua como porta de entrada para o dimensionador. Com o Cliente é possível determinar qual máquina virtual deve ser dimensionada. Além disso, com o Cliente é possível definir quando o dimensionamento deve ocorrer, ou seja, cabe ao Cliente definir quais são os limites superior e inferior de uso de uma máquina virtual.

Tais limites indicam até que ponto a VM pode operar sem a necessidade de ocorrer um dimensionamento. Por exemplo, na Figura 3(a), a VM 1 pode ser configurada com limite superior de 80% de uso de memória e estar com uso corrente de 85%. Isso é o suficiente para que o dimensionador atue neste exemplo, entregando mais memória para a VM 1. O resultado pode ser visto na Figura 3(b). Os limites de uso variam para cada máquina virtual. Outra operação que o Cliente executa, é a finalização do dimensionamento de uma máquina virtual.

O componente Gerente é o coração do protótipo do dimensionador. Ele mantém o conjunto de máquinas virtuais que está dimensionando, bem como realiza a tomada de decisão de dimensionar uma VM, dado o seu grau de uso. Ele envia os comandos desejados

para o Agente.

O componente Agente é responsável por comunicar diretamente com a infraestrutura, aqui representado pelo OpenStack. Ele é responsável por realizar consultas sobre o estado atual das máquinas virtuais, bem como executar a alteração na quantidade de recursos das VMs. Porém, o Agente não realiza tomada de decisão, ele apenas recebe comandos oriundos do Gerente.

Uma ressalva deve ser feita aqui. O dimensionador não cria ou deleta máquinas virtuais. Ele trabalha somente com as máquinas virtuais existentes dentro da nuvem. O seu processo é gerenciar recursos para uma máquina virtual.

O dimensionador proposto neste trabalho opera com troca de mensagens entre seus três componentes: Cliente, Gerente e Agente. Exemplos de como as mensagens são construídas podem ser vistas no Apêndice A. Os componentes também operam no modelo *publish/subscribe*, que é um modelo onde se tem produtores e consumidores. Um produtor publica uma mensagem sem saber se existe um consumidor do outro lado para consumir esta mensagem. E consumidores, ao se inscreverem em um tópico, passam a receber mensagens relacionadas à esse tópico.

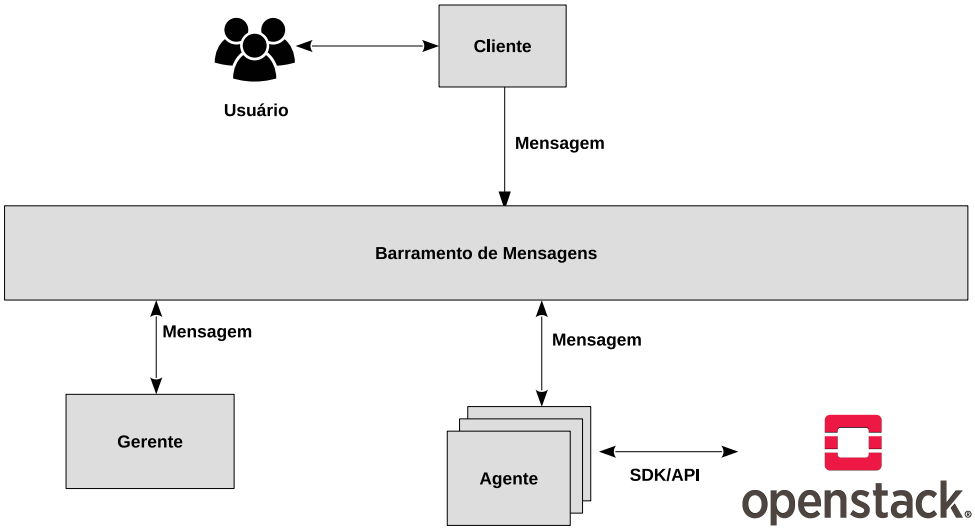


Figura 5 – Arquitetura do dimensionador de recursos.

Na Figura 5 é possível ver como que funciona o fluxo de operação do dimensionador. Tudo começa no componente Cliente, que envia uma mensagem de ligar o dimensionamento para o Gerente. A mensagem passa por um barramento de mensagens e a mensagem é recebida pelo Gerente.

O Gerente processa a mensagem, e inicia o dimensionamento. O Gerente envia uma mensagem para o Agente e o barramento de mensagens a entrega ao Agente. Esta mensagem pode ser uma consulta do estado de uma máquina virtual, bem como uma operação de dimensionamento (aumento/redução) de recursos.

O Agente recebe a mensagem do Gerente, acessa a infraestrutura via APIs e SDKs fornecidas pela nuvem, executa o pedido do Gerente e obtém o resultado da requisição. Por fim, o Agente envia o resultado para o Gerente e o barramento de mensagens entrega o resultado do Agente para o Gerente.

Todo esse processo é executado indefinidamente, até que o Cliente envie uma mensagem para desligar o dimensionamento para o Gerente.

Um exemplo de operação do protótipo de dimensionador pode ser visto na Figura 6. Na Figura 6(a) mostra o início do processo, com o Cliente enviando ao Gerente, uma mensagem para ligar o dimensionamento da VM 1. Na Figura 6(b), o Gerente envia mensagem ao Agente, requisitando dados sobre a VM 1. O Agente consulta a nuvem OpenStack para saber as condições da VM 1 e devolve os dados para o Gerente.

Na Figura 6(c) o Gerente detecta que a VM 1 está com uso elevado de CPU e memória. Então o Gerente envia uma mensagem para o Agente, requisitando que este atue na VM 1, entregando mais recursos de CPU e memória. Na Figura 6(d), o Gerente continua a análise da VM 1, verificando se a máquina virtual necessita de mais ou menos recursos.

Na Figura 6(e), o Cliente envia uma mensagem de desligamento para o Gerente. Por fim, na Figura 6(f), o Gerente processa a mensagem do Cliente e desliga o processo de dimensionamento para a VM 1.

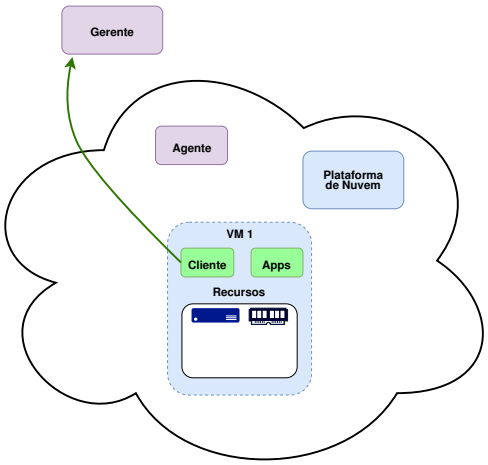
Os detalhes de implementação dos componentes do dimensionador, bem como seus respectivos fluxogramas, são apresentados nos Subcapítulos 3.3 para o Cliente, 3.4 para o Gerente e 3.5 para o Agente. No Subcapítulo 3.6 é mostrado como é feito o cálculo de uso de recursos para uma máquina virtual.

3.2 Implementação do Dimensionador

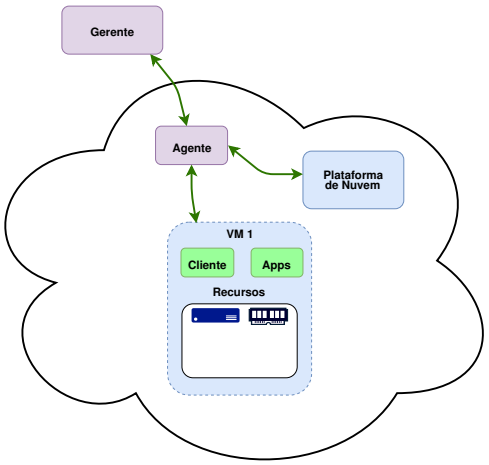
O protótipo do dimensionador foi implementado utilizando diferentes tecnologias. O código fonte foi todo escrito em Python e pode ser encontrado no repositório: `https://github.com/cardosodg/dimensionador_vertical.git`. A linguagem de programação Python é interpretada, interativa e orientada a objetos (PYTHON, 2019).

As vantagens do uso de Python incluem uma sintaxe simples com uma variada quantidade de bibliotecas e chamadas de sistema operacional. A linguagem Python permite uso de diferentes tecnologias para construção do protótipo do dimensionador.

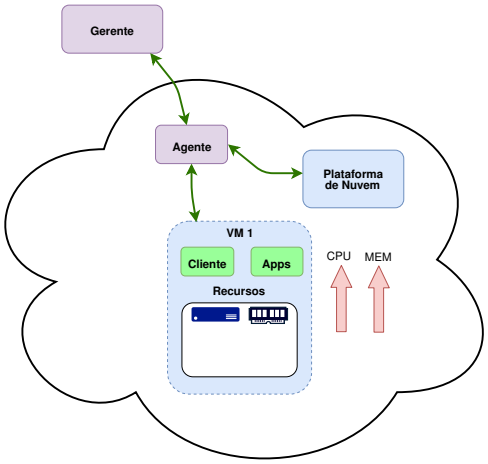
A infraestrutura de nuvem utilizada para a implementação do protótipo foi o OpenStack. O OpenStack é um sistema operacional de nuvem que controla uma variada gama de recursos de computação, rede e armazenamento (OPENSTACK, 2019). É totalmente *open source* e é financiado ou patrocinado por várias empresas de grande porte, tais



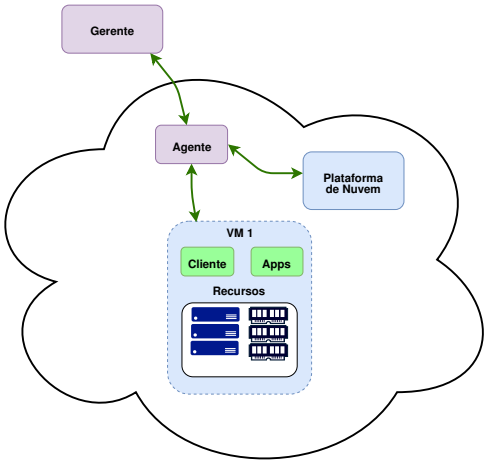
(a) Cliente envia mensagem de ligar dimensionamento para o Gerente.



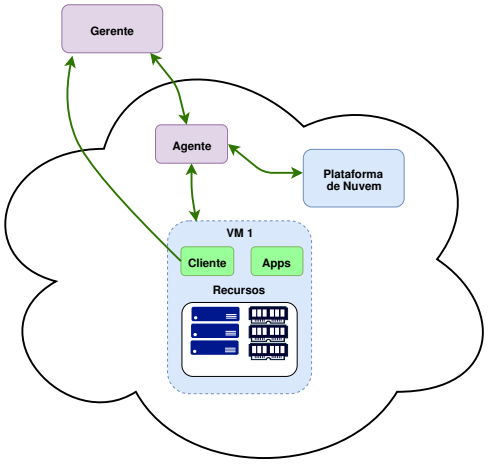
(b) Gerente envia mensagem ao Agente requerendo dados sobre a VM 1.



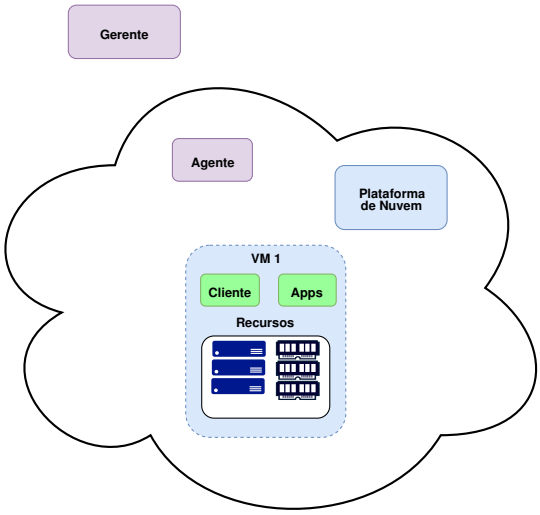
(c) Gerente detecta que VM 1 tem alto uso de recursos. Gerente envia mensagem para Agente atuar na VM 1.



(d) Agente atua na VM 1, aumentando recursos de CPU e memória.



(e) Cliente envia mensagem para o Gerente para desligar o dimensionamento.



(f) O dimensionamento é finalizado para a VM 1.

Figura 6 – Exemplo de operação do Gerente.

como Huawei, Ericsson, AT&T, RedHat, Canonical, Dell, entre muitas outras. Todos os componentes do OpenStack são divididos em módulos, o que permite personalização na implantação da nuvem.

A Figura 7 mostra a arquitetura conceitual do OpenStack. Na figura, é possível ver a relação entre os módulos do sistema. Por ser um projeto grande, o OpenStack possui uma gama muito maior de módulos que não estão representados na Figura 7. Os módulos na figura, ilustram a relação entre módulos para um ambiente de produção, considerando aspectos como alta disponibilidade, *backup* de dados e serviços e modelagem de clusterização.

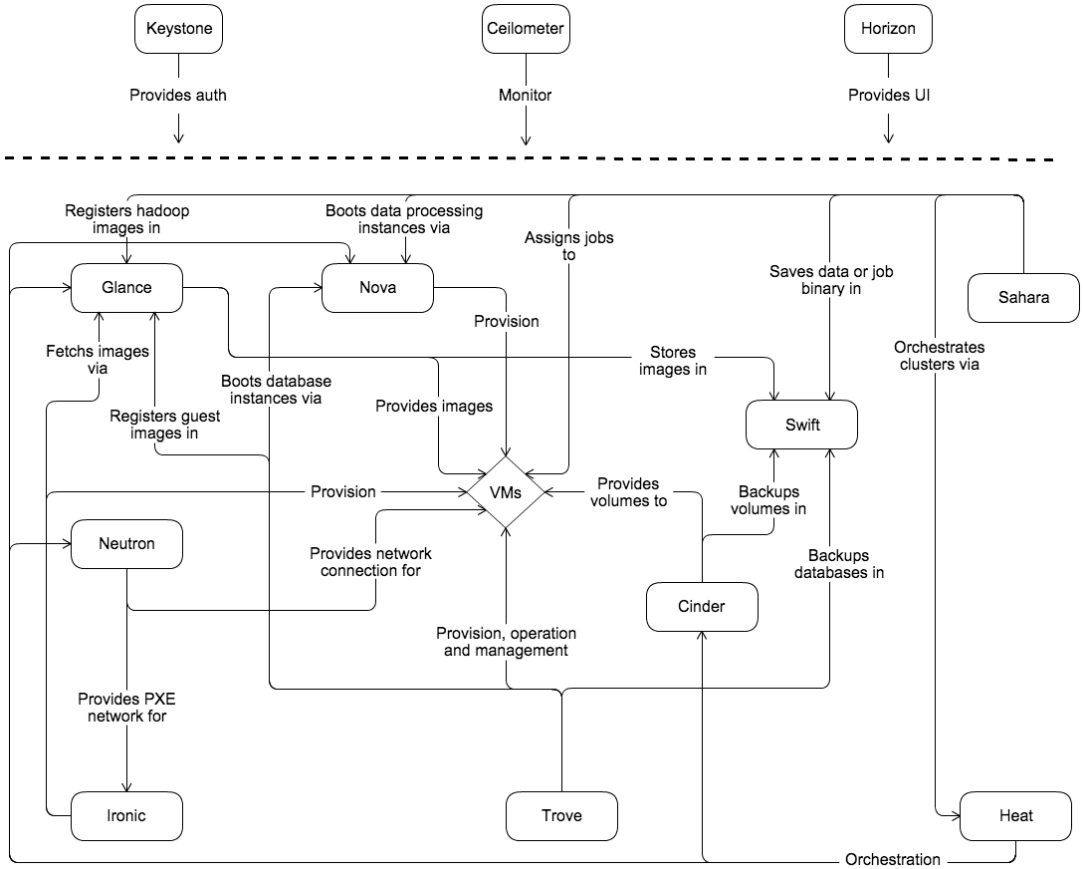


Figura 7 – Arquitetura conceitual do OpenStack.

Acessado em: https://docs.openstack.org/install-guide/get-started-conceptual-architecture.html

Destaca-se também que não é objetivo deste trabalho investigar o OpenStack, mas sim propôr e implementar um serviço de dimensionamento vertical, cujo protótipo escolhido utiliza o OpenStack como infraestrutura, mas qualquer outra plataforma de nuvem poderia ser escolhida. Entretanto, para se explicar e compreender o funcionamento do protótipo, é necessário, ao menos, entender como funciona a infraestrutura da nuvem.

A seguir estão relacionados os módulos do OpenStack que foram instalados afim de ser possível a implementação do protótipo do dimensionador:

- **Nova:** Gerencia recursos de computação. É o módulo responsável por manter funcionando as máquinas virtuais provisionadas no OpenStack. Mandatório em uma instalação mínima;
- **Neutron:** Módulo que gerencia redes e conexões para ambientes de computação virtuais no OpenStack. Mantém a conectividade entre máquinas virtuais e conectividade com o mundo externo. Mandatório em uma instalação mínima;
- **Glance:** Gerencia imagens para máquinas virtuais. É capaz de descobrir, registrar e recuperar imagens. Mandatório em uma instalação mínima;
- **Keystone:** Serviço de autenticação, responsável por gerência de autenticação e acesso à nuvem. Mandatório em uma instalação mínima;
- **Horizon:** Serviço de interface, interface gráfica oficial do OpenStack. Opcional, porém foi instalado para facilitar a gerência da nuvem;
- **Ceilometer:** Serviço de telemetria, responsável por coletar e armazenar a telemetria das VMs. opcional, porém foi instalado para que o dimensionador possa atuar.

O módulo Ceilometer recebe uma atenção especial, pois ele realiza a telemetria das máquinas virtuais dentro do OpenStack e salva em um banco de dados externo, para posterior consulta. O dimensionador utiliza o Ceilometer para colher informações de uso das VMs e assim tomar as decisões de alteração de recursos.

Como mostrado na Figura 5, é necessário uma forma com que o dimensionador se comunique com a infraestrutura da nuvem. Isto é feito pelo Agente através de algumas SDKs e APIs. Uma API (*Application Programming Interface*) é um conjunto de bibliotecas, métodos e funções que permitem a integração de um aplicativo com outros aplicativos externos. Uma SDK (*Software Development Kit*) é um *software* usado para desenvolver aplicativos para um sistema operacional.

Neste contexto, para o dimensionador ser capaz de se comunicar com o OpenStack, foram necessários dois itens: a SDK *shade* (OPENSTACK, 2017) e a API *libvirt* (LIBVIRT, 2019).

O *shade* foi desenvolvido e é mantido pela comunidade do OpenStack e é uma biblioteca usada para viabilizar interações com nuvens OpenStack (OPENSTACK, 2017). Porém, o *shade* não é capaz de realizar operações de dimensionamento vertical a quente, isto é, a troca de recursos, aumento ou redução, sem a necessidade da máquina virtual ser desligada ou reiniciada.

Dado que o dimensionamento vertical a quente não existe nas operações da nuvem OpenStack, torna-se necessário o uso da API *libvirt*. Com a *libvirt* (LIBVIRT, 2019), é possível acessar a plataforma virtualizadora e alterar os recursos da máquina virtual de

maneira dinâmica. O OpenStack utiliza a *libvirt* em suas operações, o que tornou mais simples a implementação do protótipo.

O *libvirt* e o *shade* são complementares. Com o *shade* é possível obter informações de uso de recursos de uma VM e sua localização dentro da nuvem. Localização significa em qual servidor físico a VM está hospedada. Com essas informações em mãos, é possível o *libvirt* atuar com mais facilidade.

Por último, na Figura 5, é mostrado que entre Cliente, Gerente e Agente, há trocas de mensagens. O barramento de mensagens usado no protótipo foi modelado utilizando o RabbitMQ. O RabbitMQ é capaz de montar um barramento de mensagens rápido e fácil e em ambientes heterogêneos (PIVOTAL, 2019).

Nos Subcapítulos 3.3, 3.4 e 3.5 é mostrado com mais detalhes como estas tecnologias são utilizadas no protótipo.

3.3 Cliente

O Cliente opera como porta de entrada para o dimensionador. Com o Cliente, é possível enviar mensagens para o Gerente, contendo comandos desejados pelo usuário. Todas as mensagens transmitidas pelo Cliente são no formato JSON e podem ser vistas no Apêndice A. O Código A.1 mostra a mensagem para ligar o dimensionamento para uma VM, enquanto que o Código A.2 mostra uma mensagem para desligar o dimensionamento para uma VM.

O Cliente permite que o usuário realize duas operações básicas: Iniciar o dimensionamento de uma lista de VMs, ou parar o dimensionamento de uma lista de VMs. Ainda é possível para o usuário configurar uma janela de tempo. Uma discussão sobre esta janela de tempo é feita nos Subcapítulos 3.5 e 3.6. Em resumo, a janela de tempo indica a quantidade de medidas que o dimensionador deve considerar para calcular o uso de recursos de uma VM.

O Cliente permite também que o usuário configure limites de uso de recursos para uma máquina virtual. Configurar os limites de uso significa dizer em qual momento o dimensionador deve atuar, ou seja, se uma VM possuir um uso de CPU ou memória maior que o limite configurado, o dimensionador atua entregando mais CPU ou memória.

A Figura 8 mostra o fluxograma do Cliente. O fluxograma não apresenta laços nem tomadas de decisões. O processo consiste em: o usuário entra com os dados desejados, os quais são o comando, janela de tempo e limites de uso para CPU e memória. Após a entrada de dados ser finalizada pelo usuário, o Cliente compila ou constrói a mensagem e a envia para o Gerente.

O algoritmo 1 ilustra como o Cliente funciona. Na Linha 2 do Algoritmo 1, inicializa-

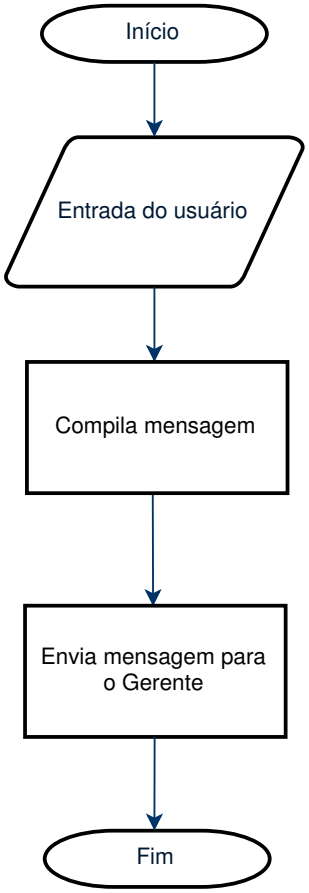


Figura 8 – Fluxograma do componente Cliente.

se uma nova mensagem com os campos *lista_instancia*, *comando*, *janela_tempo* e *limites* vazios. Na Linha 3, o algoritmo adiciona no campo *lista_instancia* da mensagem *msg*, a lista dos nomes das VMs *li*. Na Linha 5, o algoritmo adiciona o comando *c* na mensagem *msg*, podendo este ser início ou parada do dimensionamento das VMs contidas em *li*. Na linha 4 o algoritmo insere na mensagem *msg*, o valor da janela de tempo *t*. E na linha 6, o algoritmo insere os limites de uso *lim* na mensagem *msg*.

Por fim, na Linha 7, o Cliente envia a mensagem *msg* para o componente Gerente.

Algoritmo 1: Algoritmo Cliente.

Entrada: Lista com os nomes das VMs *li*, comando do usuário *c*, janela de tempo *t* e limites de uso *lim*.

Saída: mensagem *msg* construída e transmitida.

```
1 início
2     msg ← Cria_Msg()
3     Insere(msg, "lista_instancia", li)
4     Insere(msg, "comando", c)
5     Insere(msg, "janela_tempo", t)
6     Insere(msg, "lim", lim)
7     Transmite(msg)
8 fim
```

O esqueleto da mensagem enviada pelo Cliente pode ser visto a seguir:

```
{
    'lista_instancia':<lista_de_instancias>,
    'comando':<operacao_desejada>',
    'janela_tempo':<janela_de_tempo>,
    'lim': <limites_de_uso>
}
```

O esqueleto se difere do que é apresentado no Apêndice A. Enquanto que os Códigos A.1 e A.2 mostram as mensagens criadas e enviadas pelo Cliente, o esqueleto descrito serve para ilustrar o funcionamento do algoritmos.

3.4 Gerente

O Gerente é o componente principal do protótipo. Nele, consta a relação de quais máquinas virtuais estão sendo monitoradas, bem como as tomadas de decisão de dimensionamento.

Para realizar a análise das VMs, o Gerente mantém uma lista atualizada das máquinas virtuais que estão sendo monitoradas. Cada elemento desta lista contém informações sobre o estado de uma máquina virtual, tais como: nome, uso de recursos, quantidade máxima e mínima de recursos permitidos para a VM e limites de uso.

Limites de uso são parâmetros que indicam o ponto em que se deve aumentar ou reduzir recursos. O conceito de limite varia de uma máquina virtual para outra. Por exemplo, uma VM pode considerar que a utilização de memória é alta quando estiver igual ou superior 75% do total disponível, ao passo que outra VM pode considerar que essa utilização é alta quando estiver em 85% ou superior. O Gerente avalia cada caso isoladamente, permitindo que cada VM seja dimensionada de uma maneira personalizada.

O componente Gerente pode ser subdivido em 2 partes: uma responsável pela gerência das máquinas virtuais e outro responsável pela operação nas máquinas virtuais.

As mensagens trocadas pelo Gerente, aquelas vindas do componente Cliente e as mensagens trocadas com o Agente, são todas no formato JSON. E um exemplo de cada uma delas pode ser vista no Apêndice A.

A Figura 9 ilustra o fluxograma do componente Gerente. O Gerente inicia coletando uma mensagem e em seguida, processa a mensagem. Processar uma mensagem do Cliente, significa inserir uma nova VM na lista de gerenciamento de máquinas virtuais do Gerente, no caso de um comando de início de dimensionamento. Ou remover uma VM da lista de gerenciamento, no caso de um comando de parada.

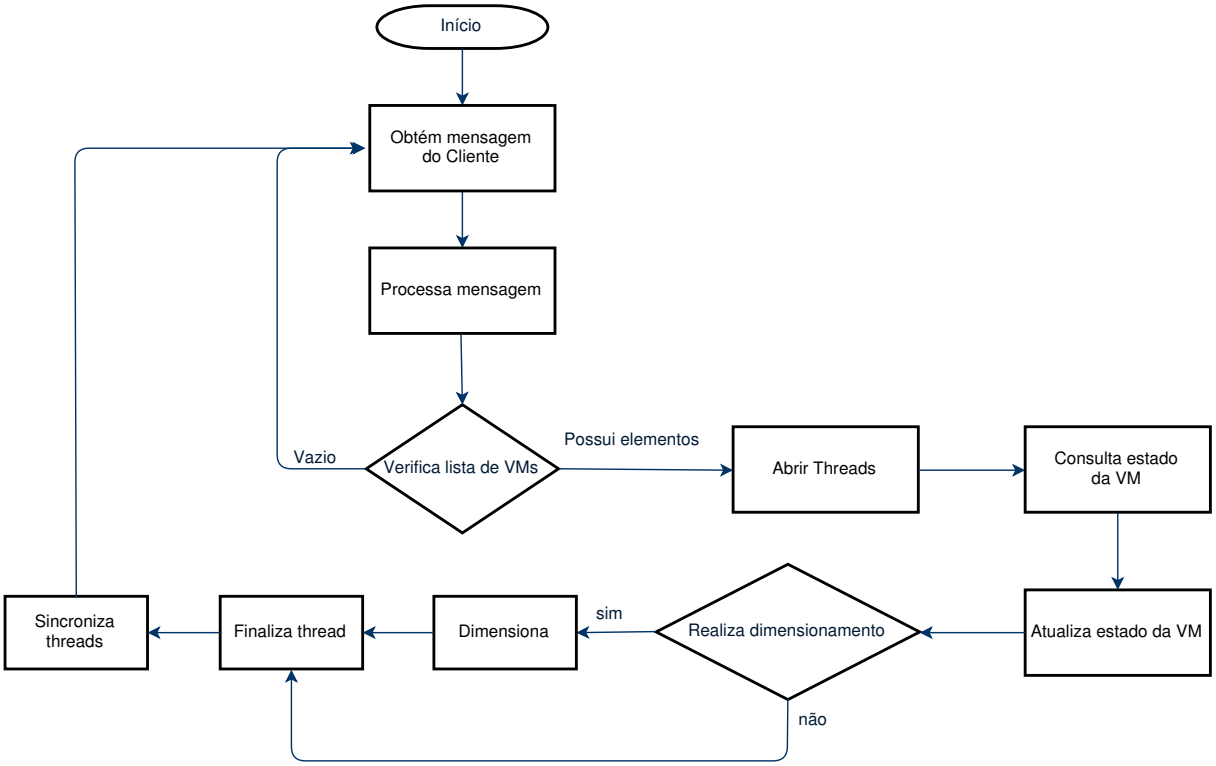


Figura 9 – Fluxograma do componente Gerente.

Após processamento da mensagem do Cliente, o Gerente verifica se a lista de gerenciamento possui VMs. Caso não possua, o Gerente volta para o início do fluxo e coleta uma nova mensagem do Cliente. Se a lista possuir ao menos 1 VM, o Gerente continua no fluxo de operação. O próximo passo consiste em criar *threads* de execução, uma para cada VM da lista de gerenciamento. Ao se criar *threads* a execução do Gerente passa a trabalhar em paralelo, o que garante velocidade no gerenciamento das máquinas virtuais. Em cada *thread*, o Gerente executa os seguintes passos: consultar uma VM, atualizar estado, analisar, dimensionar e finalizar.

Na consulta, o Gerente envia uma mensagem para o Agente, requisitando informações acerca de uma VM e com o resultado, o Gerente atualiza o estado da VM dentro da lista de gerenciamento. Com o estado da VM, isto é, com os dados referentes à quantidade de recursos alocados e uso em percentagem de recursos, o Gerente é capaz de decidir se deve dimensionar para cima ou para baixo os recursos da VM. A decisão leva em conta os limites de uso definidos pela mensagem do Cliente.

Na tomada de decisão para dimensionar, 2 caminhos são possíveis: dimensionar ou não dimensionar. No caso de dimensionar, que é o caso aonde o uso de recursos está acima ou abaixo dos limites estabelecidos pelo cliente, o Gerente envia uma mensagem para o Agente, requisitando o aumento (dimensionar para cima) ou redução (dimensionar para baixo).

As saídas da tomada de decisão levam à finalização e sincronismo da *thread*.

Finalizar significa que operações em uma VM (consultar e dimensionar) ocorrerão somente no próximo ciclo e sincronizar significa esperar que todas as *threads* cheguem no mesmo ponto. Por fim, o Gerente volta ao ponto inicial de obter uma mensagem do Cliente, e todo o ciclo descrito se repete novamente.

O Algoritmo 2 mostra o funcionamento do Gerente. Na Linha 2, o algoritmo inicia uma lista de máquinas virtuais como uma lista vazia. O laço da Linha 3 até a Linha 14 é executado ininterruptamente, pois o Algoritmo 2 não é capaz de prever quando que o Cliente enviará uma mensagem pedindo para iniciar o dimensionamento de uma VM.

Na Linha 4 o Gerente obtém uma mensagem de inicio (A.1) ou parada (A.2) vinda do Cliente. Três possíveis situações podem ocorrer: o Cliente envia um comando de início, bem como uma lista de nomes de máquinas virtuais, ou o Cliente envia um comando de parada para uma lista de nomes de máquinas virtuais ou simplesmente não existe mensagem pois o Cliente não enviou uma.

Na Linha 5, o Gerente verifica se a mensagem do Cliente *cliente_msg* não é vazia. Caso verdade a função `atualiza_lista` na Linha 6 insere ou remove VMs da lista de máquinas virtuais, respectivamente. Caso contrário o algoritmo continua sem atualizar a lista de VMs. Ao inserir uma nova VM na lista de máquinas virtuais, os dados vindos do cliente, como limites de uso e janela de tempo são também adicionadas à lista de máquinas virtuais *lista_instancia* para facilitar a execução do Gerente.

No laço da Linha 8 até a Linha 11, o Gerente realiza uma chamada do Algoritmo 3, para cada *vm* pertencente à lista de VMs *lista_instancias*. na Linha 9 o Gerente cria uma *thread* e na linha 10 o Gerente inicializa a execução da *thread*, que irá executar o Algoritmo 3, tendo como parâmetro uma máquina virtual *vm*. Na linha 12 o Gerente aguarda todas as *threads* sincronizarem, ou seja, aguarda todas as *threads* finalizarem suas execuções antes de continuar. Na linha 13, o Gerente aguarda um tempo para a telemetria da nuvem obter uma atualização no estado das máquinas virtuais dentro dela. O Gerente não envia nenhum tipo de comando ou requisição para a nuvem. Este artifício serve apenas para garantir o bom funcionamento do Gerente. Se o Gerente operar mais rápido do que a telemetria da nuvem, o Gerente irá dimensionar uma VM de maneira errada.

O Algoritmo 3 tem por finalidade analisar uma máquina virtual e em caso de constatação de utilização fora dos limites estabelecidos pelo Cliente, realizar o dimensionamento vertical, através de troca de mensagens com o componente Agente (tratado no subcapítulo 3.5). Como entrada, o algoritmo recebe uma máquina virtual, e como saída, é o estado mais recente da máquina virtual.

Na Linha 2, o Algoritmo 3 envia uma mensagem para o Agente requisitando o estado da máquina virtual *vm* e aguarda uma resposta. Um exemplo de mensagem de requisição de estado pode ser visto no Código A.3.

Algoritmo 2: Algoritmo Gerente.

```
1  início
2      lista_instancias ← List()
3      enquanto verdade faça
4          cliente_msg ← obtem_cliente_msg()
5          se cliente_msg ≠ ∅ então
6              atualiza_lista(cliente_msg, lista_instancias)
7          fim se
8          para todo vm ∈ lista_instancias faça
9              nova_thread ← Cria_Thread(Dimensiona_Instancia,vm)
10             nova_thread.start()
11         fim para todo
12         sincroniza_threads()
13         espera_telemetria()
14     fim enqto
15 fim
```

Em seguida, o Algoritmo 3 verifica, na Linha 3, se os recursos da máquina virtual *vm* estão acima do limite superior de uso definido pelo Cliente. O Algoritmo 4 ilustra como é feito esta verificação. Em caso positivo, o Algoritmo 3 então envia uma mensagem para o Agente na Linha 4, indicando que a máquina virtual *vm* deve receber mais recursos. Esses recursos podem ser memória, processador ou ambos. No Apêndice A, existem dois exemplos de mensagens que o Gerente envia para o Agente. O Código A.5 mostra um exemplo de mensagem para dimensionar CPU de uma VM e o Código A.7 mostra o mesmo para memória.

Em uma abordagem semelhante, na Linha 6, o Algoritmo 3 faz uma verificação para o limite inferior de uso definido pelo Cliente. O Algoritmo 5 ilustra como é feito esta verificação. Também em caso positivo, o algoritmo envia uma mensagem para o Agente, na Linha 7, requisitando a remoção de recursos da máquina virtual *vm*. Os Códigos A.5 e A.7 podem ser utilizados para aumento ou redução de recursos. Por fim, o Algoritmo 3 atualiza a lista de VMs *lista_intancias*, vista no Algoritmo 2 com o estado mais atual da VM *vm*.

Em ambos os casos, inserção e remoção de recursos é feito de maneira gradual. Ou seja, são usados quantidades pré-definidas. Os valores são: 1 *core* para processador e 512MB de memória RAM.

Os Algoritmos 4 e 5 mostram como é feito o teste de dimensionamento feito pelo Gerente. No Algoritmo 4, é analisado o caso do limite superior de uso de recursos, ao passo

Algoritmo 3: Algoritmo Dimensiona_Instancia.

Entrada: Uma máquina virtual *vm*

```
1  início
2      status_vm ← consulta_vm(obtem_status,vm)
3      se teste_scale_up(status_vm) então
4          status_vm ← realiza_dimensionamento(scale_up,vm)
5      fim se
6      se teste_scale_down(status_vm) então
7          status_vm ← realiza_dimensionamento(scale_down,vm)
8      fim se
9      autaliza_lista(status_vm, lista_instancias)
10 fim
```

que o Algoritmo 5 verifica o caso do limite inferior de uso de recursos.

Algoritmo 4: Algoritmo teste_scale_up.

Entrada: Status de uma máquina virtual $status\_vm$ e limiares lim

Saída: Tupla dim contendo booleanos para o dimensionamento de CPU e memória

```
1  início
2      dim.cpu ← falso
3      dim.mem ← falso
4      se status_vm.cpu > lim.cpu_up então
5          se status_vm.cont_cpu > CE então
6              dim.cpu ← verdade
7              status_vm.cont_cpu ← 0
8          fim se
9          status_vm.cont_cpu ← status_vm.cont_cpu + 1
10     fim se
11     se status_vm.mem > lim.mem_up então
12         se status_vm.cont_mem > CE então
13             dim.cpu ← verdade
14             status_vm.cont_mem ← 0
15         fim se
16         status_vm.cont_mem ← status_vm.cont_mem + 1
17     fim se
18     se status_vm.cont_cpu > CE então
19         status_vm.cont_cpu ← 0
20     fim se
21     se status_vm.cont_mem > CE então
22         status_vm.cont_mem ← 0
23     fim se
24     retorna dim
25 fim
```

Nas linhas 2 e 3 do Algoritmo 4, é inicializado a tupla *dim*, que indica se CPU e/ou memória devem ser dimensionadas ou não. O valor inicial é $Falso$ indicando o não dimensionamento. Na linha 4 o algoritmo verifica se o uso corrente de cpu *status_vm.cpu* é maior que o limite estabelecido pelo usuário *lim.cpu_up*. Em caso positivo, o algoritmo verifica então a contagem de estados na linha 5. Essa verificação permite que o Dimensionador não reaja em situações aonde só ocorre um pico/vale de uso de recursos. E em caso verdade, na linha 6 a tupla *dim* é atualizada com o valor *verdade*, o que indica que o dimensionador deve entregar mais CPU para a VM. E o contador de estados para cpu é atualizado com o valor 0 na linha 7. Na linha 9 é incrementado a contagem de de estados para CPU.

De maneira semelhante à CPU, o teste de dimensionamento é feito para memória da linha 11 até a linha 16. Das linhas 18 até a linha 22, o Algoritmo 4 verifica se pode restar os contadores de estado para CPU e memória, afim de manter consistência na execução do algoritmo. Por fim o algoritmo retorna o resultado da análise na linha 24.

O Algoritmo 5 mostra como é feito a verificação do limite inferior de uso. A operação é semelhante ao do Algoritmo 4. A diferença reside nas linhas 4 e 11, aonde o algoritmo

verifica se a VM possui uso de recursos abaixo do limite inferior estabelecido pelo usuário.

Algoritmo 5: Algoritmo teste_scale_down.

Entrada: Status de uma máquina virtual *status_vm* e limiares *lim*

Saída: Tupla *dim* contendo booleanos para o dimensionamento de CPU e memória

```
1  início
2      dim.cpu ← falso
3      dim.mem ← falso
4      se status_vm.cpu < lim.cpu_up então
5          se status_vm.cont_cpu > CE então
6              dim.cpu ← verdade
7              status_vm.cont_cpu ← 0
8          fim se
9          status_vm.cont_cpu ← status_vm.cont_cpu + 1
10     fim se
11     se status_vm.mem < lim.mem_up então
12         se status_vm.cont_mem > CE então
13             dim.cpu ← verdade
14             status_vm.cont_mem ← 0
15         fim se
16         status_vm.cont_mem ← status_vm.cont_mem + 1
17     fim se
18     se status_vm.cont_cpu > CE então
19         status_vm.cont_cpu ← 0
20     fim se
21     se status_vm.cont_mem > CE então
22         status_vm.cont_mem ← 0
23     fim se
24     retorna dim
25 fim
```

3.5 Agente

O Agente é o componente do protótipo que lida diretamente com a infraestrutura. Ele é responsável por realizar operações na nuvem, porém não detém as tomadas de decisões, que são feitas pelo Gerente. O funcionamento do Agente consiste em coletar uma mensagem do RabbitMQ, vinda do Gerente, processar, enviar a resposta e aguardar novas mensagens.

As Mensagens que o Agente recebe são oriundas do Gerente. Exemplos de mensagens

podem ser vistas no Apêndice A. O Agente trabalha com 2 tipos básicos de mensagens: uma de consulta, cujo exemplo pode ser visto no Código A.3 e outra de troca de recursos, onde exemplos de mensagens de troca de CPU e memória podem ser vistos nos Códigos A.5 e A.7, respectivamente. Todas as mensagens do Agente, tanto recebidas quanto enviadas, estão no formato JSON.

O esqueleto de uma mensagem recebida pelo Agente possui o seguinte formato:

```
{
    'comando':<operacao_desejada>,
    'parametros':[<lista_de_parametros>]
}
```

Na mensagem acima, a chave *comando* representa a operação do Gerente que deve ser executado pelo Agente, podendo ser uma consulta ao estado de uma VM ou a realização do aumento/redução de CPU e memória de uma VM. A chave *parametros* contém a lista de parâmetros necessários para executar a operação dentro da chave *comando*. Tais parâmetros podem ser o nome de uma máquina virtual, a janela de tempo e descritor de uma VM para realização de aumento/redução de recursos.

Uma vez executado, o Agente deve responder o Gerente com o resultado da operação e o esqueleto da mensagem de resposta é como segue:

```
{
    'comando':<operacao_desejada>,
    'parametros':[<lista_de_parametros>],
    'resultado': <resultado_da_operacao>
}
```

A mensagem de resposta possui o mesmo formato da mensagem recebida, acrescida da chave *resultado*, que contém o resultado da operação executada e que deve ser devolvida ao Gerente.

A Figura 10 ilustra o fluxograma do Agente. O componente atua em um laço infinito, aguardando mensagens vindas do Gerente. Ao receber uma mensagem, o Agente verifica qual é a mensagem, podendo ser de dois tipos: uma consulta ou atualização. Após processar a mensagem, isto é, executar o comando contido nela, o Agente compila o resultado e envia de volta para o Gerente. Após devolver o resultado para o Gerente, o Agente retorna ao estado inicial, para aguardar uma nova mensagem do Gerente.

Como dito, duas operações são executadas pelo Agente: consulta e atualização. Para cada operação, existe também na mensagem, qual o alvo da operação, isto é, qual máquina virtual deve ser consultada ou atualizada. No caso de uma consulta, o Agente coleta informações acerca da máquina virtual. Tais informações são a quantidade de CPU

e memória que a VM possui, bem como o uso em porcentagem de CPU e memória.

No caso de uma operação de atualização, o Agente modifica a quantidade de recursos da máquina virtual, podendo ser aumento de CPU/memória, ou redução de CPU/memória. Neste ponto, cabe ao Gerente decidir quando alterar os recursos e a responsabilidade de modificar fica à cargo do Agente. Tanto consultas quanto atualizações, o Agente comunica diretamente com a infraestrutura, sendo neste trabalho representado pelo OpenStack.

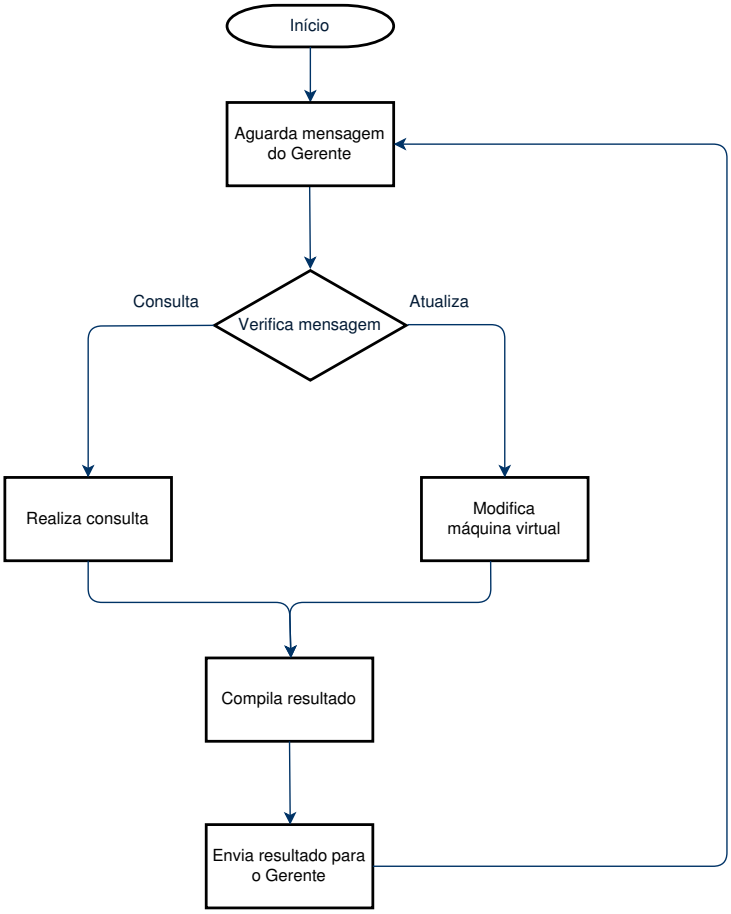


Figura 10 – Fluxograma do componente Agente.

No Agente se encontram a SDK *shade* para comunicação com o OpenStack, a API *libvirt* para acesso direto das VMs. Conforme dito anteriormente, por meio dessa API são executadas as operações que o OpenStack não provê, tais como o dimensionamento vertical e a conexão com o RabbitMQ. O algoritmo do Agente pode ser visto no Algoritmo 6.

O Algoritmo 6 mostra como o Agente opera. Seu funcionamento consiste de um laço infinito da Linha 2 até a Linha 8. Na Linha 3, o Agente obtém uma mensagem do RabbitMQ, oriunda do Gerente (visto no subcapítulo 3.4).

Após obter uma mensagem, o Agente decodifica na Linha 4, descobrindo assim, qual ação deve executar. Três ações são possíveis: obter estado da VM, dimensionar CPU e dimensionar memória. Na Linha 5, o Agente executa o comando vindo do Gerente. Na

Algoritmo 6: Algoritmo Agente

```
1 início
2     enquanto verdade faça
3         msg ← obtem_msg()
4         commando ← decodifica_msg(msg)
5         resultado ← executa_comando(comando)
6         envia_msg(resultado)
7         espera_por_evento()
8     fim enqto
9 fim
```

Linha 6 o Agente envia o resultado da operação de volta para o Gerente e na Linha 7 o Agente aguarda novas mensagens do Gerente.

Uma operação fundamental do Agente é a de consulta ao estado de uma máquina virtual. Com ela é possível determinar o uso de recursos de uma VM e com o cálculo do uso, permite ao Gerente tomar decisões acerca do estado da VM. No Subcapítulo 3.6 é mostrado como que o Agente realiza o cálculo de uso de recursos para uma VM.

3.6 Dimensionador e Telemetria

Um ponto importante acerca do dimensionador é a forma como ele responde à telemetria da nuvem, pois a gerência dos recursos de uma máquina virtual depende de como o seu uso é interpretado.

Este subcapítulo trata de como o dimensionador calcula o corrente de uma máquina virtual. A partir do cálculo de uso, é possível para o dimensionador determinar o momento para inserir ou remover recursos.

O dimensionador divide a lista de medidas lm em duas sublistas. A divisão é feita utilizando o tamanho n da lista de medidas, multiplicado por um fator γ. A equação a seguir mostra como isso é feito:

$$k = n * \gamma \tag{3.1}$$

O fator γ pertence ao intervalo [0,1] e indica qual deverá ser o tamanho de cada sublista formada a partir da lista de medidas lm Por exemplo, se $\gamma = 0.7$ e $n = 10$, significa que a primeira sublista irá conter os 7 primeiros itens da lista lm, ao passo que a segunda sublista irá conter os 3 últimos itens da lista de medidas lm. A variável k da equação 3.1 representa a posição de divisão da lista lm.

Para cada sublista, o dimensionador calcula a média de seus valores, aplicando um peso para cada valor. As equações 3.2 e 3.3 ilustram como é feito o cálculo e podem ser

vistas a seguir:

$$up_1 = \frac{1}{k} * \sum_{i=0}^{k} V_i * \alpha, \quad \forall\ V_i \in lm \tag{3.2}$$

$$up_2 = \frac{1}{n-k} * \sum_{j=k}^{n} V_j * \beta, \quad \forall\ V_j \in lm \tag{3.3}$$

As equações 3.2 e 3.3 tratam de cada subdivisão da lista de medidas lm. Sendo que a equação 3.2 trata da primeira sublista, que contém as medidas mais antigas obtidas pelo dimensionador, enquanto que a equação 3.3 trata das medidas mais recentes da lista lm.

Os pesos α e β, das equações 3.2 e 3.3 são aplicados para cada medida da lista lm. O dimensionador proposto considera que o peso α seja menor do que o peso β, indicando que as medidas mais recentes tem prioridade sobre as medidas mais antigas, porém as antigas não devem ser totalmente descartadas. Ainda sim, a condição a seguir deve ser satisfeita:

$$\alpha + \beta = 1 \tag{3.4}$$

Por fim, a equação a seguir mostra que o uso de recurso calculado pelo dimensionador é feito realizando a soma dos usos parciais vistos anteriormentes nas equações 3.2 e 3.3:

$$ur = up_1 + up_2 \tag{3.5}$$

No capítulo 4, mais especificamente no subcapítulo 4.1 é mostrado em como se chegou nesta definição de tratamento para as medidas de uso de recursos, bem como estudos de caso utilizando o modelo proposto.

4 Resultados e Discussão

Neste capítulo é apresentado a avaliação do dimensionador descrito no Capítulo 3. Os resultados obtidos em cada avaliação serão discutidos em cada subcapítulo. O subcapítulo 4.1 mostra o estudo feito para ajustar a escolha e tratamento da janela de tempo pelo dimensionador. O Subcapítulo 4.2 é focado em mostrar o funcionamento de todo o processo de dimensionamento. No Subcapítulo 4.3 é mostrado o uso do dimensionador para dimensionamento de um servidor *web*. No Subcapítulo 4.4 é mostrado os resultados da avaliação de uma aplicação de um robô em um espaço inteligente.

4.1 Tratamento das medidas pelo Dimensionador

Neste subcapítulo, é mostrado como diferentes formas de tratar afetam na avaliação de uso de recursos de uma VM. Os estudos foram feitos com base em um mesmo experimento.

O experimento base consiste em uma máquina virtual dentro de uma nuvem OpenStack. A máquina Virtual possui um gerador de carga sintética cujo objetivo é consumir recursos de processador e/ou memória. O gerador é baseado na ferramenta *stress-ng*[1].

Para o gerador de carga funcionar são necessários alguns parâmetros. O primeiro deles indica qual recurso a ser consumido, podendo ser processador ou memória. Depois, é definido o tempo total em que o gerador irá funcionar. Por fim, também é passado o tempo de operação e o tempo de espera. Tempo de operação é o tempo em que o gerador de carga permanece consumindo o recurso e tempo de espera é o tempo em que o gerador de carga fica aguardando até a próxima bateria de carga de uso.

Um exemplo de entrada para o gerador é a tupla (*cpu*,60,15,3), o que significa que o gerador irá consumir processador (*cpu*), gerando carga por 15 segundos e ficando ocioso por 3 segundos. Este procedimento então é repetido por um total de 60 segundos. O gerador busca sempre consumir o máximo disponível dentro todos os recursos solicitados.

Para o gerador, foram configurados os seguintes parâmetros: (*cpu*,85,15,5). O Cliente do dimensionador foi configurado para utilizar como limites máximo e mínimo os seguintes valores: 75.0% e 25.0%, respectivamente. Estes valores são fixados como valores padrão utilizados pelo dimensionador. A configuração da nuvem utilizada para hospedar a VM segue:

[1] https://wiki.ubuntu.com/Kernel/Reference/stress-ng

- Nó controlador com 32GB de memória RAM, processador Intel® Xeon® E3-1240 de 3.70GHz, Sistema Operacional Ubuntu 16.04;
- 2 Nós de computação com 32GB de memória RAM, processador Intel® Xeon® E3-1240 de 3.70GHz, Sistema Operacional Ubuntu 16.04;
- 1 Nó de computação com 128GB de memória RAM, processador Intel® Xeon® E5-2650 de 2.20GHz, Sistema Operacional Ubuntu 16.04;

O primeiro resultado pode ser visto na Figura 11. A figura é subdivida em 2 partes, sendo a parte superior monstrando o uso de CPU e a inferior mostrando a quantidade de CPU inserido ou removido na máquina virtual. Em ambas as subfiguras, o eixo x representa o tempo decorrido durante o experimento, dado em segundos.

A Figura 11 mostra como o Dimensionador reage, quando configurado para considerar o instante atual da condição da máquina virtual. Nota-se que o Dimensionador atual imediatamente no momento em que o uso de CPU está acima ou abaixo dos limites de uso estabelecidos. Isso faz com que a VM possa ficar instável, pois o dimensionador não é capaz de estabilizar a quantidade de CPU.

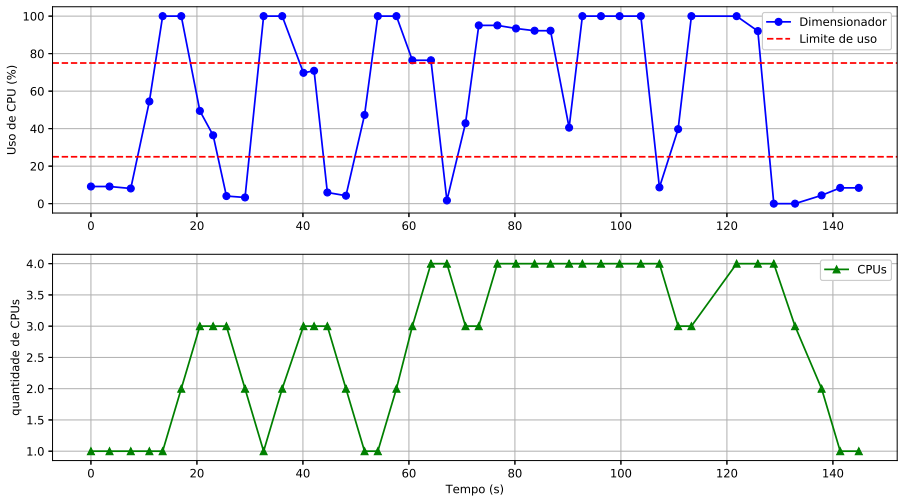


Figura 11 – Análise do dimensionador para carga sintética do instante de consumo de processador.

A próxima figura, a Figura 12 possui configuração semelhante à feita para a Figura 11. Com a diferença de que o Dimensionador agora realiza uma média simples de todas medidas dentro da janela de tempo estipulada pelo Cliente. Neste experimento, a janela foi configurada para tamanho de 40 segundos.

Nota-se uma diferença na percepção de uso de CPU por parte do Dimensionador. Com o cálculo da de todas as medidas, o Dimensionador atua como se a VM sempre

estivesse com carga máxima de uso e entrega todos os recursos disponíveis. Isso torna a atuação do dimensionador como binária: ora com o mínimo para atender a VM, ora com o máximo. A premissa do Dimensionador é que a entrega de recursos seja gradual.

Outro problema com a média está exatamente na transição na quantidade de CPU. Exatamente no momento em que a alocação de recursos ocorre. Por exemplo, em um determinado momento, a VM está com 1 CPU à 100% de uso. Quando o Dimensionador insere mais 1 CPU, o uso cai para 50%. Na média, o cálculo considera os 50% atuais bem como os 100% passados. E as medidas passadas não representam mais a realidade da VM.

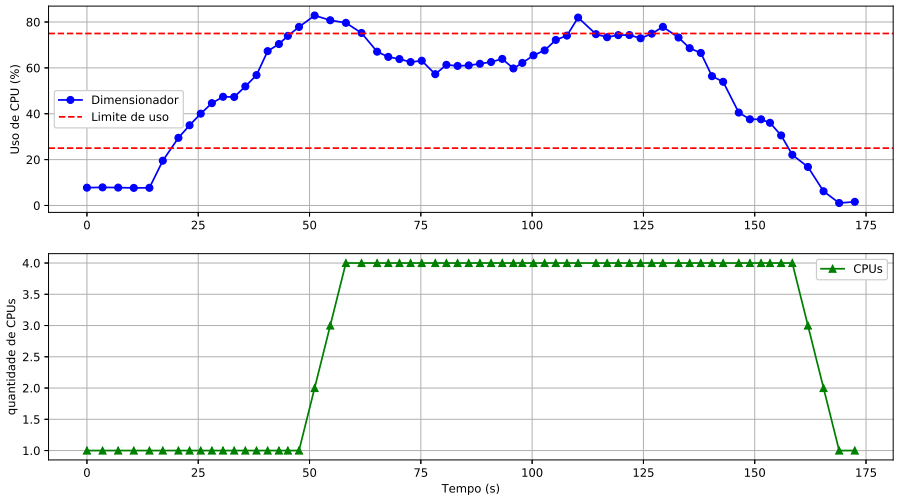


Figura 12 – Análise do dimensionador para carga sintética da média do consumo de processador.

Porém, se ignorar as medidas anteriores, o Dimensionador recai no caso visto na Figura 11. Por isso é necessário a adoção de uma estratégia em que considere o que já ocorreu com a máquina virtual, mas entregando um peso menor em detrimento ao estado atual da VM.

A Figura 13 mostra o resultado do Dimensionador utilizando a estratégia apresentada no subcapítulo 3.6, considerando as equações 3.1, 3.2, 3.3 e 3.5 para o cálculo de uso de recursos. Na figura, os seguintes parâmetros são adotados: $\gamma = 0.6$, $\alpha = 0.3$ e $\beta = 0.7$.

Algo para se notar no gráfico da Figura 13, o Dimensionador reage de maneira gradual à demanda da máquina virtual. Mas casos atípicos como o que ocorre em torno de 100 segundos de experimento, aonde o Dimensionador reagiu instantaneamente à um uso de CPU menor que o limite inferior. Para contornar esta situação, foi adicionado ao Dimensionador uma contagem de estados. Tal contagem é para averiguar se o que houve foi apenas um pico ou vale de uso de recursos.

Os subcapítulos 4.2, 4.3 e 4.4 mostram como que esta estratégia funcionam, utili-

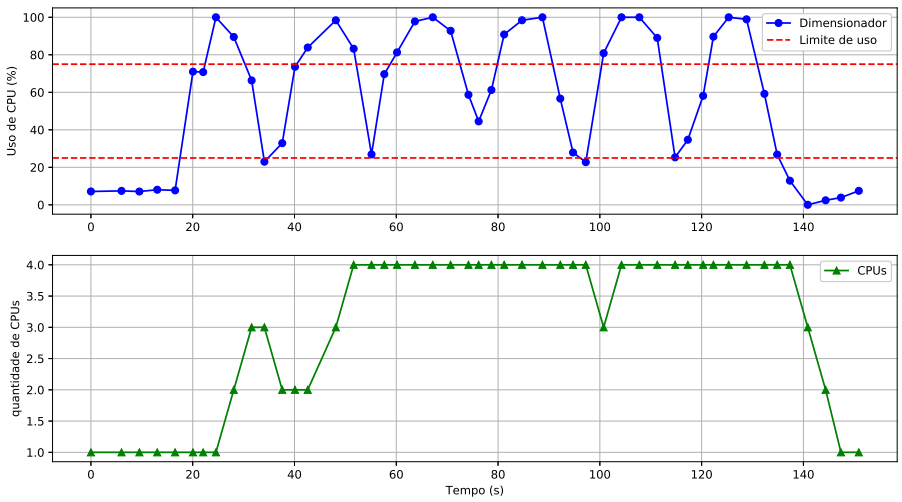


Figura 13 – Análise do dimensionador para carga sintética de consumo de processador, considerando tratamento diferenciado para medidas recentes.

zando os seguintes parâmetros das equações 3.1, 3.2, 3.3 e 3.5 e do contador de estados visto no subcapítulo 3.4 como constantes:

- $\gamma = 0.6$
- $\alpha = 0.3$
- $\beta = 0.7$
- Contagem de estados: 3

4.2 Dimensionador com um gerador de carga sintético

Nesta avaliação tem-se uma máquina virtual em uma nuvem OpenStack. E na máquina virtual, tem-se um gerador de carga sintética para consumir recursos, idêntico ao utilizado no subcapítulo 4.1.

O gerador de carga está instanciado em uma VM no OpenStack. Para este experimento, foi utilizado o OpenStack versão Pike, distribuído em quatro servidores. Suas respectivas configurações seguem:

- Nó controlador com 32GB de memória RAM, processador Intel® Xeon® E3-1240 de 3.70GHz, Sistema Operacional Ubuntu 16.04;
- 2 Nós de computação com 32GB de memória RAM, processador Intel® Xeon® E3-1240 de 3.70GHz, Sistema Operacional Ubuntu 16.04;

- 1 Nó de computação com 128GB de memória RAM, processador Intel® Xeon® E5-2650 de 2.20GHz, Sistema Operacional Ubuntu 16.04;

Os testes com carga sintéticas podem ser divididas em 2 (duas) partes:

- A primeira, que gera carga para consumo de processador;
- A segunda, que gera carga para consumo de memória.

Os testes são divididos em 2 partes para facilitar a apresentação dos resultados. Porém o dimensionador opera analisando e dimensionando processador e memória simultaneamente.

Para o gerador, foram configurados os seguintes parâmetros: (*cpu*,65,15,1). O Cliente do dimensionador foi configurado para utilizar como limites máximo e mínimo os seguintes valores: 75.0% e 25.0%, respectivamente. Estes valores são fixados como valores padrão utilizados pelo dimensionador.

O resultado da execução do dimensionador para carga sintética de processador pode ser visto na Figura 14. A figura é dividida em 2 partes, sendo a parte superior mostrando o uso de CPU e a inferior, mostrando a quantidade de CPU inseridos ou removidos na VM a partir da atuação do dimensionador. Em ambas as subfiguras, o eixo x representa o tempo decorrido durante o experimento, dado em segundos.

O experimento começa com a VM com cerca de 10% de uso de CPU. O gerador de carga então começa a agir, sempre consumindo tudo o que estiver disponível. Isso faz com que o uso de CPU chegue a 100%. A partir do momento em que o uso ultrapassa o limite superior (75%), o dimensionador toma a decisão de entregar mais CPU para a VM a fim de suprir essa demanda e reduzir o uso de processamento da VM. A subfigura inferior da Figura 14, mostra os momentos em que ocorreram o dimensionamento de CPU.

Após cerca de 80 segundos de operação, o gerador de carga libera completamente o uso do processador. Quando utilização cai abaixo do limite inferior, o dimensionador age novamente, removendo CPUs da máquina virtual sendo monitorada.

Para avaliar o uso do recurso de memória o gerador sintético foi configurado para realizar duas cargas, espaçadas por um tempo de 35 segundos. As duas cargas são usadas para mostrar o funcionamento do dimensionador para recursos de memória. A primeira carga foi feita com os parâmetros (*mem*,210,30,1) e a segunda carga foi feita com os parâmetros (*mem*,110,30,1). O cliente do orquestrador foi configurado para utilizar como limites máximo e mínimo os valores 70.0% e 30.0%, respectivamente. Estes valores foram fixados como padrões pelo dimensionador.

A Figura 15 ilustra os resultados obtidos para a carga de uso de memória. A figura também é dividida em duas partes, sendo a superior mostrando o uso de memória obtido

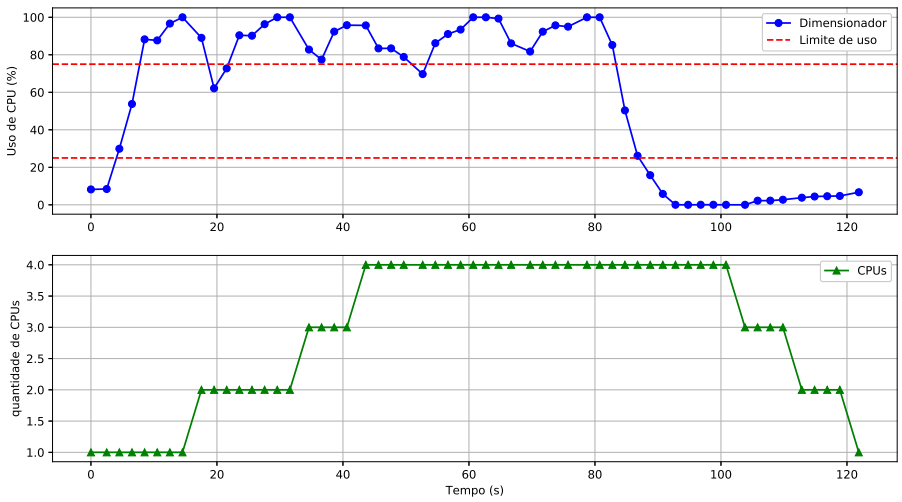


Figura 14 – Análise do dimensionador para carga sintética de consumo de processador.

monitorado pelo dimensionador, bem como os limites de uso. A figura inferior mostra a quantidade corrente de memória da VM, dado em KiB, bem como o momento em que o dimensionamento ocorre (se a quantidade aumenta ou reduz, o dimensionador atuou). Em ambas o eixo x representa o decurso do tempo durante o experimento, dado em segundos.

O experimento começa com a máquina virtual com cerca de 25% de uso. A medida que o gerador de carga começa a agir, o consumo de memória aumenta, fazendo o dimensionador atuar entregando mais memória para a máquina virtual. A quantidade de memória varia de 1 GB até 4 GB durante o experimento.

Após terminar a primeira carga, há um período de 35 segundos em que o uso de memória vai abaixo do limite inferior, fazendo o dimensionador atuar, reduzindo a quantidade de memória. Ao iniciar a segunda carga, o dimensionador reage à essa mudança, aumentando novamente a quantidade de memória. Por fim o gerador finaliza sua operação e remove recursos de memória que foram alocados para a máquina virtual.

Um ponto interessante do dimensionador é que ele não opera exclusivamente com memória ou processador e sim com ambos simultaneamente. Isto pode ser notado na Figura 16. A sua descrição é semelhante à da Figura 14, porém as curvas apresentadas são referentes ao experimento da Figura 15.

A ferramenta stress-ng, ao consumir memória, faz com que um processo de alocação contínua de memória seja executado, e com isso consome-se, também, recursos de processamento. A atuação do dimensionador ocorre tanto na memória quanto no processador, e isto pode ser observados na Figura 15 e na Figura 16. Nos momentos em que a memória está sendo requisitada o processador também está crescendo seu nível de utilização, ocorrendo um aumento (e posteriormente uma redução) no número de CPUs fornecidos à VM pelo

dimensionador.

Os resultados deste experimento mostram que o dimensionador é capaz de reagir ao aumento no consumo de processador, memória e também é capaz de agir a demandas simultâneas de processador e memória.

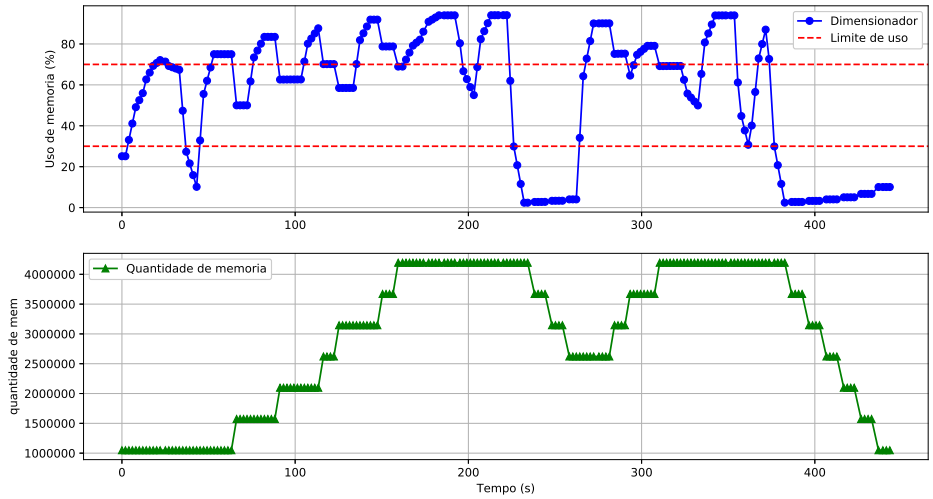


Figura 15 – Análise do dimensionador para carga sintética de uso de memória.

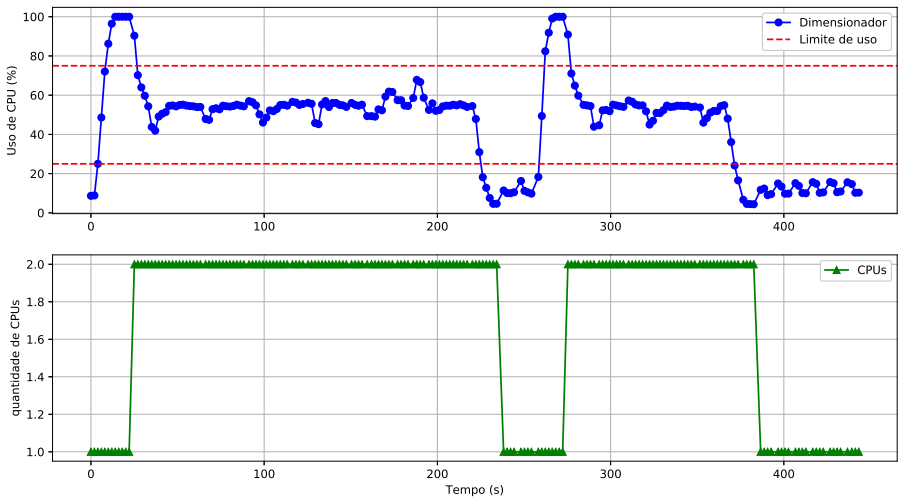


Figura 16 – Análise do dimensionador para carga sintética de uso de memória.

4.3 Caso de estudo com um servidor *web*

Este caso de estudo também faz uso de um máquina virtual em uma nuvem OpenStack. Esta VM hospeda um servidor *web*, principal alvo do monitoramento de

recursos por parte do dimensionador. Além disso, existem outras três máquinas virtuais, cada uma responsável por gerar uma determinada carga de requisições para o servidor *web*, emulando usuários simultâneos deste serviço.

Para que uma máquina virtual possa gerar requisições e emular usuários foi instalada a ferramenta Siege[2]. Como o Siege possui algumas limitações, que não cabem na discussão deste trabalho, uma única máquina virtual não seria o suficiente para gerar toda a carga necessária para estressar o uso de recursos do servidor *web*. Para emular uma quantidade de usuários suficientes para consumir recursos do serviço *web* foram utilizadas três VMs com o Siege.

O total de requisições geradas pode ser vista na Figura 17. Nela, o eixo x representa o decurso do tempo, em segundos, necessário para emitir todas as requisições de serviço. O eixo y representa a quantidade de usuários *web* que a ferramenta Siege gerou durante o experimento. Os valores do eixo y representam o somatório total dos usuários oriundos das três máquinas virtuais. As requisições foram de, no mínimo 1200 clientes até 4200 clientes tentando acessar simultaneamente o servidor *web* na VM monitorada pelo dimensionador de recursos.

A requisição de clientes representados pela Figura 17 apresenta dois momentos. No primeiro há um aumento gradual da carga de requisições. No segundo uma carga única com todos os clientes realizando requisições é utilizado.

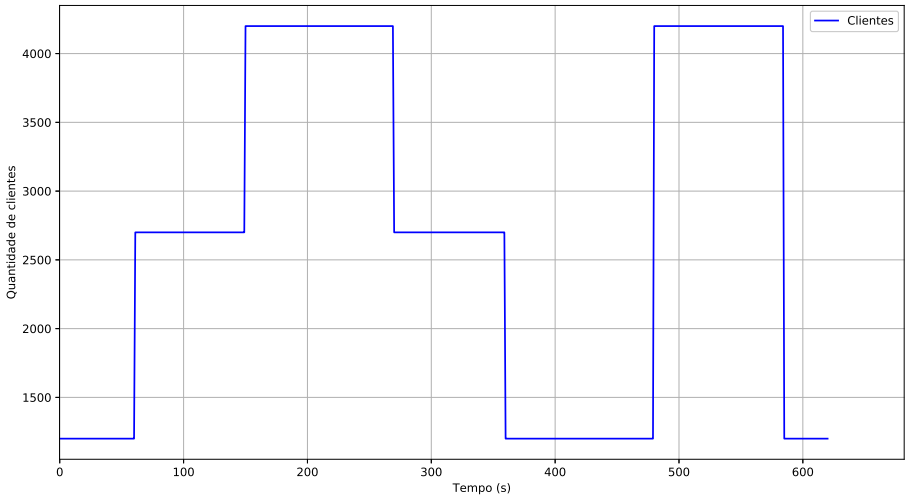


Figura 17 – Quantidade de clientes gerada para o servidor *web*.

Toda a instalação do caso de estudo foi feito no OpenStack Queens e a configuração dos servidores segue detalhada a seguir:

[2] https://www.joedog.org/siege-home/

- 1 nó controlador com 16GB de memória RAM, processador Intel® Xeon® E5-2620 v3 de 2.40GHz, Sistema Operacional CentOS 7.6;
- 1 nó de computação com 32GB de memória RAM, processador Intel® Xeon® E5-2620 v3 de 2.40GHz, Sistema Operacional CentOS 7.6;

Nota-se que este *setup* do OpenStack é diferente ao do caso tratado no subcapítulo 4.2. Isto indica que o dimensionador é agnóstico em relação à versão do OpenStack.

Por fim, o cliente do dimensionador, presente na máquina virtual do servidor *web*, foi configurado como segue: limites máximo e mínimo de cpu foram de 85% e 25%, respectivamente. Memória foi configurado como limites máximo e mínimo, os valores padrão que o dimensionador oferece, a saber 70% e 30%, respectivamente. Para este experimento é esperado que o servidor *web* fique com uso de processador menor do que 85%.

Antes de realizar o experimento, foi feito uma análise de comportamento do servidor *web*. O objetivo desta análise foi verificar como que o servidor se comporta em diferentes configurações de processador. A análise foi feita variando o número de CPUs, sem a atuação do dimensionador. Os resultados podem ser vistos na figura 18.

Na Figura 18, cada curva representa o comportamento do servidor *web*, quando este possui dois, três e quatro CPUs, sem a presença do dimensionador. Cada curva remete ao comportamento para a entrada vista na Figura 17. Pode-se observar que, quanto mais CPUs o servidor *web* possuir, menor é a sua carga de trabalho.

Porém, simplesmente entregar a maior quantidade de recurso disponível de uma única vez não é uma boa estratégia, pois uma nuvem não é exclusiva de uma aplicação. Havendo a existência de outras aplicações, entregar todos os recursos para uma única aplicação, pode causar perdas de desempenho para outras aplicações.

Um segundo motivo para uma entrega fixa e superdimensionada ser ruim, isto é, inicializar a aplicação com o máximo de recursos, é a questão de custo. Nuvens comerciais tais como Amazon (AMAZON, 2019) e Google Cloud (GOOGLE, 2019) cobram por uso de recursos. Logo é mais atrativo ter aplicações com poucos recursos em nuvens e a partir de uma certa demanda de uso, entregar dinamicamente novos recursos.

Os resultados do experimento do servidor *web* com o dimensionador pode ser visto na Figura 19. A figura pode ser dividida em duas partes, sendo a superior mostrando o uso de CPU e a inferior, mostrando a quantidade de CPUs inseridos ou removidos na VM.

Na Figura 19 é possível perceber que o dimensionador procura manter o uso de CPU abaixo do limite superior. Sempre que é necessário, o dimensionador adiciona mais CPUs para a máquina virtual. E no momento em que o uso de processador está abaixo do mínimo, o dimensionador remove CPUs da máquina virtual.

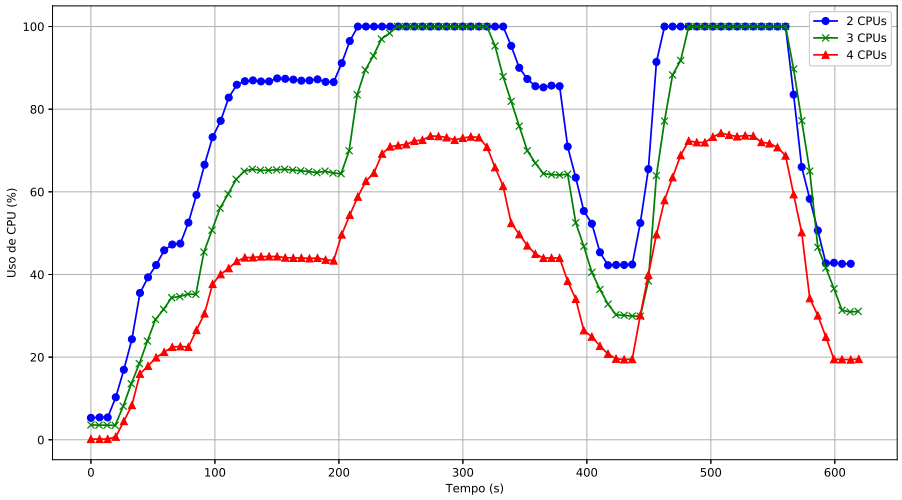


Figura 18 – Análise do uso de CPU do servidor *web* sem o dimensionador para diferentes quantidades de CPUs.

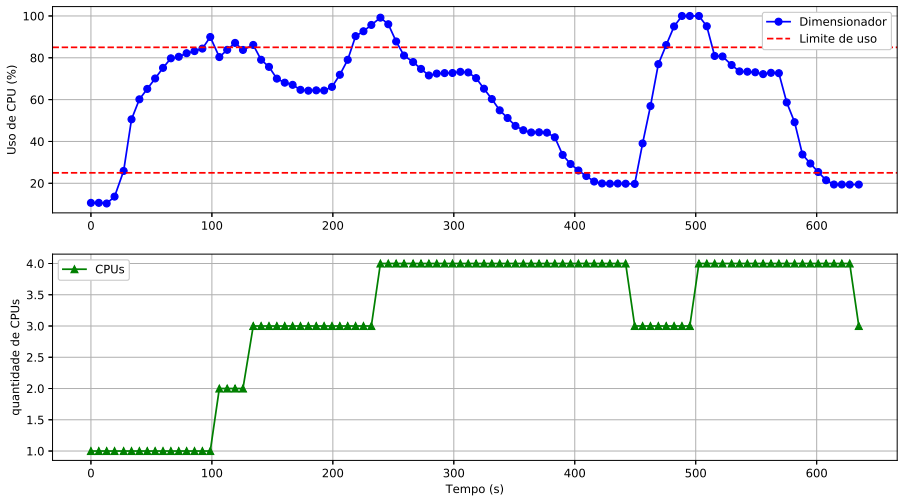


Figura 19 – Análise do uso de CPU do servidor *web* com o dimensionador.

A Figura 20 mostra o consumo de memória para o experimento do servidor *web*. Nela é possível perceber que quando o servidor ultrapassou o limite superior, mais memória foi adicionada à máquina virtual.

Em ambos os casos esse comportamento comprova, em uma situação real, a operação do dimensionador conforme descrito no subcapítulo 4.2.

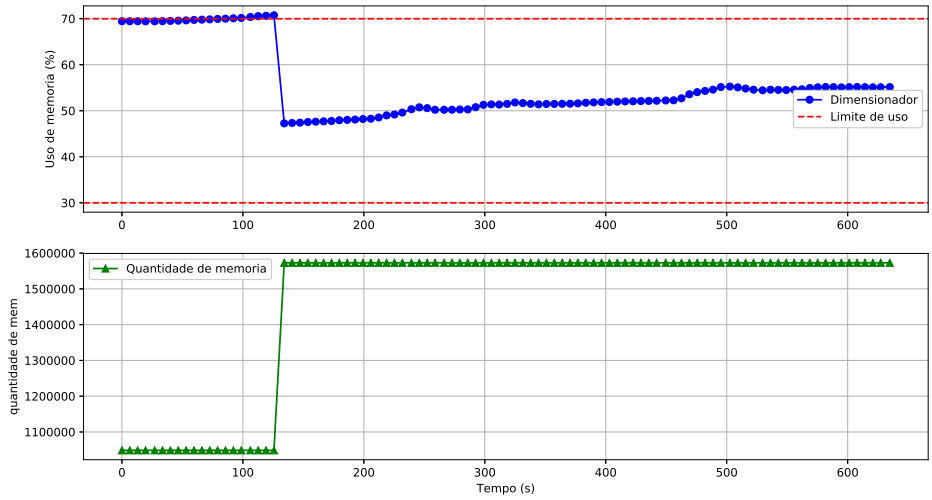


Figura 20 – Análise do uso de memória do servidor *web* com o dimensionador.

4.4 Caso de estudo de um robô em um espaço inteligente

Este estudo de caso trata do controle remoto de um robô em um espaço inteligente. O estudo de caso consiste em um robô que não é dotado de inteligência, sendo apenas capaz de se locomover e mudar de direção. Todo o processamento fica a cargo da nuvem, ou seja, os serviços do espaço inteligente ficam alocados em máquinas virtuais providas pela nuvem.

Os serviços do espaço inteligente presentes na nuvem devem garantir que o robô siga uma trajetória pré-determinada. Isto implica que se houver gargalos em algum serviço dentro da nuvem, o robô ficará fora da trajetória pré-determinada, o que configura um erro de trajetória. Além da trajetória, o robô tem um tempo determinado para completar seu percurso, sendo repetido por uma determinada quantidade de vezes.

A Figura 21 ilustra a dinâmica do espaço inteligente. O processo inicia com a câmera colhendo as imagens do robô e enviando para um serviço de processamento de imagens dentro da nuvem. Após processamento, o resultado é enviado para o serviço de localização, que verifica qual a posição atual do robô e qual deveria ser sua posição.

Um ponto crítico deste caso de estudo é o serviço de processamento de imagens (SANTOS et al., 2018), pois ele deve ser capaz de processar imagens das câmeras em diferentes FPS (*frames* por segundo), e se ele não for capaz de processar por falta de recursos, falhas de processamento acarretarão em um maior erro de trajetória do robô.

O experimento feito com o espaço inteligente foi definido com o robô seguindo uma trajetória no formato da Lemniscata de Bernoulli. Durante o experimento, o robô executa determinado número de voltas com um tempo fixo de duração por volta. A cada teste as

câmeras são atualizadas para aumentar a taxa de *frames* por segundo. As configurações são sumarizadas a seguir:

- 6 voltas, 20 segundos cada, à 5 FPS;
- 6 voltas, 20 segundos cada, à 10 FPS;
- 6 voltas, 20 segundos cada, à 15 FPS;
- 6 voltas, 20 segundos cada, à 5 FPS.

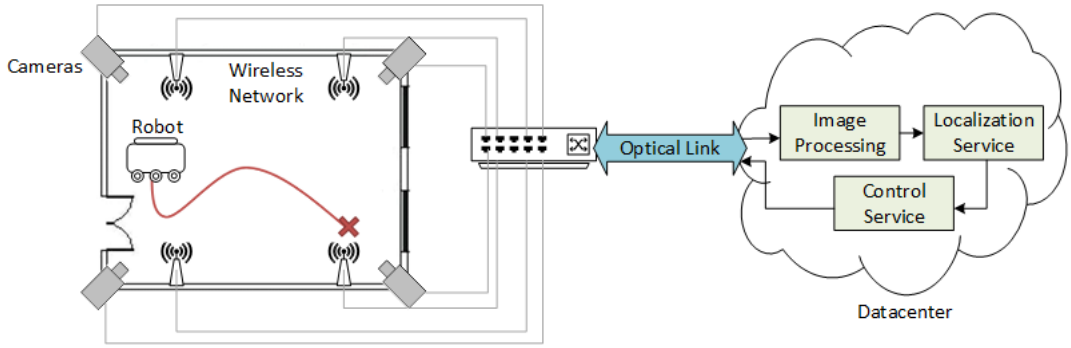


Figura 21 – Esquemático do funcionamento do espaço inteligente.

A nuvem utilizada foi o OpenStack versão Pike, distribuído em quatro servidores. Suas respectivas configurações são idênticas à nuvem do subcapítulo 4.2:

- Nó controlador com 32GB de memória RAM, processador Intel® Xeon® E3-1240 de 3.70GHz, Sistema Operacional Ubuntu 16.04;
- 2 Nós de computação com 32GB de memória RAM, processador Intel® Xeon® E3-1240 de 3.70GHz, Sistema Operacional Ubuntu 16.04;
- 1 Nó de computação com 128GB de memória RAM, processador Intel® Xeon® E5-2650 de 2.20GHz, Sistema Operacional Ubuntu 16.04;

Para o dimensionamento foi configurado o limite superior de utilização em 65% de uso de CPU e limite inferior em 45%.

A Figura 22 mostra o resultado do dimensionador atuando durante o experimento do espaço inteligente. A figura é dividida em 2 partes, sendo que a subfigura superior mostra o uso de processador e a inferior mostra a quantidade de CPUs que a máquina virtual contendo o serviço de processamento de imagens possui. Santos *et. al.* (SANTOS et al., 2018) mostrou, em seu trabalho, que a VM que possui o serviço de processamento de imagem requer uma alta demanda por recursos de processamento. Em ambas as subfiguras, o eixo x representa o tempo decorrido durante o experimento, dado em segundos.

O experimento inicia com as câmeras entregando 5 FPS para o serviço de processamento de imagens por cerca de 120 segundos. Após isso, aumenta-se o FPS para 10. Nota-se um aumento no uso de processador, ultrapassando o limite superior de uso. Isso faz com que o dimensionador atue, entregando mais CPU para a VM. Após cerca de 120 segundos, as câmeras mudam para 15 FPS. Novamente, o dimensionador atua, entregando mais CPUs para a máquina virtual.

É interessante notar que após os 180 segundos operando a 15 FPS, as câmeras voltam para 5 FPS e têm-se uma queda brusca no gráfico superior da Figura 22. Devido à queda brusca de demanda (queda de 15 FPS para 5 FPS), tem-se um caso onde existe muito recurso para pouco processamento e o uso de CPU fica abaixo do limite inferior de uso. Neste contexto o dimensionador atua removendo CPUs da VM até que a utilização fique acima do limite inferior.

É importante destacar que o dimensionador sempre procura manter o uso de recursos (processador ou memória) em um patamar que fique abaixo do limite superior e acima do limite inferior definido pelo administrador do dimensionador do recursos da nuvem.

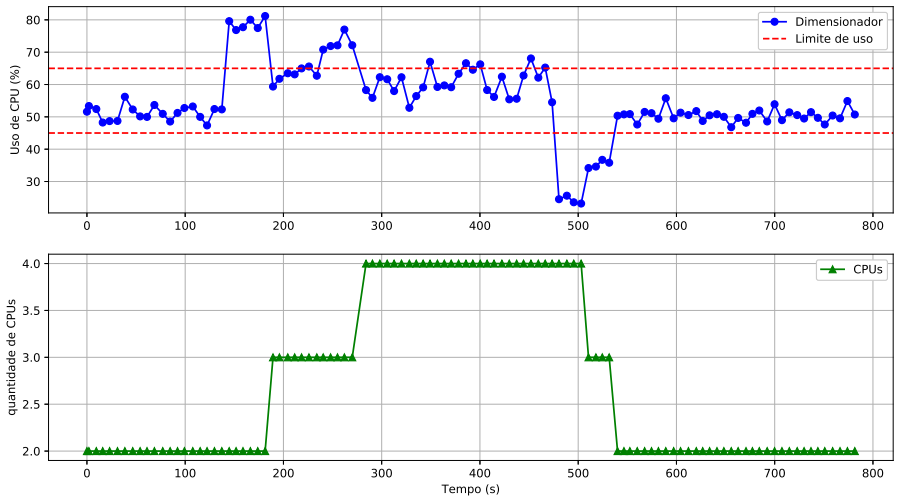


Figura 22 – Análise do uso de CPU no serviço de processamento de imagem do robô.

5 Conclusão e Trabalhos Futuros

Este trabalho apresentou um protótipo de dimensionador para alocação dinâmica de recursos em máquinas virtuais instanciadas em nuvens computacionais. O dimensionador aloca recursos visando evitar que a VM tenha gargalos de utilização e desaloca recursos para evitar subutilização dos mesmos.

Para avaliar o desempenho do dimensionador, foi proposto três estudos de caso, sendo o primeiro estudo caso um gerador de carga sintética, o segundo um servidor *web* sob alta demanda de usuários, e o terceiro o controle remoto de um robô em um espaço inteligente.

No primeiro estudo de caso foi possível observar que o dimensionador é capaz de operar com diferentes recursos simultaneamente (no caso, CPU e memória) e não apenas um recurso por vez. No segundo estudo de caso observou-se que o dimensionador é capaz de reagir à uma variação de uso de recursos por parte da demanda de carga de um servidor *web*. O terceiro estudo de caso, que é o robô em um espaço inteligente. mostrou que, aumentando-se o FPS das câmeras, aumenta a demanda por recursos de processamento da máquina virtual que contém o serviço de processamento de imagens. Logo, foi necessário que o dimensionador atuasse com eficiência para atender a demanda. Por outro lado, com a redução do uso de processador por parte da VM, o dimensionador foi capaz de remover recursos subutilizados.

O protótipo do dimensionador foi implementado utilizando tecnologias modernas e é passível de melhorias. O dimensionador é um sistema distribuído, logo melhorias em algum dos seus algoritmos podem ser feitas, tendo impacto mínimo nos outros, o que permite implementações de variações do dimensionador.

Para trabalhos futuros, considera-se melhorias e novas funcionalidades para o dimensionador proposto. Um deles é criar um analisador inteligente para dimensionar recursos, tendo como base o histórico de uso da máquina virtual. Considerar também a junção do dimensionador com dimensionadores horizontais, na tentativa de unificar os pontos positivos de cada um. Por fim, aumentar o leque recursos analisados do dimensionador (hoje CPU e memória), adicionando recursos de rede, por exemplo.

Referências

AMAZON. *AWS Auto Scaling*. 2018. Http://aws.amazon.com/autoscaling/. Disponível em: <http://aws.amazon.com/autoscaling/>. Citado 2 vezes nas páginas 13 e 17.

AMAZON. *Amazon Web Service (AWS) - Cloud Computing Services*. 2019. Https://aws.amazon.com/. Disponível em: <https://aws.amazon.com/>. Citado 2 vezes nas páginas 15 e 47.

ARABNEJAD, H. et al. An auto-scaling cloud controller using fuzzy q-learning-implementation in openstack. In: SPRINGER. *European Conference on Service-Oriented and Cloud Computing*. [S.l.], 2016. p. 152–167. Citado 2 vezes nas páginas 12 e 13.

CHEN, T.; BAHSOON, R. Self-adaptive trade-off decision making for autoscaling cloud-based services. *IEEE Transactions on Services Computing*, IEEE, v. 10, n. 4, p. 618–632, 2017. Citado 2 vezes nas páginas 18 e 19.

DAWOUD, W.; TAKOUNA, I.; MEINEL, C. Elastic virtual machine for fine-grained cloud resource provisioning. In: *Global Trends in Computing and Communication Systems*. [S.l.]: Springer, 2012. p. 11–25. Citado 4 vezes nas páginas 14, 18, 19 e 20.

DIKAIAKOS, M. D. et al. Cloud computing: Distributed internet computing for it and scientific research. *IEEE Internet computing*, IEEE, v. 13, n. 5, 2009. Citado na página 12.

FERNANDEZ, H.; PIERRE, G.; KIELMANN, T. Autoscaling web applications in heterogeneous cloud infrastructures. In: IEEE. *Cloud Engineering (IC2E), 2014 IEEE International Conference on*. [S.l.], 2014. p. 195–204. Citado 2 vezes nas páginas 17 e 19.

GONG, Z.; GU, X.; WILKES, J. Press: Predictive elastic resource scaling for cloud systems. *CNSM*, v. 10, p. 9–16, 2010. Citado 2 vezes nas páginas 18 e 19.

GOOGLE. *Google Cloud*. 2019. Https://cloud.google.com/. Disponível em: <https://cloud.google.com/>. Citado 4 vezes nas páginas 13, 15, 17 e 47.

GROZEV, N.; BUYYA, R. Dynamic selection of virtual machines for application servers in cloud environments. In: *Research Advances in Cloud Computing*. [S.l.]: Springer, 2017. p. 187–210. Citado 2 vezes nas páginas 17 e 19.

JADEJA, Y.; MODI, K. Cloud computing-concepts, architecture and challenges. In: IEEE. *Computing, Electronics and Electrical Technologies (ICCEET), 2012 International Conference on*. [S.l.], 2012. p. 877–880. Citado na página 12.

LIBVIRT. *libvirt: the virtualization API*. 2019. Https://libvirt.org/. Disponível em: <https://libvirt.org/>. Citado 2 vezes nas páginas 15 e 25.

MELL, P.; GRANCE, T. et al. The nist definition of cloud computing. Computer Security Division, Information Technology Laboratory, National Institute of Standards and Technology Gaithersburg, 2011. Citado na página 19.

OPENSTACK. *OpenStack Docs: Welcome to shade's documentation!* 2017. Https://docs.openstack.org/shade/latest/. Disponível em: <https://docs.openstack.org/shade/latest/>. Citado na página 25.

OPENSTACK. *Build the future of Open Infrastructure*. 2019. Https://www.openstack.org/. Disponível em: <https://www.openstack.org/>. Citado 2 vezes nas páginas 18 e 22.

PIVOTAL. *Messages that just work - RabbitMQ*. 2019. Https://www.rabbitmq.com/. Disponível em: <https://www.rabbitmq.com/>. Citado 2 vezes nas páginas 15 e 26.

PYTHON. *Welcome to Python.org*. 2019. Https://www.python.org/. Disponível em: <https://www.python.org/>. Citado na página 22.

QU, C.; CALHEIROS, R. N.; BUYYA, R. Auto-scaling web applications in clouds: A taxonomy and survey. *ACM Comput. Surv.*, ACM, New York, NY, USA, v. 51, n. 4, p. 73:1–73:33, jul. 2018. ISSN 0360-0300. Disponível em: <http://doi.acm.org/10.1145/3148149>. Citado 5 vezes nas páginas 11, 12, 14, 17 e 18.

SANTOS, P. B. et al. Monitoramento de recursos para aplicações de rob ótica em espaços inteligentes baseados em uma nuvem openstack. In: SBC. *Anais do XVI Workshop em Clouds e Aplicações (WCGA-SBRC 2018)*. [S.l.], 2018. v. 16. Citado 2 vezes nas páginas 49 e 50.

SHARMA, U.; SHENOY, P.; TOWSLEY, D. F. Provisioning multi-tier cloud applications using statistical bounds on sojourn time. In: ACM. *Proceedings of the 9th international conference on Autonomic computing*. [S.l.], 2012. p. 43–52. Citado 2 vezes nas páginas 17 e 19.

THAMES, L.; SCHAEFER, D. Software-defined cloud manufacturing for industry 4.0. *Procedia CIRP*, Elsevier, v. 52, p. 12–17, 2016. Citado 2 vezes nas páginas 11 e 12.

UOL. *UOL CLoud OpenStack: Computação em Nuvem - UOL HOST*. 2019. Https://uolhost.uol.com.br/openstack. Disponível em: <https://uolhost.uol.com.br/openstack>. Citado na página 11.

WANG, C.; GUPTA, A.; URGAONKAR, B. Fine-grained resource scaling in a public cloud: A tenant's perspective. In: IEEE. *Cloud Computing (CLOUD), 2016 IEEE 9th International Conference on*. [S.l.], 2016. p. 124–131. Citado 2 vezes nas páginas 18 e 19.

XU, J. et al. On the use of fuzzy modeling in virtualized data center management. In: IEEE. *Autonomic Computing, 2007. ICAC'07. Fourth International Conference on*. [S.l.], 2007. p. 25–25. Citado 2 vezes nas páginas 12 e 13.

YAZDANOV, L.; FETZER, C. Vscaler: Autonomic virtual machine scaling. In: IEEE. *2013 IEEE Sixth International Conference on Cloud Computing*. [S.l.], 2013. p. 212–219. Citado na página 11.

Apêndices

APÊNDICE A – Mensagens do dimensionador

Todas as mensagens utilizadas pelo dimensionador para comunicação entre os componentes são construídas em JSON[1]. JSON é um acrônimo para *JavaScript Object Notation* e é uma maneira de se guardar informação de maneira organizada e de fácil acesso, além de ser uma coleção de informações humanamente legível. A seguir são mostrados alguns exemplos de mensagens em formato JSON utilizados pelo dimensionador:

Código A.1 – Mensagem para ligar o dimensionamento de uma VM chamada 'arucos'.

```
{
    u'min_memory_usage':0,
    u'max_vcpu_usage':65,
    u'instance':[
        u'arucos'
    ],
    u'max_memory_usage':0,
    u'time':5,
    u'action':u'start',
    u'min_vcpu_usage':45,
    u'project_id':u'dfc9d23cd35f462287a151558052f127'
}
```

Código A.2 – Mensagem para desligar o dimensionamento de uma VM chamada 'arucos'.

```
{
    u'min_memory_usage':0,
    u'max_vcpu_usage':65,
    u'instance':[
        u'arucos'
    ],
    u'max_memory_usage':0,
    u'time':5,
    u'action':u'stop',
    u'min_vcpu_usage':45,
    u'project_id':u'dfc9d23cd35f462287a151558052f127'
}
```

[1] https://www.w3schools.com/whatis/whatis_json.asp

Código A.3 – Mensagem do Gerente para o Agente requisitando sobre uma VM chamada 'arucos'

```
{
    u'command':u'find_instance',
    u'parameters':[
        u'arucos',
        u'dfc9d23cd35f462287a151558052f127',
        5
    ]
}
```

Código A.4 – Mensagem do Agente para o Gerente contendo o resultado da requisição de informações sobre uma VM chamada 'arucos'

```
{
    u'command':u'find_instance',
    'result':{
        'status':{
            'min_vcpus':1,
            'current_vcpus':2,
            'max_memory':4194304,
            'max_vcpus':4,
            'min_memory':1048576,
            'current_memory':3145728
        },
        'instance_name':u'arucos',
        'host_name':u'compute1',
        'usage':{
            'cpu_usage':47.64239210851,
            'mem_usage':60.64453125
        },
        'flavor':Munch(        {
            'name':u'p1.scale',
            'ephemeral':0,
            'ram':4096,
            'is_disabled':False,
            'properties':Munch( {
                u'OS-FLV-DISABLED: disabled':False,
                u'OS-FLV-EXT-DATA: ephemeral':0,
                u'os-flavor-access: is_public':True
```

```
            } ),
            u'OS-FLV-DISABLED: disabled ': False,
            'vcpus ':4,
            'extra_specs ': Munch( {
                u'hw: cpu_policy ':u'dedicated '
            } ),
            'location ': Munch( {
                'project ': Munch( {
                    'domain_id ': 'default ',
                    'id ':u'dfc9d23cd35f462287a151558052f127 ',
                    'name': 'admin ',
                    'domain_name': 'default '
                }             ),
                'zone ': None,
                'region_name ': 'RegionOne ',
                'cloud ': 'amprod '
            }           ),
            u'os-flavor-access: is_public ': True,
            'rxtx_factor ':1.0,
            'is_public ': True,
            u'OS-FLV-EXT-DATA: ephemeral ':0,
            'disk ':30,
            'id ':u'88035bf7-5d0a-4284-952f-84e8a327396a ',
            'swap ':0
        }       ),
        'id ':u'3d990a3b-9fe9-4185-b250-5b66fa71d279 ',
        'libvirt_name ':u'instance-000000cb '
    },
    u'parameters ':[
        u'arucos ',
        u'dfc9d23cd35f462287a151558052f127 ',
        5
    ]
}
```

Código A.5 – Mensagem enviada do Gerente para o Agente requisitando troca de CPU na VM 'arucos'

```
{
    u'command ':u'change_cpu ',
```

```
u'parameters':[
    {
        u'status':{
            u'min_vcpus':1,
            u'current_vcpus':3,
            u'max_memory':4194304,
            u'max_vcpus':4,
            u'min_memory':1048576,
            u'current_memory':3145728
        },
        u'instance_name':u'arucos',
        u'host_name':u'compute1',
        u'usage':{
            u'cpu_usage':35.22567063485,
            u'mem_usage':61.230468749999986
        },
        u'flavor':{
            u'name':u'p1.scale',
            u'ephemeral':0,
            u'ram':4096,
            u'is_disabled':False,
            u'properties':{
                u'OS-FLV-DISABLED: disabled':False,
                u'OS-FLV-EXT-DATA: ephemeral':0,
                u'os-flavor-access: is_public':True
            },
            u'OS-FLV-DISABLED: disabled':False,
            u'vcpus':4,
            u'extra_specs':{
                u'hw: cpu_policy':u'dedicated'
            },
            u'location':{
                u'project':{
                    u'id':u'dfc9d23cd35f462287a151558052f127',
                    u'domain_name':u'default',
                    u'name':u'admin',
                    u'domain_id':u'default'
                },
                u'zone':None,
```

```
                u'region_name':u'RegionOne',
                u'cloud':u'amprod'
            },
            u'os-flavor-access: is_public':True,
            u'rxtx_factor':1.0,
            u'is_public':True,
            u'OS-FLV-EXT-DATA: ephemeral':0,
            u'disk':30,
            u'id':u'88035bf7-5d0a-4284-952f-84e8a327396a',
            u'swap':0
        },
        u'id':u'3d990a3b-9fe9-4185-b250-5b66fa71d279',
        u'libvirt_name':u'instance-000000cb'
    },
    2
]
}
```

Código A.6 – Mensagem enviada do Agente para o Gerente com o resultado da troca de CPU na VM 'arucos'

```
{
    u'command':u'change_cpu',
    'result':"VCPU of VM 'arucos' (instance-000000cb) changed
    in host 'compute1'.",
    u'parameters':[
        {
            u'status':{
                u'min_vcpus':1,
                u'current_vcpus':3,
                u'max_memory':4194304,
                u'max_vcpus':4,
                u'min_memory':1048576,
                u'current_memory':3145728
            },
            u'instance_name':u'arucos',
            u'host_name':u'compute1',
            u'usage':{
                u'cpu_usage':35.22567063485,
                u'mem_usage':61.230468749999986
```

```
        },
        u'flavor':{
            u'name':u'p1.scale',
            u'ephemeral':0,
            u'ram':4096,
            u'is_disabled':False,
            u'properties':{
                u'OS-FLV-DISABLED: disabled':False,
                u'OS-FLV-EXT-DATA: ephemeral':0,
                u'os-flavor-access: is_public':True
            },
            u'OS-FLV-DISABLED: disabled':False,
            u'vcpus':4,
            u'extra_specs':{
                u'hw: cpu_policy':u'dedicated'
            },
            u'location':{
                u'project':{
                    u'id':u'dfc9d23cd35f462287a151558052f127',
                    u'domain_name':u'default',
                    u'name':u'admin',
                    u'domain_id':u'default'
                },
                u'zone':None,
                u'region_name':u'RegionOne',
                u'cloud':u'amprod'
            },
            u'os-flavor-access: is_public':True,
            u'rxtx_factor':1.0,
            u'is_public':True,
            u'OS-FLV-EXT-DATA: ephemeral':0,
            u'disk':30,
            u'id':u'88035bf7-5d0a-4284-952f-84e8a327396a',
            u'swap':0
        },
        u'id':u'3d990a3b-9fe9-4185-b250-5b66fa71d279',
        u'libvirt_name':u'instance-000000cb'
    },
    2
```

```
    ]
}
```

Código A.7 – Mensagem enviada do Gerente para o Agente requisitando troca de memória na VM 'arucos'

```
{
    u'command':u'change_memory',
    u'parameters':[
        {
            u'status':{
                u'min_vcpus':1,
                u'current_vcpus':4,
                u'max_memory':4194304,
                u'max_vcpus':4,
                u'min_memory':1048576,
                u'current_memory':4194304
            },
            u'instance_name':u'arucos',
            u'host_name':u'compute1',
            u'usage':{
                u'cpu_usage':19.957089361029997,
                u'mem_usage':21.313476562499996
            },
            u'flavor':{
                u'name':u'p1.scale',
                u'ephemeral':0,
                u'ram':4096,
                u'is_disabled':False,
                u'properties':{
                    u'OS-FLV-DISABLED: disabled':False,
                    u'OS-FLV-EXT-DATA: ephemeral':0,
                    u'os-flavor-access: is_public':True
                },
                u'OS-FLV-DISABLED: disabled':False,
                u'vcpus':4,
                u'extra_specs':{
                    u'hw: cpu_policy':u'dedicated'
                },
                u'location':{
```

```
                u'project':{
                    u'id':u'dfc9d23cd35f462287a151558052f127',
                    u'domain_name':u'default',
                    u'name':u'admin',
                    u'domain_id':u'default'
                },
                u'zone':None,
                u'region_name':u'RegionOne',
                u'cloud':u'amprod'
            },
            u'os-flavor-access: is_public':True,
            u'rxtx_factor':1.0,
            u'is_public':True,
            u'OS-FLV-EXT-DATA: ephemeral':0,
            u'disk':30,
            u'id':u'88035bf7-5d0a-4284-952f-84e8a327396a',
            u'swap':0
        },
        u'id':u'3d990a3b-9fe9-4185-b250-5b66fa71d279',
        u'libvirt_name':u'instance-000000cb'
    },
    3670016
]
}
```

Código A.8 – Mensagem enviada do Agente para o Gerente com o resultado da troca de memória na VM 'arucos'

```
{
    u'command':u'change_memory',
    'result':"Memory of VM 'arucos' (instance-000000cb) changed
    in host 'compute1'.",
    u'parameters':[
        {
            u'status':{
                u'min_vcpus':1,
                u'current_vcpus':4,
                u'max_memory':4194304,
                u'max_vcpus':4,
                u'min_memory':1048576,
```

```
        u'current_memory':4194304
    },
    u'instance_name':u'arucos',
    u'host_name':u'compute1',
    u'usage':{
        u'cpu_usage':19.957089361029997,
        u'mem_usage':21.313476562499996
    },
    u'flavor':{
        u'name':u'p1.scale',
        u'ephemeral':0,
        u'ram':4096,
        u'is_disabled':False,
        u'properties':{
            u'OS-FLV-DISABLED: disabled':False,
            u'OS-FLV-EXT-DATA: ephemeral':0,
            u'os-flavor-access: is_public':True
        },
        u'OS-FLV-DISABLED: disabled':False,
        u'vcpus':4,
        u'extra_specs':{
            u'hw: cpu_policy':u'dedicated'
        },
        u'location':{
            u'project':{
                u'id':u'dfc9d23cd35f462287a151558052f127',
                u'domain_name':u'default',
                u'name':u'admin',
                u'domain_id':u'default'
            },
            u'zone':None,
            u'region_name':u'RegionOne',
            u'cloud':u'amprod'
        },
        u'os-flavor-access: is_public':True,
        u'rxtx_factor':1.0,
        u'is_public':True,
        u'OS-FLV-EXT-DATA: ephemeral':0,
        u'disk':30,
```

```
            u'id':u'88035bf7-5d0a-4284-952f-84e8a327396a',
            u'swap':0
        },
        u'id':u'3d990a3b-9fe9-4185-b250-5b66fa71d279',
        u'libvirt_name':u'instance-000000cb'
    },
    3670016
]
}
```