



Universidade de Brasília - UnB
Faculdade UnB Gama - FGA
Engenharia de Software

Um Estudo da Interferência Entre Máquinas Virtuais em seus Desempenhos

Autor: Maxwell Almeida Santos
Orientador: Dr. Paulo Meirelles
Coorientador: Bel. Rafael Manzo

Brasília, DF
2016



Maxwell Almeida Santos

Um Estudo da Interferência Entre Máquinas Virtuais em seus Desempenhos

Monografia submetida ao curso de graduação em (Engenharia de Software) da Universidade de Brasília, como requisito parcial para obtenção do Título de Bacharel em (Engenharia de Software).

Universidade de Brasília - UnB

Faculdade UnB Gama - FGA

Orientador: Dr. Paulo Meirelles

Coorientador: Bel. Rafael Manzo

Brasília, DF

2016

Maxwell Almeida Santos

Um Estudo da Interferência Entre Máquinas Virtuais em seus Desempenhos/
Maxwell Almeida Santos. – Brasília, DF, 2016-
82 p. : il. (algumas color.) ; 30 cm.

Orientador: Dr. Paulo Meirelles

Trabalho de Conclusão de Curso – Universidade de Brasília - UnB
Faculdade UnB Gama - FGA , 2016.

1. Computação em Nuvem, Virtualização. 2. Desempenho de Sistemas, KVM.
I. Dr. Paulo Meirelles. II. Universidade de Brasília. III. Faculdade UnB Gama.
IV. Um Estudo da Interferência Entre Máquinas Virtuais em seus Desempenhos

CDU 02:141:005.6

Maxwell Almeida Santos

Um Estudo da Interferência Entre Máquinas Virtuais em seus Desempenhos

Monografia submetida ao curso de graduação em (Engenharia de Software) da Universidade de Brasília, como requisito parcial para obtenção do Título de Bacharel em (Engenharia de Software).

Trabalho aprovado. Brasília, DF, 01 de junho de 2013:

Dr. Paulo Meirelles
Orientador

Bel. Rafael Manzo
Orientador

Dr. Fernando William Cruz
Convidado 1

Dr. Tiago Alves
Convidado 2

Brasília, DF
2016

Resumo

O uso de ambientes virtualizados vem tendo um aumento crescente nos últimos anos e isso se deve em parte ao crescente uso da computação em nuvem. Entretanto, aplicações disponibilizadas em ambientes virtualizados podem ter uma perda considerável de desempenho, se comparadas as mesmas aplicações disponibilizadas em máquinas físicas convencionais. Um dos fatores que colaboram para perda de desempenho, é a interferência sofrida pelas máquinas virtuais quando executadas em um mesmo servidor. Os resultados deste trabalho comprovam que além disso, essa interferência é variável dado o tipo de aplicação que está sendo executada nas máquinas virtuais. Com isso, a partir de trabalhos já realizados, esse trabalho propõe a aplicação de um estudo de interferência de desempenho em máquinas virtuais, utilizando o *kvm* como *hypervisor*. Para coleta de dados, foram desenvolvidos procedimentos, bem como, foram definidas ferramentas tipicamente utilizadas para medição de desempenho em sistemas computacionais.

Palavras-chaves: Computação em Nuvem; Virtualização; Desempenho de Sistemas; KVM.

Abstract

The use of virtualized environments has had a growing increase in recent years and this is due in part to the increasing use of cloud computing. However, applications deployed in virtualized environments may have a considerable loss of performance, compared the same applications available in conventional physical machines. One of the factors that contribute to performance loss, is the interference suffered by virtual machines when running on the same server. The results of this work show that in addition, this interference is variable given the type of application that is running in virtual machines. So, from related works, this work proposes the application of a performance interference study on virtual machines using the kvm as hypervisor. For data collection, procedures have been developed, as well as, was defined using tools typically used for measuring performance in computer systems.

Key-words: Cloud Computing; Virtualization; Performance Systems; KVM.

Lista de ilustrações

Figura 1 – Responsabilidade da virtualização	24
Figura 2 – Representação da implementação de um ambiente virtual	25
Figura 3 – Representação da virtualização total	26
Figura 4 – Representação da virtualização total	27
Figura 5 – Representação da paravirtualização	28
Figura 6 – Arquitetura do KVM	29
Figura 7 – Arquitetura do XEN	30
Figura 8 – Visão simplificada de uma instalação mínima do cloudstack	31
Figura 9 – Visão geral da infraestrutura do <i>Cloudstack</i>	32
Figura 10 – Visão geral da infraestrutura do OpenNebula	33
Figura 11 – Funcionamento do <i>System Datastore</i> na configuração <i>SSH</i>	34
Figura 12 – Funcionamento do <i>System Datastore</i> na configuração <i>shared</i>	34
Figura 13 – Arquitetura de implantação por zonas	35
Figura 14 – Taxa de escrita em disco para experimentos voltados para E/S.	39
Figura 15 – Fatores que influenciam no desempenho de plataformas virtualizadas	39
Figura 16 – Pontuação combinada de aplicações selecionadas para o primeiro experimento	47
Figura 17 – Pontuação normalizada contra um conjunto de aplicações	48
Figura 18 – Variação de desempenho para um conjunto de aplicações	50
Figura 19 – Desempenho alcançado das aplicações para requisições de escrita e leitura em disco	51
Figura 20 – Tempo que as aplicações levam para executar uma operação de leitura e escrita em disco	51
Figura 21 – Porcentagem de utilização de cpu para cada aplicação	52
Figura 22 – Pontuação normalizada das métricas a nível de sistema de <i>Bzip2</i> contra diferentes tipos de aplicações.	52
Figura 23 – Pontuação normalizada das métricas a nível de sistema de <i>dd</i> contra diferentes aplicações.	54
Figura 24 – Pontuação normalizada das métricas a nível de sistema de <i>Grep</i> contra diferentes aplicações.	54
Figura 25 – Visão geral da Infraestrutura	65
Figura 26 – Arquitetura do Portal do Software Público	68
Figura 27 – Implantação do <i>OpenNebula</i>	70
Figura 28 – Visão geral da Infraestrutura após implantação do <i>OpenNebula</i>	71

Lista de tabelas

Tabela 1 – Comparativo entre <i>Cloudstack</i> e <i>OpenNebula</i>	35
Tabela 2 – Contadores de desempenho de disco para <i>Windows</i> e <i>Linux</i> (POPIO-LEK; MENDIZABAL, 2012)	38
Tabela 3 – Aplicações utilizadas para geração de cargas e trabalho	44
Tabela 4 – Desvio Padrão para diferentes quantidades de valores para <i>Povray</i> e <i>Make</i>	45

Código

2.1	Código exemplo de uma receita Chef	36
-----	--	----

Lista de abreviaturas e siglas

API	Application Programming Interface - Interface de Programação de Aplicações
CPD	Centro de Processamento de Dados
CPU	Central Process Unit - Unidade central de processamento
DEVOPS	Development and Operations - Desenvolvedor e Operações
FGA	Faculdade UnB Gama
HDD	<i>Hard Disk Drive</i> - Disco rígido
IP	Internet Protocol - Protocolo de Internet
LAPPIS	Laboratório Avançado de Pesquisa e Inovação de Software
LVM	<i>Logical Volume Manager</i> - Gerenciador de Volume Lógico
PCA	Principal Component Analysis - Análise de Componente Principal
SGD	Sistema de Gestão do Desempenho
SPB	Portal do Software Público
SRA	Sistemas de Registros de Atendimento
SSVM	Secondary Storage Virtual Machine - Máquina virtual para armazenamento secundário
VLAN	Virtual Local Area Network - Rede local Virtual
VM	Virtual Machine - Máquina virtual

Sumário

1	INTRODUÇÃO	19
1.1	Contextualização	19
1.2	Justificativa	20
1.3	Objetivos	20
1.4	Estrutura do trabalho	21
2	VIRTUALIZAÇÃO	23
2.1	Conceituação	23
2.2	Máquinas virtuais	24
2.3	Monitor de Máquinas virtuais	25
2.4	Tipos de virtualização	26
2.5	Ferramentas de Virtualização	28
2.6	Suporte Tecnológico	30
2.6.1	Cloudstack	30
2.6.2	OpenNebula	31
2.6.3	Comparativo entre as ferramentas de Plataforma em nuvem	34
2.6.4	Automatização da Infraestrutura	36
3	METODOLOGIA	37
3.1	Trabalhos relacionados	37
3.2	Ambiente de Testes	40
3.3	Coleta de dados	42
3.3.1	Cálculo da Interferência	42
3.3.2	Métricas de desempenho a nível de sistema	43
3.3.3	Procedimentos experimentais	44
4	ANÁLISE DE RESULTADOS	47
4.1	Interferência de Desempenho	47
4.2	Desempenho contra diferentes tipos de Aplicações	48
4.3	Interferência nas Métricas de Desempenho a nível de Sistema	50
5	CONCLUSÃO	57
	Referências	59

	APÊNDICES	61
	APÊNDICE A – ESTUDO DE CASO	63
A.1	Lappis	63
A.2	Infraestrutura	64
A.3	Migração de Máquinas Virtuais	66
A.4	Implementação da Plataforma em nuvem	66
A.5	Consolidação da Infraestrutura	70
	APÊNDICE B – CÓDIGO FONTE DA RECEITA DO <i>OPENNE-BULA FRONTEND</i>	73
	APÊNDICE C – CÓDIGO FONTE DA RECEITA DO <i>OPENNE-BULA NODE</i>	75
	APÊNDICE D – CÓDIGO FONTE DA RECEITA DO <i>REDMINE</i>	77
	APÊNDICE E – CÓDIGO FONTE DA RECEITA DO SERVIDOR <i>PROXY USANDO SQUID</i>	81

1 Introdução

1.1 Contextualização

Ultimamente, devido a tendências como computação em nuvem, TI verde e a consolidação de servidores, a virtualização vem ganhando cada vez mais importância. Antigamente usada para uso mais eficiente de recursos físicos de *mainframes*, hoje em dia a virtualização é novamente utilizada para executar múltiplas máquinas virtuais em uma infraestrutura compartilhada, aumentando dessa forma a utilização de recursos, promovendo flexibilidade e centralizando a administração (HUBER; KONEV; HAUCK, 2011). Segundo Popiolek e Mendizabal (2012), essa popularidade se deve também ao amplo uso de infraestruturas distribuídas por parte de sistemas computacionais modernos, o que colabora para o desenvolvimento de aplicações colaborativas e promove o compartilhamento de recursos remotos.

Neste contexto, a computação em nuvem, alinhada à virtualização, permite um conjunto de servidores físicos disponibilizar dezenas ou centenas de máquinas virtuais. Desse modo, proporciona-se aumento na escalabilidade, maximizando o uso de recursos (POPIOLEK; MENDIZABAL, 2012). Entretanto, antes de migrar aplicações de ambientes não virtuais para ambientes virtuais, é necessário entender como será o desempenho dessas aplicações nesse novo ambiente (SOUZA, 2006). A adoção de servidores virtualizados vem com o aumento de custo na complexidade e dinâmica do sistema. O aumento da dinâmica é causada pela falta de controle direto sobre o hardware, pelas interações complexas entre as aplicações e cargas de trabalho que compartilham os mesmos recursos físicos, introduzindo novos desafios em sistemas de gestão (HUBER; KONEV; HAUCK, 2011).

De acordo com Koh et al. (2007), observa-se que as tecnologias atuais que possibilitam a criação de máquinas virtuais não fornecem um isolamento efetivo. Enquanto o *hypervisor* (software responsável pela disponibilização e gerenciamento de máquinas virtuais), reparte os recursos e os aloca para as VMs, o comportamento de cada VM ainda pode afetar o desempenho das outras de maneira negativa, devido ao uso de recursos compartilhados no sistema.

Além disso, o custo de desempenho de virtualização pode variar de forma significativa, dependendo da configuração do ambiente virtual. Uma escolha de configuração que afeta criticamente o desempenho do sistema é a alocação de CPUs à várias máquinas virtuais (SOUZA, 2006).

A partir do que foi abordado até agora, entende-se que é fundamental compreender

os fatores que impactam no desempenho de ambientes virtualizados, bem como quantificar e avaliar o desempenho de aplicações em tais ambientes para que possa implantá-las e configurá-las adequadamente. Tendo isso em vista, torna-se importante o conhecimento de ferramentas e técnicas ou metodologias que auxiliem nas atividades relacionadas a análise de desempenho em ambientes virtualizados.

1.2 Justificativa

A virtualização introduz um outro nível de complexidade para a modelagem de servidores e análise de desempenho. Como CPDs rapidamente adotaram a virtualização como meio principal para consolidar múltiplas aplicações em um servidor, torna-se essencial que a desempenho de máquinas virtuais seja bem compreendidas ([TICKOO et al.,](#)). Para [Huber, Konev e Hauck \(2011\)](#) os provedores de plataformas virtualizadas se deparam com as seguintes questões:

- Qual desempenho teria um novo serviço disponibilizado em um infraestrutura virtualizado e quanto de recurso deve ser alocado para tal serviço?
- Como a configuração do sistema deve ser adaptada para evitar problemas de desempenho decorrentes de mudanças de cargas de trabalho?

Tais questões evidenciam a necessidade de analisar os fatores que colaboram para a perda de desempenho em um ambiente virtualizado, bem como determinar quais configurações seriam adequadas para que tais fatores sejam minimizados sem que haja também o desperdício de recursos de hardware. Desse modo, em um cenário hipotético, sem ter um estudo ou uma metodologia prévia referente à análise de desempenho em ambientes virtualizados, e dado o uso dos mesmos no contexto de computação em nuvem, um profissional na área de computação em nuvem ao se deparar com problemas relacionados a desempenho, pode tomar decisões equivocadas ocasionando perda de tempo na disponibilização de serviços em nuvem.

1.3 Objetivos

A partir do contexto e da justificativa apresentada, o objetivo deste trabalho é aplicar um estudo de interferência de desempenho entre máquinas virtuais utilizando como estudo de caso a infraestrutura do LAPPIS¹. Como colaboração ao LAPPIS e garantia de um provimento de máquinas virtuais de fácil gerenciamento para realização do estudo,

pretende-se também estabelecer uma plataforma em nuvem nessa infraestrutura. Para isso, os seguintes objetivos devem ser perseguidos:

- Implantar uma plataforma em nuvem nos servidores do LAPPIS.
- Automatizar parcialmente a infraestrutura virtual do LAPPIS (plataforma em nuvem, serviços oferecidos).
- Estabelecer um ambiente para estudo de interferência de desempenho entre ambientes virtuais.
- Aplicar a análise de desempenho em um dos servidores físicos do LAPPIS.
- Analisar e documentar os resultados obtidos a partir da análise de desempenho.

1.4 Estrutura do trabalho

Este trabalho está organizado em quatro capítulos subsequentes. No Capítulo 2, são apresentados conceitos referentes à ambientes virtualizados. Esses conceitos englobam a definição de elementos que constituem os ambientes virtualizados, tais como: máquina virtuais, monitor de máquinas virtuais, tipos de virtualização e ferramentas de virtualização. Por fim, ainda neste capítulo, são apresentadas as ferramentas utilizadas para provimento de uma infraestrutura de computação em nuvem. No capítulo 3, são apresentados os trabalhos relacionados bem como os procedimentos adotados para aplicação do estudo de interferência em ambientes virtuais. No Capítulo 4, são apresentados os resultados obtidos a partir da execução dos procedimentos definidos no Capítulo 3, sendo também feita uma análise desses resultados. Por fim, no Capítulo 5, são feitas ponderações referentes à execução do trabalho bem como dos resultados alcançados. Neste capítulo, também são apresentadas algumas sugestões de trabalhos futuros.

¹ O Laboratório Avançado de Produção, Pesquisa & Inovação em Software (LAPPIS), localizado na Faculdade UnB Gama (FGA) foi criado em 2012 e foi concebido para atuar em áreas tecnológicas desde sistemas de informação até os sistemas embarcados, em especial, objetivando as oportunidades de pesquisas teóricas e aplicadas. (LAPPIS, 2014)

2 Virtualização

No contexto da computação, frequentemente uma alteração tecnológica torna alguma idéia obsoleta e ela desaparece rapidamente. Entretanto, outra mudança tecnológica poderia reavivá-la (TANENBAUM, 2007). Este é o caso da virtualização, dado que sua utilização teve oscilações ao longo do tempo. A principal motivação para virtualização nos começo dos anos 70 era aumentar o nível de compartilhamento e utilização dos caros recursos computacionais tais como *mainframes* (MENASCÉ, 2005). Nos anos 80 com a queda dos custos de hardware as grandes corporações trocaram os grandes e despidiosos *mainframes* por coleções de computadores pessoais, tendo assim a virtualização caído em desuso.

Seu ressurgimento só viria acontecer, nos anos 90 dentro de um contexto ao qual tinha-se o crescimento de novos paradigmas computacionais, tais como cliente-servidor e sistemas *peer-to-peer*, cujas estruturas eram constituídas basicamente de máquinas clientes conectadas a vários servidores. Esses novos ambientes trouxeram com eles diversos desafios e problemas incluindo confiabilidade, segurança, aumento no custo de administração e complexidade, espaço físico e consumo de energia (MENASCÉ, 2005). Desse modo, o ressurgimento da virtualização vem promovendo a mitigação de tais problemas recorrentes nesses novos paradigmas computacionais (alteração tecnológica).

2.1 Conceituação

Define-se que virtualização é a técnica que permite particionar um único sistema computacional em vários outros denominados de máquinas virtuais. Cada máquina virtual oferece um ambiente completo muito similar a uma máquina física. Com isso, cada máquina virtual pode ter seu próprio sistema operacional, aplicativos e serviços de rede (Internet) (CARISSIMI, 2016). A virtualização pode ser feita das seguintes formas:

- **Virtualização de servidores:** a mais comum e fácil de ser justificada. Diferente da época dos *mainframes*, a virtualização agora é feita em servidores *x86*.
- **Virtualização de *desktops*:** trata da configuração dos *desktops* dos usuários finais em uma infraestrutura centralizada virtual. Isso significa que as aplicações de *desktop* também passam a executar em um *datacenter*, sob a forma de máquinas virtuais. Esse é o conceito de *Virtual Desktop Infrastructure (VDI)*, que permite a montagem dinâmica de *desktops*, oferecendo maior confiabilidade e otimização do uso de espaço em disco com a consollidação do armazenamento e flexibilidade na escolha do sistema peracional (VERAS; CARISSIMI, 2015).

- **Virtualização do armazenamento (*storage*):** a ideia é introduzir um componente que permite às diversas unidade heterogêneas de armazenamento (discos físicos) serem vistas como um conjunto homogêneo de recursos (VERAS; CARISSIMI, 2015).
- **Virtualização das aplicações:** trata do conceito de execução do programa por completo, em um repositório central, permitindo a configuração centralizada do aplicativo, o que melhora seu gerenciamento, por permitir que seja feita em um único lugar (VERAS; CARISSIMI, 2015).
- **Virtualização de redes:** Arquitetura que proporciona um ambiente de rede separado para cada grupo ou organização. Esses ambientes lógicos são criados sobre uma única infraestrutura compartilhada de rede (VERAS; CARISSIMI, 2015).

A Figura 1 apresenta um modelo que pode ser utilizado para conceituar a virtualização: uma camada de abstração entre o hardware e o software, que protege o acesso direto do software aos recursos físicos do hardware. A forma pela qual essa camada de abstração é implementada dá origem às máquinas virtuais de processo e aos monitores de máquinas virtuais também chamados de *hypervisor* (VERAS; CARISSIMI, 2015).



Figura 1: Responsabilidade da virtualização (VERAS; CARISSIMI, 2015).

2.2 Máquinas virtuais

Uma máquina virtual é uma abstração em software de uma máquina física real. Destaca-se que é executada como uma aplicação padrão de usuário sobre um sistema operacional. A própria máquina virtual emula uma máquina física possuindo assim seus próprios discos e dispositivos (MCEWAN, 2002). Desse modo, umas das vantagens de máquinas virtuais reside na independência de uso do seu sistema operacional com relação ao sistema operacional da máquina física ao qual se encontra. Assim, em uma máquina física pode-se executar várias máquinas virtuais cada uma delas com sistemas operacionais

diversos. A Figura 2, apresenta um exemplo de arquitetura de um ambiente virtualizado, utilizando um *hypervisor* *VMware*. Nela, pode se vê que é na camada de virtualização que são providas as máquinas virtuais com seus respectivos sistemas operacionais e seus conjuntos de aplicações.

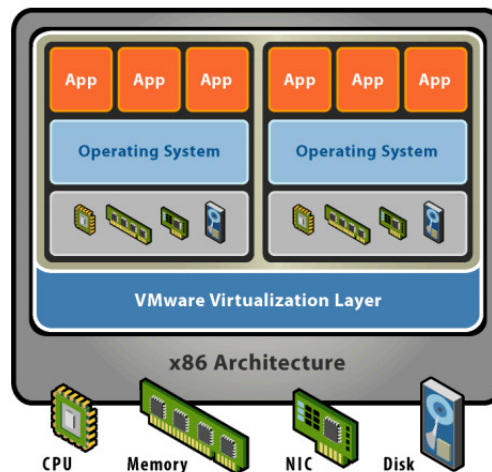


Figura 2: Representação da implementação de um ambiente virtual (JUNIOR, 2008).

2.3 Monitor de Máquinas virtuais

O *hypervisor* (ou também conhecido como monitor de máquinas virtuais) é o software que possui mecanismos capazes de prover máquinas virtuais. Suas principais funções consistem no escalonamento de tarefas, gerência da memória e manutenção do estado da máquina virtual (VERAS; CARISSIMI, 2015). Desse modo, atributos como desempenho e escalabilidade são determinantes para definir a qualidade dos serviços fornecidos por um *hypervisor*. Algumas características são essenciais a um *hypervisor*: segurança sobre os recursos virtualizados e agilidade na reconfiguração de recursos computacionais, sem interromper as operações do servidor de máquinas virtuais (VERAS; CARISSIMI, 2015). Os *hypervisors* são classificados em dois tipos:

- **Tipo I**(*bare metal*, nativo ou supervisor): executa diretamente no hardware do servidor. Controla o hardware e o acesso do sistema operacional convidado(*guest OS*). O papel do *hypervisor* nativo é compartilhar os recursos de hardware entre as máquinas virtuais, de forma que cada uma delas imagine ter recursos exclusivos (VERAS; CARISSIMI, 2015). Exemplos desse tipo incluem: *VMware ESXi*, *KVM*, *Citrix XenServer*, e *Microsoft Hyper-V*.
- **Tipo II**(*hosted*): aplicação que fornece um ambiente de execução para outras aplicações. Executa sob um sistema operacional nativo como se fosse um processo deste.

A camada de virtualização é composta por um sistema operacional hóspede e um hardware virtual, que são criados sobre os recursos de hardware oferecidos por meio do sistema operacional nativo (VERAS; CARISSIMI, 2015). Exemplos desse tipo incluem: *Oracle Virtual Box*, *VMware*, *workstation*.

2.4 Tipos de virtualização

A virtualização pode ser realizada de diferentes maneiras, cada uma com seus prós e contras. Na prática, em arquiteturas x86, as opções de virtualização alteram o nível de privilégios padrões. As soluções baseadas em *hypervisor* incluem a virtualização completa e a paravirtualização (VERAS; CARISSIMI, 2015).

Antes de falar sobre os tipos de virtualização, é necessário que seja feita uma breve explicação sobre os níveis (ou anéis) de privilégio existentes em arquiteturas x86. Os anéis de privilégios, são níveis de permissões utilizados no gerenciamento de acessos ao *hardware* de um sistema com arquitetura x86. Como apresentado na Figura 3, são definidos 4 anéis de privilégio. Desse modo, aplicações a nível de usuário são geralmente executadas no Anel 3. Enquanto que sistemas operacionais, que precisam ter acesso direto à memória e ao *hardware*, executam instruções privilegiadas no Anel 0 (VMWARE INC, 2008).

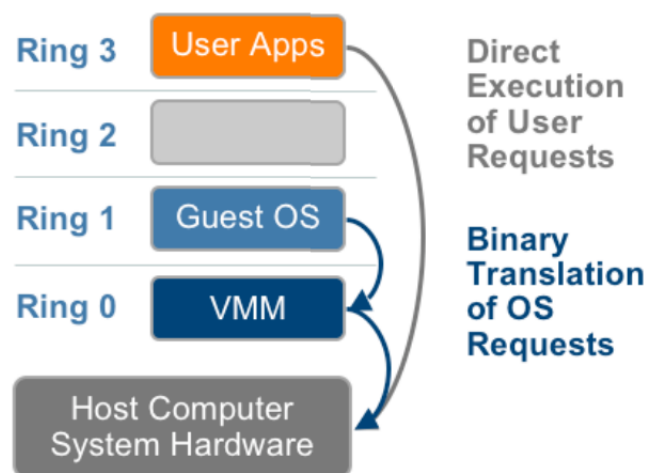


Figura 3: Representação da virtualização total
(VMWARE INC, 2008).

Na virtualização total, uma estrutura completa de hardware é virtualizada, portanto o sistema operacional a ser virtualizado (sistema operacional hóspede) não precisa sofrer qualquer tipo de alteração, sendo esses um dos principais benefícios dessa técnica (LAUREANO, 2006). Entretanto, o sistema virtualizado executa de forma mais lenta e o monitor de máquinas virtuais precisa implementar alternativas para que as operações

privilegiadas possam ser executadas em processadores que não suportem a virtualização nativamente (LAUREANO, 2006). Uma dessas alternativas é apresentada na Figura 4, que mostra uma abordagem de virtualização total implementada pelo *hypervisor* VMware. Nessa abordagem, cada instrução sensível é substituída por uma chamada a uma rotina VMware que a gerencia. Essa técnica é denominada de tradução binária. Já programas ou códigos que não possuem instrução sensíveis, são executados diretamente no *hardware* (TANENBAUM, 2007).

Nota-se na Figura 4, que o sistema operacional da máquina virtual (sistema operacional hóspede) está sendo executado em Anel 1, o que o impossibilitaria de executar instruções sensíveis. Entretanto, o *hypervisor*, que está em Anel 0, intercepta as instruções sensíveis e as emula através da técnica de tradução binária, permitindo dessa forma, que o sistema operacional hóspede, mesmo estando em Anel 1, execute instruções sensíveis.

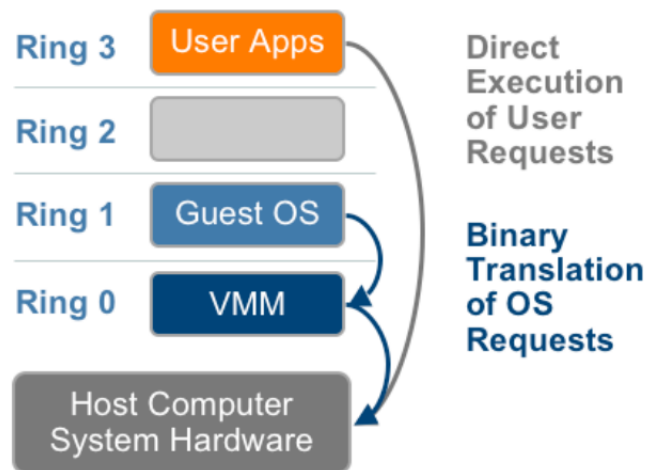


Figura 4: Representação da virtualização total (VMWARE INC, 2008).

Na paravirtualização, o sistema a ser virtualizado sofre modificações para que a interação com o monitor de máquinas virtuais seja mais eficiente, possibilitando que o sistema operacional hóspede consiga acessar recursos do hardware diretamente. O acesso é monitorado pelo monitor de máquinas virtuais, que fornece ao sistema convidado recursos do sistema, tais como endereços de memória que podem ser utilizados e endereçamento em disco, por exemplo (LAUREANO, 2006).

Dessa forma, como apresentado na Figura 5, o sistema operacional hóspede é modificado de forma a executar, no lugar das instruções sensíveis, chamadas de *hypervisor* (TANENBAUM, 2007). Destaca-se, que essa técnica faz com que o sistema operacional hóspede seja executado em Anel 0. A desvantagem dessa técnica de virtualização está relacionada, com a necessidade de se ter disponível o código fonte do sistema operacional hóspede, de modo que possa ser efetuadas as modificações necessárias para que, o torne

um sistema operacional paravirtualizado. Tornando difícil a aplicação dessa técnica em sistemas operacionais proprietários como o *Windows* (TANENBAUM, 2007).

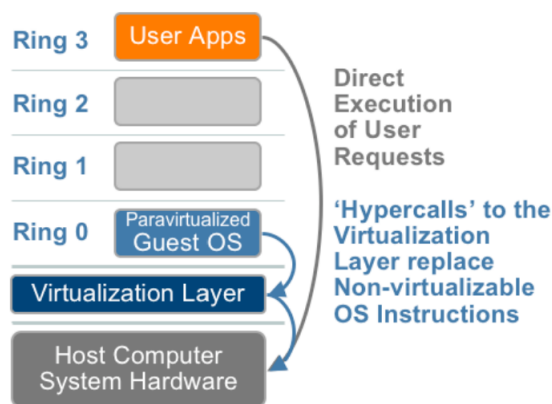


Figura 5: Representação da paravirtualização
(VMWARE INC, 2008).

2.5 Ferramentas de Virtualização

Nessa Seção serão abordadas algumas das ferramentas utilizadas para provimento de ambientes virtualizados: *Xen*, *KVM*. As ferramentas de virtualização basicamente são as plataformas responsáveis por prover a camada de virtualização responsável pela disponibilização de máquinas virtuais.

O *KVM* é uma solução de virtualização total voltada para arquiteturas x86, possui suporte para tecnologias de virtualização *Intel VT* e *AMD-V*. Foi incorporado ao *kernel* em janeiro de 2007, tornando-se assim um componente do *Linux* sendo capaz de herdar as funcionalidade principais do mesmo (RED HAT, 2015; QUMRANET, INC, 2006). Pelo fato do *KVM* ter sido incorporado ao kernel do *Linux*, o seu desenvolvimento passou a ter a colaboração ativa e o suporte da ampla comunidade do *Linux* bem como de algumas fornecedoras da indústria do software tais como, Red Hat, AMD, HP, IBM, Intel, Novell, Siemens, SGI entre outros (RED HAT, 2015).

O *KVM* é implementado como um processo convencional do *Linux*, de modo que cada *CPU* virtual aparece como um processo regular. Proporcionando assim, ao *KVM* os benefícios de todas as funcionalidades do *kernel* do *Linux* (RED HAT, 2015). Como apresentado na Figura 6, a emulação dos dispositivos e as operações de entrada e saída nos sistemas operacionais hóspedes, fica por conta de uma versão modificada do *QEMU* (RED HAT, 2015; QUMRANET, INC, 2006). Neste, caso o *KVM* intercepta eventos a níveis de sistema que precisam ser emulados, tais como leitura do disco ou o envio de um pacote pela rede, e invoca o *QEMU* que emula a funcionalidade do *hardware* requisitado

(RASMUSSEN; CORCORAN, 2014). Por exemplo, a placa de rede virtual da máquina virtual que precisasse ser usada para algum envio de pacotes pela rede, teria esse tipo de funcionalidade emulada a partir da placa de rede física da máquina hospedeira.

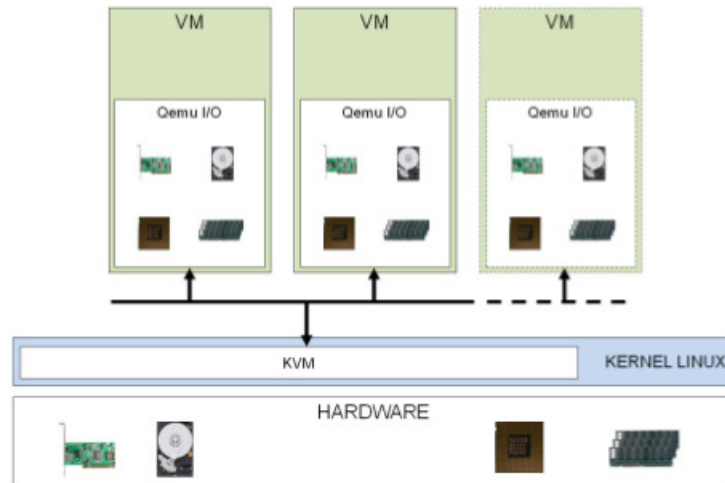


Figura 6: Arquitetura do KVM

(CARVALHO; BELLEZI, 2014).

Já o *Xen* foi criado por Keir Fraser e Ian Pratt como parte do projeto de pesquisa *Xenosever* pela *Cambridge University*. Sendo que em 2002, teve seu código fonte aberto afim de promover melhorias no mesmo com a contribuição da comunidade de desenvolvedores. É um dos mais populares *hypervisor* a implementar técnica de paravirtualização (XEN PROJECT, 2016). Desse modo é conhecido por possuir uma baixa perda de desempenho, se aproximando do desempenho nativa do servidor (WALTERS et al., 2008). Isso é justificado pelo fato de que o sistema operacional das máquinas virtuais é modificado de modo que as chamadas privilegiadas são substituídas por chamadas diretas ao *hypervisor*, ao contrário do que é feito com abordagens que utilizam emulação e tradução binária que acabam por focar no gerenciamento de chamadas privilegiadas (RED HAT, 2015).

A Figura 7 mostra que a arquitetura do *Xen* é composta por dois componentes: o próprio *hypervisor Xen* e pelo domínio 0 (ou *Dom0*). O *hypervisor Xen* é responsável por virtualização de memória e CPU, gerenciamento de energia e escalonamento das máquinas virtuais. Enquanto que o *Dom0* é uma máquina virtual instanciada pelo próprio *hypervisor Xen* que possui acesso direto ao hardware, sendo responsável por prover *drivers* dos dispositivos de E/S para máquinas virtuais (RED HAT, 2015). As máquinas virtuais são conhecidas como *DomU* (*unprivileged domain*), sendo que as operações feitas por dispositivos de E/S são realizadas através da comunicação entre o processo *front end*, existente no núcleo modificado das *DomU*, e o processo *back end*, existente no núcleo da *Dom0* (RED HAT, 2015).

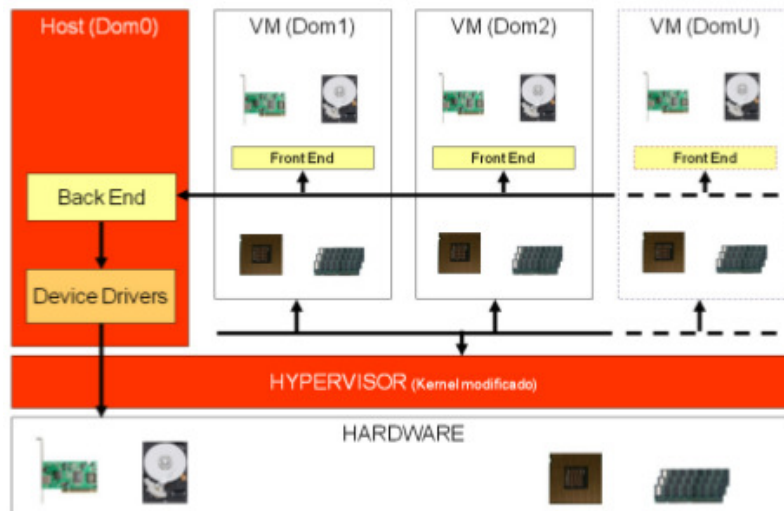


Figura 7: Arquitetura do XEN
(CARVALHO; BELLEZI, 2014).

2.6 Suporte Tecnológico

Nesta Seção são apresentadas as ferramentas utilizadas para a consolidação da infraestrutura do LAPPIS. Basicamente, as ferramentas utilizadas englobam o nicho de computação em nuvem *Cloudstack* e *OpenNebula* e automatização da infraestrutura utilizando *chef-solo* e *chake*.

2.6.1 Cloudstack

Em um modelo simplificado, o *Cloudstack* é composto de uma máquina de gerenciamento e dos recursos a serem gerenciados. Tais recursos compreendem: faixa de endereços *IP*, dispositivos *storage*, servidores e *VLAN'S*. Para implementação em uma configuração mínima, pode-se utilizar uma máquina dedicada apenas para a interface de gerenciamento, mantendo o servidor físico apenas com o *hypervisor*, ou utilizar o servidor físico executando a interface de gerenciamento e o *hypervisor* simultaneamente como apresentado na Figura 8.

Em modelo mais complexo, o *Cloudstack* apresenta seu potencial de disponibilidade escalabilidade e gerenciamento. Proporcionando uma modelagem de várias infraestruturas em nuvens em uma determinada região. Desse modo o *Cloudstack* possui os seguintes níveis de abstrações (HIGGINBOTTOM, 2013):

- **Regiões:** são a primeira e maior unidade de escala de uma implementação de uma *cloud* com *CloudStack*. Uma Região consiste em múltiplas Zonas de Disponibilidade, a segunda maior unidade de escala.

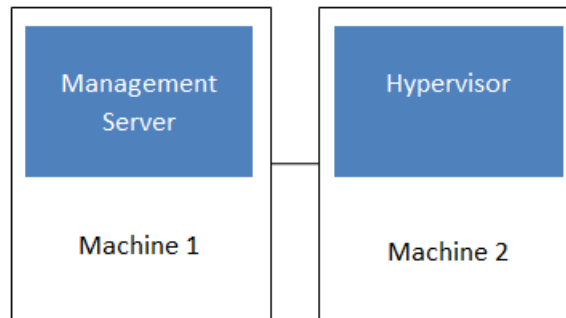


Figura 8: Visão simplificada de uma instalação mínima do cloudstack
([CLOUDSTACK, 2016](#)).

- **Zonas:** Tipicamente existe apenas uma Zona por *Data Center* e cada Zona contém PODs, *hosts* e *storage*.
- **Pods:** PODs têm propriedades lógicas e físicas com componentes como endereçamento IP e algoritmo de alocação de máquinas virtuais sendo influenciados por PODs dentro de uma Zona.
- **Clusters:** São simples grupos de servidores homogêneos combinados com um Storage Primário. Cada Cluster utiliza um mesmo tipo de hypervisor mas em uma Zona pode coexistir combinações de todos os hypervisores suportados. Cada *cluster* utiliza um mesmo tipo de *hypervisor* mas em uma Zona pode coexistir combinações de todos os *hypervisores* suportados.
- **Hosts:** Responsável por disponibilizar a camada de computação real em que Máquinas Virtuais são executadas.
- **Storage Primário:** onde os discos das Máquinas Virtuais residem e pode ser utilizado o disco local de um *host* ou um *storage* compartilhado como *NFS*, *iSCSI*, *Fiber Channel*, etc.
- **Storage Secundário:** onde é armazenado os *templates* de máquinas virtuais, arquivos ISO e *snapshots* e é utilizado o protocolo NFS para este *storage*.

A abstração desses níveis e como eles são alocados, são apresentados na Figura 9.

2.6.2 OpenNebula

O *OpenNebula*, assim como o *Cloudstack*, é uma ferramenta de código aberto que emergiu como um projeto de pesquisa em 2005 tendo seu primeiro lançamento público em março de 2008. Oferece uma solução simples mas repleta de funcionalidades para construir e gerenciar nuvens corporativas e *data centers* virtuais. Além disso, combina tecnologias de virtualização existentes com funcionalidades avançadas para fornecimento automático

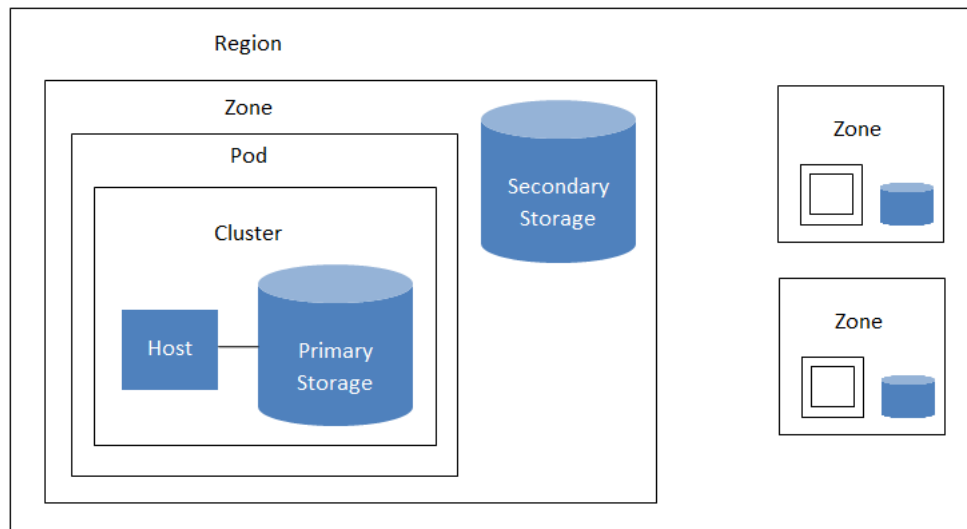


Figura 9: Visão geral da infraestrutura do *Cloudstack* (CLOUDSTACK, 2016).

e elasticidade, seguindo uma abordagem *bottom-up*, guiado pelas reais necessidades de administradores de sistemas e *devops* (OPENNEBULA, 2016).

Em uma configuração mínima, a arquitetura do *OpenNebula* é composta por três componentes: *hosts* (ou *worker-node*, *datastores* e *front-end*. Essa arquitetura é apresentada na Figura 10. O *frontend* é o ambiente responsável por disponibilizar a interface de gerenciamento. Através da rede, monitora os *hosts* e máquinas virtuais, bem como inicia operações relacionadas com máquinas virtuais e *datastores*. O *host* é servidor físico propriamente dito, onde serão criadas as máquinas virtuais com o suporte de um *hypervisor*. Por fim, o *datastore* é o *storage* utilizado como repositório de imagens e para manter os discos das máquinas virtuais em execução. A partir de *drivers* próprios do *OpenNebula*, é possível o uso de *LVM* e *Ceph* como gerenciadores de discos das máquinas virtuais. Não precisa ser necessariamente um *storage* dedicado, podendo ser uma máquina física com mais capacidade de disco ou até mesmo sendo um dos próprios *hosts*.

O *System Datastore* é uma abstração do *OpenNebula* que é responsável por manter os discos das máquinas virtuais em execução. Possui três tipos (OPENNEBULA, 2016):

- **shared** - O *System Datastore* é compartilhado entre todos os outros *hosts* usando *NFS*, por exemplo.
- **vmfs** - Necessário quando o *hypervisor* utilizado é o *VMware*. Uma versão da opção *shared* voltada para o sistemas de arquivos do *VMware*.
- **ssh** - Nesse caso, cada *host* possui seu próprio *System Datastore*.

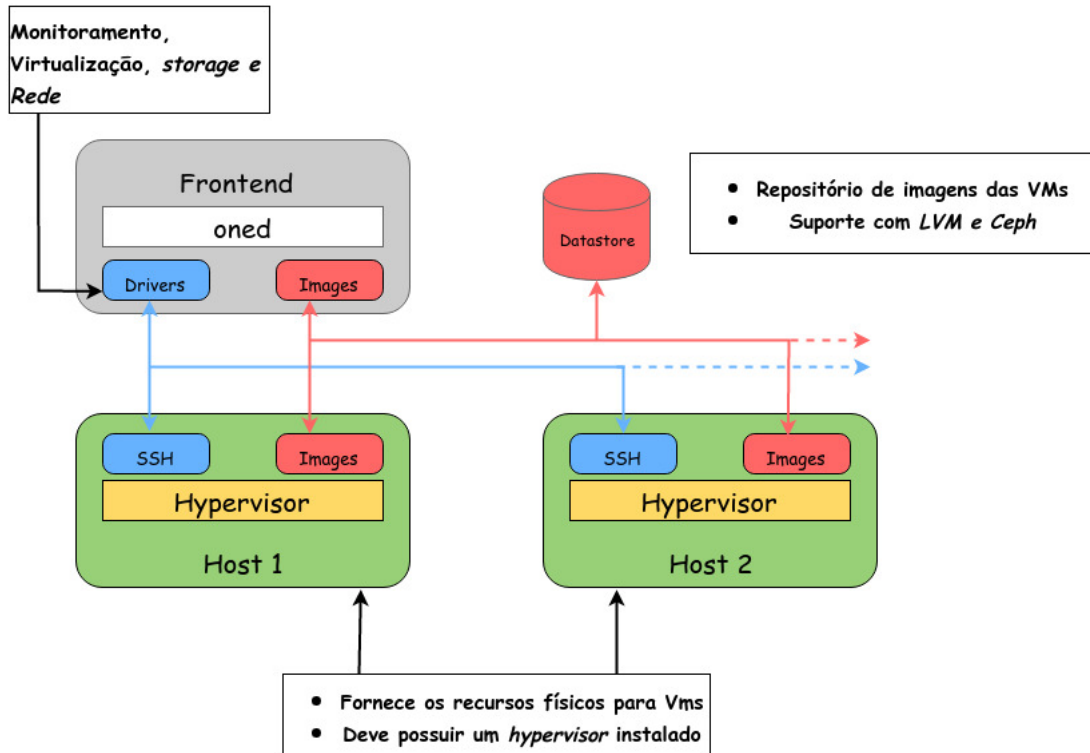


Figura 10: Visão geral da infraestrutura do OpenNebula
(OPENNEBULA, 2016).

As Figuras 11 e 12, apresentam uma abstração do *System Datastore* quando configurados como *shared* e *ssh*, respectivamente. Nota-se, que na opção *shared*, o *System Datastore* é centralizado no próprio *Datastore*, isso promove melhor desempenho na criação de máquinas virtuais. Entretanto, é necessário uma infraestrutura que possibilite uma boa velocidade de tráfego em rede, para que não haja perda de desempenho nas máquinas virtuais. Já na opção *ssh*, o *System Datastore* é mantido localmente não havendo riscos de perda de desempenho da máquina virtual por tráfego de rede, entretanto leva-se mais tempo para criação de máquinas virtuais do que na opção *shared*, dado que, a imagem da máquina virtual será transferida do *Datastore* para o servidor de destino.

O *OpenNebula* possui modelos de implementação tanto para nuvens privadas mais simplificadas quanto para ambientes de infraestrutura mais complexos. Dessa forma, para locais ou empresas que possuem múltiplos *data centers*, que por sua vez, possuem vários *clusters* de servidores, o *OpenNebula* prover funcionalidades que colaboram para que vários *data centers* separados regionalmente possam ser gerenciados a partir de uma interface em nuvem com acesso externo, essa integração é denominada de Federação.

A Figura 13 apresenta uma abstração de Federação composta por várias instâncias do *OpenNebula*. Cada instância do *OpenNebula* é denominada de zona, desse modo em uma infraestrutura com múltiplas zonas pode ser configuradas como uma Federação. Assim, tem-se um compartilhamento da base de dados entre as zonas (Usuários,

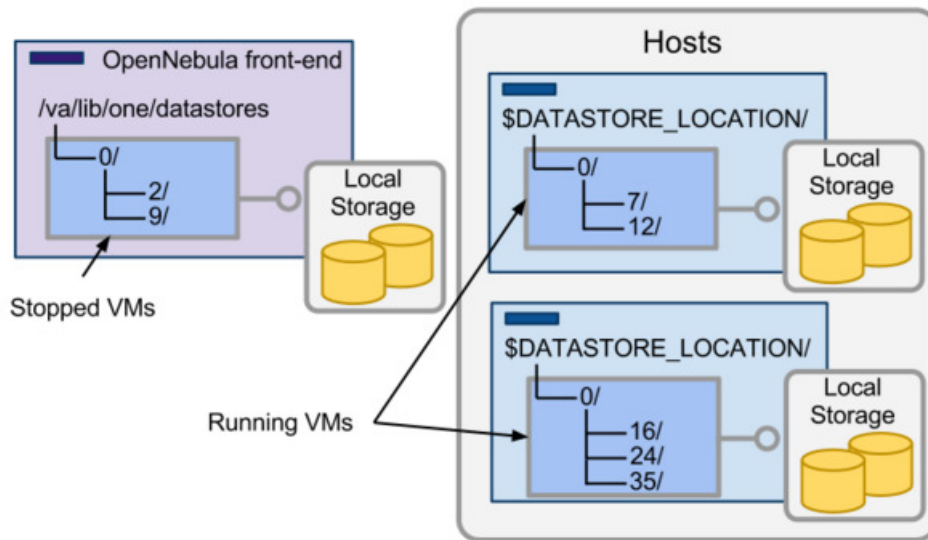


Figura 11: Funcionamento do *System Datastore* na configuração *SSH*.
(OPENNEBULA, 2016).

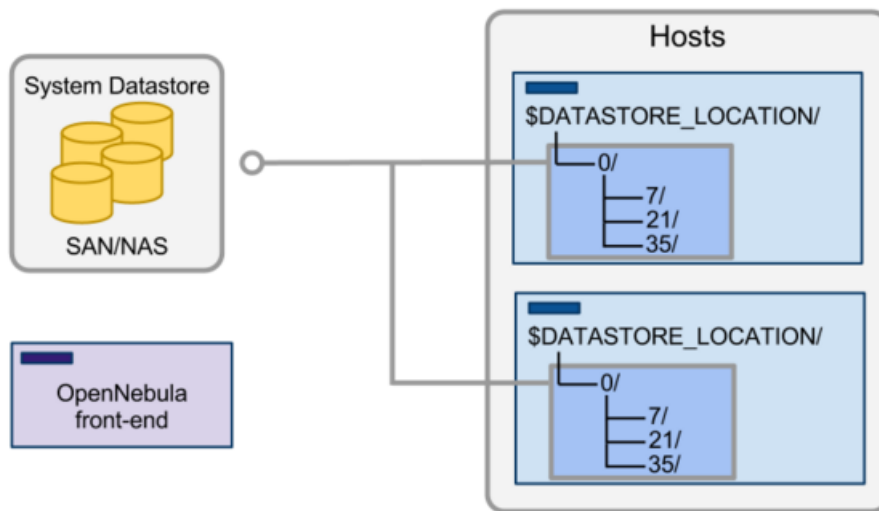


Figura 12: Funcionamento do *System Datastore* na configuração *shared*.
(OPENNEBULA, 2016).

grupos). Nessa configuração, uma das zonas tem o papel de *master*, ao qual é o responsável por escrever as informações na base dados, mantendo assim a consistência nos dados (OPENNEBULA, 2016).

2.6.3 Comparativo entre as ferramentas de Plataforma em nuvem

Nesta Seção é apresentado um breve comparativo entre as ferramentas *OpenNebula* e *Cloudstack* baseado no trabalho de Ismaeel et al. (2015). O *OpenNebula* tem como

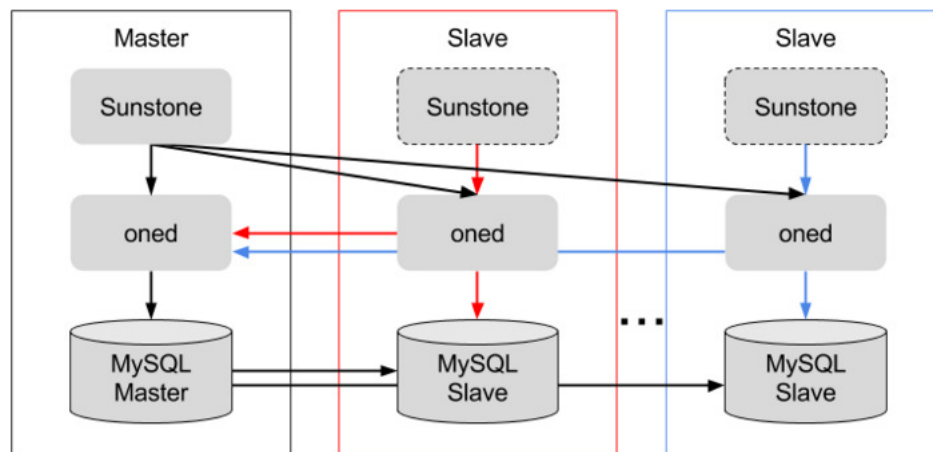


Figura 13: Arquitetura de implantação por zonas
(OPENNEBULA, 2016).

pontos fortes a flexibilidade e escalabilidade o que proporciona um certo dinamismo na adição de novos recursos. Já o *Cloudstack* provê uma API que proporciona facilidades no que diz respeito a integração de outras ferramentas, possibilitando bons mecanismos para configuração da plataforma como um todo (ISMAEEL et al., 2015). Entretanto, dos fatores apresentados na Tabela 1, a robustez contra erros foi decisivo para escolha em favor do OpenNebula, dado que, como é relatado no apêndice A, o *Cloudstack* mostrou-se bastante instável e intolerante a erros.

Tabela 1: Comparativo entre *Cloudstack* e *OpenNebula*
(ISMAEEL et al., 2015).

	OpenNebula	Cloudstack
Arquitetura	Árvore de módulos contendo todos os componentes	Servidor de Gerenciamento Central
Linguagem de Programação	Java, Ruby	Java, Python
Modelo de nuvem suportado	Pública, Privada e Híbrida	Pública, Privada
Hypervisor Suportado	VMware, LXC, KVM e Xen	libvirt, hyper-V, VMware, XenServer 6.2, baremetal, docker, Xen, LXC via libvirt
Transferência de dados	NFS or Secure Copy(SCP)	Fornecer uma ponte entre os usuários finais e a Área de armazenamento
Area de aplicação	Grande companhias comerciais e instituições públicas	Pequenas companhias comerciais e de pesquisa
Interface com o usuário	Linha de comando ¹	Interface web baseada no AJAX, gerencia requisições de sistemas para administradores e usuários.
Licença	Apache2	Apache2
Robustez contra erros	Banco permanente para guardar informação sobre servidores, redes e VMs	Limitado e centralizado
Sistema operacional	CentOS, Debian, OpenSUSE	CentOS, Debian, Fedora, RHEL, openSUSE, Ubuntu
Segurança	A interface web gera uma chave codificada pública/privada emparelhada para autenticação com o usuário	Integrado com LDAP e Active Directory, inclui diversos níveis de acesso
Compatibilidade com serviços em nuvem da Amazon	EC2, S3	EC2, S3

2.6.4 Automação da Infraestrutura

Para automação bem como documentação via código dos procedimentos adotados para implantação da plataforma e nuvem, e dos serviços disponibilizados pelo LAPPIS (apresentados no apêndice A.2, foi utilizado uma ferramenta de código aberto chamada *Chake*². O *Chake* é uma ferramenta que ajuda a gerenciar múltiplos ambientes, a partir do *Chef solo*. As configurações são geralmente implantadas via *rsync* com o auxílio do *SSH*, e aplicadas invocando o *Chef Solo* em cada ambiente. O uso dessas ferramentas contribui para gerência dos serviços oferecidos, promovendo também um meio de compartilhamento de conhecimento através do código.

As configurações são feitas a partir de arquivos denominados de recursos e receitas *Chef*. Os recursos descrevem algum pedaço da infraestrutura tal como um arquivo, template ou pacote. A receita *Chef* é um arquivo que contém os recursos relacionados à configuração de um servidor *web* ou de banco de dados. No código 2.1, é apresentado um exemplo de uma receita *Chef*. Na linha 1 é feita a instalação de um servidor *apache* usando a diretiva *package* que se equivale a um *yum install* ou *apt-get install*. Nas linhas 3 a 5 usa-se o recurso *service* para habilitar e iniciar o serviço do *apache*. Por fim, o recurso *template* define um arquivo de configuração, no caso *index.html.erb*, já pronto a ser utilizado na máquina alvo no endereço */var/www/html/index.html*.

```
1 package 'httpd'
2
3 service 'httpd' do
4   action [:enable, :start]
5 end
6
7 template '/var/www/html/index.html' do
8   source 'index.html.erb'
9 end
```

Código 2.1: Código exemplo de uma receita Chef

A escolha do *Chake* deve-se ao fato de utilizar como base o *Chef-solo*, que é uma ferramenta bastante consolidada no que diz respeito à automação da infraestrutura. Com isso é possível a disponibilização de serviços, sejam de infraestrutura ou de aplicações, de maneira automatizada. Outra vantagem reside no uso de uma linguagem de domínio específico para especificação dos recursos nas receitas *Chef*, o que promove a documentação via código.

¹ *OpenNebula* também prover uma interface *web*, possui um módulo responsável por isso chamado *Sunstone*

² <https://github.com/terceiro/chake>

3 Metodologia

Dado os conceitos que envolvem a diferença de desempenho entre a paravirtualização e a virtualização total e os problemas relacionados a interferência entre máquinas virtuais, neste capítulo serão definidas as abordagens que irão dar norte para o desenvolvimento deste trabalho. Desse modo, em um primeiro momento são apresentados os trabalhos relacionados à análise de desempenho em ambientes virtuais, de modo que é definida a abordagem mais adequada. Por fim, são apresentados as especificações voltadas para ambientes de testes e coleta de dados.

3.1 Trabalhos relacionados

Essa Seção tem como intuito expor alguns trabalhos relacionados à análise de desempenho. Esses trabalhos também servirão de insumo para desenvolvimento do estudo de interferência de desempenho proposto por este trabalho.

No trabalho apresentado por (KOH et al., 2007) é feito um estudo de interferência entre aplicações executadas sobre o mesmo *hardware* a partir de duas máquinas virtuais diferentes. Como cargas de trabalho, foram escolhidas aplicações do mundo real utilizadas para compressão, compilação de código fonte, e renderização de *frames*, bem como foram utilizadas ferramentas voltadas para testes de desempenho (*benchmark*).

Este trabalho mostra que duas aplicações, cada uma sendo executada em uma máquina virtual diferente, podem interferir no desempenho da outra. A análise de dados é baseada no cálculo de degradação de desempenho de uma aplicação, para o seguinte conjunto de métricas: média de uso de CPU, *cache hits*, *cache misses*, troca de máquinas virtuais por segundo, bloqueio de operações de entrada e saída por segundo, quantidade de requisição de leitura e escrita por segundo e tempo gasto na leitura e escrita para o disco da máquina virtual.

Além disso, é mostrado que com resultados desses desempenhos é possível fazer predição de desempenho de uma aplicação qualquer, a partir de análises estatísticas. Dada a quantidade variáveis, que são as métricas utilizadas para medição de desempenho, as análises estatísticas escolhidas foram a análise de componente principal (PCA) e a análise de regressão linear. Um comparativo entre as análises foi feito de modo que chegou-se a conclusão que análise por PCA apresentava uma porcentagem de erro menor. Esse trabalho demonstrou que a taxa de erro utilizando o modelo PCA se manteve igual, mesmo utilizando outros servidores físicos com configurações diferentes. Entretanto, uma das restrições deste trabalho é que o mesmo foi aplicado em aplicações que estavam sendo

executadas em duas máquinas virtuais. Desse modo, como trabalhos futuros é proposta a investigação no uso de mais máquinas virtuais, e aplicação de modelos não lineares para predição de desempenho bem como a adição de novos tipos de métricas que envolvam outras características a nível de sistema, tal como desempenho de aplicações de rede.

O trabalho de [Popiolek e Mendizabal \(2012\)](#), disserta sobre a importância de se utilizar métricas nativas (Tabela 2) de sistemas operacionais tais como *Linux* e *Windows* para detecção de gargalos de desempenho. Esse trabalho acaba focando na aferição de operações de entrada e saída, memória e uso de CPU, utilizando ferramentas de *benchmark* para geração de cargas de trabalho. Seus cenários de teste são basicamente variando de uma a seis máquinas virtuais, sendo que o *hypervisor* utilizado é o *KVM*. As análises permitem mostrar a queda de desempenho como um todo quando se tem o aumento do número de máquinas virtuais em execução (Figura 14). Como trabalho futuros, propõe-se que sejam feitas análise estatísticas afim de comprovar possíveis relações entre as métricas observadas, além de determinar o limiar que um sistema pode operar sem ter perda significativa no desempenho.

Tabela 2: Contadores de desempenho de disco para *Windows* e *Linux* ([POPIOLEK; MENDIZABAL, 2012](#))

Windows	Linux		Descricao
Monitor de Desempenho	iostat	df	
%Idle Time	-	-	Porcentagem de tempo que o disco permanece inativo
(Disk Bytes/sec)/ 1024	(rKB/s)+(wKB/s)	-	Número de Kilobytes lidos/escritos por segundo
Disk Transfers/sec	(r/s)+(w/s)	-	Número de requisições por segundo completadas
Split IO/sec	-	-	Número de requisições por segundo que foram divididas em múltiplas requisições
Free Megabytes	-	Disponível	Megabytes disponíveis para uso em unidade de armazenamento
Avg. Disk sec/Transfer	Await	-	Média de tempo para completar uma requisição
Avg. Disk Queue Length	avgqu-sz	-	Média de tamanho de filas de requisições esperando pelo disco rígido

Por fim, o trabalho de [Huber, Konev e Hauck \(2011\)](#) tem como intuito prover um

Os trabalhos de Koh et al. (2007) e Huber, Konev e Hauck (2011) tem como ponto em comum o desenvolvimento de mecanismos que possam prever o desempenho em ambientes virtualizados, uma característica interessante é que Huber, Konev e Hauck (2011) aplica os mesmos experimentos em *hypervisors* com arquiteturas distintas. Já o trabalho de Koh et al. (2007), apesar de fazer essa análise em apenas um *hypervisor*, acaba focando na interferência que determinadas aplicações utilizadas no cotidiano, dependendo de sua característica a nível de sistema, podem ocasionar em outras aplicações executadas em máquinas virtuais diferentes, no mesmo servidor físico, trazendo assim um experimento mais próximo do que acontece no mundo real, tornando isso um diferencial deste trabalho. Desse modo, ainda com relação ao trabalho de Koh et al. (2007), um questionamento que poderia ser feito é com relação a aplicabilidade deste experimento em um outro *hypervisor* com um tipo de virtualização diferente do apresentado no trabalho. Já o trabalho de Popiolek e Mendizabal (2012) é mais voltado para monitoramento, análise de desempenho e detecção de gargalos a partir de métricas nativas do sistema operacional não sendo feito, como nos outros trabalhos, qualquer iniciativa de predição de desempenho.

Dado os trabalhos relacionados apresentados, o de Koh et al. (2007) foi o que apresentou resultados mais significativos em um nível de detalhamento mais avançado, sendo dessa forma considerado o mais adequado para ser utilizado como insumo para desenvolvimento deste trabalho. Ainda mais, quando se pode verificar a extensibilidade do trabalho de Koh et al. (2007) para outro *hypervisor* como o KVM. Entretanto, em aspectos metodológicos, o trabalho se demonstrou bastante deficiente, dificultando de certo modo a replicação dos experimentos realizados. A partir disso, uma outra contribuição deste trabalho se comparado ao de Koh et al. (2007) é o detalhamento dos experimentos efetuados, de modo que o mesmo possa ser replicado em trabalhos futuros.

3.2 Ambiente de Testes

Para o estudo proposto por este trabalho foi definido um ambiente de testes que consiste no uso de máquinas virtuais com aplicações voltadas para realização de testes de *benchmark*. Tais aplicações foram definidas a partir do trabalho de Koh et al. (2007), tendo essas sido escolhidas visando o estresse de vários aspectos de sistema e de *hardware*. Com a infraestrutura de computação em nuvem implementada com o *OpenNebula* foi possível a criação desses ambientes de testes em máquinas virtuais criadas remotamente. As máquinas virtuais possuíam, como configuração, sistema operacional *Centos 7*, espaço em disco de 15GB e 1GB de memória *RAM*. Em um primeiro momento as aplicações foram instaladas e testadas de modo que se pudesse observar quais são os comandos utilizados para funcionamento das mesmas, em seguida foram criados *snapshots* das máquinas virtuais com o auxílio provido pelo *OpenNebula*.

Assim era possível criar, destruir e efetuar quaisquer tipos de testes com relativa comodidade. Entre as aplicações escolhidas estão típicas provedoras de *stress* computacional no cotidiano, tais como compilação de código fonte, compressão e criptação de arquivos e renderização de *frames*. Há também ferramentas voltadas para geração de testes de *benchmark* tais como *Cachebench* e *AIM Benchmark suite*. A seguir é feita uma breve descrição das ferramentas utilizadas.

- *Add_double* ¹ é um dos vários programas de testes de carga existentes no *AIM benchmark suite*. É responsável por medir operações de adição de dupla precisão.
- *Bzip2* ² e *Gzip* ³ são aplicações típicas para compressão e descompressão de arquivos. Com uso de arquivos grandes é possível gerar cargas de trabalhos usando essas ferramentas.
- *Ccrypt* ⁴ é uma ferramenta de código aberto voltada para encriptação e descriptação de arquivos. Foi desenvolvido com o intuito de substituir a aplicação padrão do *unix*, o *crypt*.
- *Cachebench* ⁵ é uma ferramenta de *benchmark* de código aberto desenvolvida para avaliar o desempenho do subsistema de memória. Atualmente é integrado *LLC-bench* (*Low-Level Characterization Benchmarks*).
- *Cat* e *Grep* são comandos padrões em sistemas *Linux* que são responsáveis por gerar requisições de leitura no disco. *Cat* é responsável por mostrar conteúdo de arquivos bem como combina-los e criar outros novos. Enquanto que *Grep* é utilizado para busca de palavras em arquivos texto.
- *cp* e *dd* são outros comandos padrões em sistemas *Linux*, neste caso, responsáveis por gerar atividades voltadas para escrita de disco. *cp* é utilizado para copiar arquivos e diretórios. O comando *dd* é utilizado para criação de imagens e cópias de arquivos.
- *Iozone* ⁶ é uma ferramenta de *benchmark* utilizada, voltada para testes de operações de disco, tais como leitura e escrita.
- *Make* é um comando nativo em sistemas operacionais *Linux*, responsável por automatizar um conjunto de procedimentos, principalmente a compilação de programas grandes que possuem vários arquivos com códigos fontes.
- *Povray* ⁷ é uma ferramenta de código aberto voltada para renderização de quadros com gráficos 3-D.

3.3 Coleta de dados

Dada a configuração do servidor apresentado no Apêndice A.2, um dos receios era de que os testes de *benchmark* feitos não fossem suficientes para apresentarem interferência entre as máquinas virtuais, desse modo optou-se por dividir a coleta de dados em três experimentos práticos de modo que, nos dois primeiros experimentos fossem verificadas as interferências entre os diversos tipos de aplicações. Assim, para cada experimento foi definido objetivos específicos a serem alcançados.

O primeiro experimento tinha como objetivo verificar a existência da interferência entre máquinas virtuais no servidor físico utilizado, sendo neste experimento executado com poucas ferramentas dado que os resultados eram mais voltados para verificação da interferência e motivação da continuidade do trabalho em si do que para uma análise mais profunda. No segundo experimento, teve como objetivo verificar a variação de interferência para o conjunto completo das ferramentas já apresentadas sendo construída uma matriz $n \times n$ com todas as possibilidades. No terceiro experimento, teve como foco a avaliação da interferência para diversos tipos de características a níveis de sistema tais como média de utilização de *CPU* e *leitura e escrita de disco por segundo*, por exemplo. Sendo os resultados deste terceiro e último experimentos utilizados como insumo para análise de dados apresentada neste trabalho.

3.3.1 Cálculo da Interferência

Os procedimentos adotados para o cálculo da interferência seguem os propostos no trabalho de Koh et al. (2007). Desse modo, duas máquinas virtuais, com as especificações e aplicações de *benchmark* apresentadas na Seção 3.2, são criadas em um servidor utilizando *hypervisor KVM*. Cada máquina virtual, denominadas '*dom1*' e '*dom2*' respectivamente, executa uma das aplicações de *benchmarking*.

Uma aplicação executando em *dom1* é chamada de aplicação *foreground*, e a que estiver executando em *dom2* é a aplicação *background*. Por questões de notação uma aplicação *foreground* executando contra uma aplicação *background* é denotada como F@B. Um dos procedimentos adotados é garantir que a aplicação *background* mantenha sua execução até que a aplicação *foreground* termine. Para o segundo e terceiro experimento cada aplicação é executada de modo que seja tanto *background* quanto *foreground*, sendo construída dessa forma uma matrix $n \times n$ com todos os possíveis resultados.

¹ AIM Benchmark (<http://sourceforge.net/projects/aimbench>)

² Bzip2 (<http://www.bzip.org/>)

³ Gzip (<http://www.gzip.org/>)

⁴ Ccrypt (<http://ccrypt.sourceforge.net/>)

⁵ Cachebench memory benchmark (<http://icl.cs.utk.edu/projects/llcbench/cachebench.html>)

⁶ IOzone Filesystem Benchmark (<http://www.iozone.org>)

⁷ The Persistence of Vision Raytracer (<http://www.povray.org>)

A fim de observar o quanto o desempenho é afetado pela interferência gerada por uma aplicação executada em outra máquina virtual, é feita a medida da degradação a partir do desempenho padrão de uma aplicação, essa medida denomina-se pontuação normalizada. Assim, para calcular a pontuação normalizada de uma aplicação, primeiro é definida a pontuação de desempenho inativa que é a pontuação de uma aplicação quando executada contra uma máquina virtual inativa, ou seja sem nenhuma aplicação executando. Então, em seguida é feito o cálculo da pontuação normalizada de uma aplicação F contra B , dividindo a pontuação de desempenho de F contra B pela pontuação de desempenho inativa de F . Assim define-se $NS(F@B)$, como sendo a pontuação normalizada de F contra B ,

$$NS(F@B) = PontuaçãoDesempenho(F@B) / PontuaçãoDesempenho(F@Inativo) \quad (3.1)$$

A partir disso, é feito cálculo do desempenho combinado de duas aplicações, F e B , em cada máquina virtual.

$$NS(F + B) = NS(F@B) + NS(B@F) \quad (3.2)$$

Sendo $NS(F@B)$ e $NS(B@F)$ medidos em dois testes separados. Para medida de desempenho, nos dois primeiros experimentos são utilizadas as pontuações geradas pelas próprias aplicações. Entretanto, algumas aplicações, aquelas que não são voltadas para *benchmark*, não geram pontuações explícitas. Para essas aplicações, fora definido como o inverso do tempo necessário para sua execução, como pontuação de desempenho. Na tabela 3 é apresentado o maior recurso utilizado bem como a medida de desempenho utilizada para cada ferramenta. Para o terceiro experimento as pontuações utilizadas são métricas de desempenho a nível de sistema.

Dessa forma, uma pontuação normalizada($F@B$) próxima ou igual a 1 é um resultado que indica que o desempenho de uma aplicação F contra B sofreu baixa degradação.

3.3.2 Métricas de desempenho a nível de sistema

No trabalho de Koh et al. (2007) são utilizadas métricas de desempenho a nível de sistema (denominadas *System-level Workload Characteristics*) de forma a analisar melhor o grau de interferência em diversos aspectos de sistema na máquina virtual. Segundo Koh et al. (2007), o fato desse tipo de métrica de desempenho ser independente de qualquer tipo de microarquitetura subjacente, garante que seja possível ser feitas comparações através dos diferentes tipos de servidores físicos.

No trabalho de Koh et al. (2007) essas métricas são obtidas através de um *hypervisor* instrumentalizado, de modo que elas são coletadas de fora da máquina virtual,

Tabela 3: Aplicações utilizadas para geração de cargas e trabalho

Nome	Maior Recurso Utilizado	Medida de Desempenho
Add_double	CPU	Pontuação
Bzip2	Misto	Tempo
Cat	Disco	Tempo
Cachebench	Memória	Pontuação
Ccrypt	Misto	Tempo
Cp	Disco	Tempo
Dd	Disco	Tempo
Grep	Disco	Tempo
Gzip	Misto	Tempo
Iozone	Disco	Pontuação
Make	Misto	Tempo
Povray	Misto	Tempo

possibilitando assim que a interferência da coleta seja praticamente nula. A partir disso, fora feita uma investigação de como o *hypervisor* utilizado, o *KVM*, ou a plataforma em nuvem *OpenNebula* poderia prover essas métricas sem que fosse necessário qualquer tipo de interferência na máquina virtual para coleta desses dados.

Para o *KVM* chegou-se a ferramentas como o *iperf-kvm* e *kvm-stat*. Entretanto, as informações apresentadas por essas ferramentas não eram claras e, para o caso do *kvm-stat*, não detalhava por máquina virtual e sim para *hypervisor* inteiro. Mesmo no *OpenNebula* as informações apresentadas consistiam em uso de espaço em disco, memória utilizada e quantidade de *CPU* alocado por máquina virtual, não sendo essas informações, relevantes para o estudo proposto. Assim, a abordagem escolhida foi o uso de uma ferramentas típica para monitoramento de desempenho: *iostat* para operações de disco e o *mpstat* para *CPU*.

- **Média de CPU**(*cpuutil*): É calculado dividindo o *tempo de CPU* alcançado por uma máquina virtual para executar uma aplicação, pelo tempo de execução dessa aplicação. O tempo de CPU é coletado pelo *libvirt*.
- **Requisições de escrita e leitura de disco por segundo** (*writes_issued*, *reads_issued*) e **Tempo gasto para escrita e leitura do disco**(*time_writing*, *time_reading*): Quantidade de requisições no disco e o tempo para operações de escrita e leitura são bons indicadores de operações de entrada e saída. Essa valores são coletados utilizando o *Munin*.

3.3.3 Procedimentos experimentais

Um dos impasses encontrados no desenvolvimento deste trabalho foi executar os mesmos procedimentos feitos por [Koh et al. \(2007\)](#), dado que não fica tão explícito quais foram os caminhos adotados para execução de cada aplicação ou ferramentas de *bench-*

mark. Dessa maneira, fora necessário definir experimentalmente os procedimentos a serem adotados para geração stress computacional e coleta de dados. De maneira geral, cada ferramenta era executada 15 vezes, em cada execução o tempo de execução ou a pontuação alcançada (para ferramentas que tinha pontuações explícitas) era coletado. Alcançada a quantidade de 15 execuções era calculada uma média dos dados coletados.

Inicialmente foi definido um número de 30 execuções por ferramentas (uma amostragem de 30 valores). Entretanto, principalmente no segundo experimento, a estimativa de tempo para execução de todas ferramentas se mostrou demasiadamente grande. Dessa forma, cogitou-se o uso de 10 ou 15 valores. De modo a chegar a um valor adequado para amostragem, levando se conta a quantidade o tempo necessário para essa quantidade. Calculou-se o desvio padrão das ferramentas *Make* e *Bzip* para 10, 15 e 30 valores de amostragem, a fim de verificar a diferença de dispersão entre esses valores. A tabela 4 apresenta esses resultados.

Tabela 4: Desvio Padrão para diferentes quantidades de valores para *Povray* e *Make*

Ferramenta	Quantidade de Valores	Desvio Padrão	Média
Make	10	0.993032191489	44.5126
	15	0.436379367501	44.3846666667
	30	0.379019873315	44.3230666667
Gzip	10	0.394128701484	31.9069
	15	0.229156361673	32.0740666667
	30	0.249032463378	32.1289333333

A diferença do desvio padrão para uma amostragem entre 15 e 30 valores foi considerada irrelevante se comparada com a média alcançada dos resultados. O desvio padrão para uma amostragem de 10 valores mostrou uma dispersão mais elevada, mesmo assim ainda irrelevante a diferença se comparada com a média, mas visando uma margem mais segura de erro, acabou-se definido o uso de uma amostragem de 15 valores.

Para geração de carga de trabalho a partir das ferramentas selecionadas, alguns procedimentos foram definidos empiricamente, principalmente para aplicações utilizadas no cotidiano. A seguir são descritos alguns procedimentos adotados para cada ferramenta selecionada.

Para a coleta de dados referentes às métricas de desempenho a nível de sistema, foi definido uma coleta de 300 valores por cada métrica.

- *Add_double*: A única opção para execução dessa ferramenta de *Benchmarking* é a quantidade de tempo que é executada. Foi escolhido um tempo de 30 segundos.
- *Bzip2*, *cp*, *cat*, *gzip*: Para essas ferramentas foi definido o uso de um arquivo com tamanho de 3 GB (o triplo da memória da máquina virtual) de modo que se evitasse o efeito de *cache* pela memória. Desse modo, os comandos *cat* e *cp* geravam cargas

a partir da geração de cópias desse arquivo. Aplicações *Bzip2* e *gzip*, efetuavam a compreensão desse arquivo com a opção *-best*.

- *Cachebench*: Possui diferentes tipos de testes de *benchmark* para o sistema de memória, sendo executados para tamanhos variáveis de alocação de memória. É executado com as opções: *-b* (*Read/Modify/Write benchmark*), *-x0*, *-m24*, *-d1* (1 segundo por iteração), *-e1* (1 repetição do teste por tamanho de alocação da memória) . Como resultado é considerado apenas o valor da taxa de transferência em *MB/s* apresentado para o maior tamanho de alocação de memória que a ferramenta suporta.
- *Ccrypt*, *grep*: Para essas ferramentas foi utilizado um arquivo de *2GB*, gerado com números *randômicos* a partir de */dev/random*. Com *Ccrypt* é efetuada a encriptação do arquivo, com o *grep* é feita uma busca pelo número *'123'*.
- *Iozone*: Essa ferramenta foi executada de modo que se efetuasse os testes de leitura e escrita no modo sequencial e randômico.
- *Make*: Para geração de carga foi efetuada a compilação do código fonte da versão mais recente do *apache* ⁸ . Coletado o tempo para execução da compilação, a mesma é desfeita para que em seguida seja executada novamente, obtendo assim a quantidade de valores necessária para o cálculo da média.
- *Povray*: Possui um módulo próprio para execução de testes de *benchmark*. O próprio módulo de *benchmark* já contabiliza o tempo de execução dos cenários de renderização de *frame* da ferramenta.

Para coleta de dados de dados foi construídos *scripts*, utilizando a linguagem *Python* ⁹ com a quantidade de execução bem como os comandos utilizados para cada aplicação e ferramenta de *benchmark*, de modo a garantir mais comodidade na obtenção dos dados. Esses *scripts* estão disponíveis via repositório remoto ¹⁰.

⁸ <http://www.apache.org/>

⁹ <https://www.python.org/>

¹⁰ <https://github.com/MaxAlmeida/TCC-SCRIPTS-MONITORING>

4 Análise de Resultados

4.1 Interferência de Desempenho

No primeiro experimento tinha como intuito, observar se de fato havia algum tipo de interferência significativa durante a execução de aplicações em máquinas virtuais concorrentes. Para isso, foram selecionadas as seguintes ferramentas: *bzip2*, *grep*, *povray*, *crypt* e *cp* para calculo da degradação. A Figura 16 apresenta os resultados para esse conjunto de aplicações, utilizando a equação 3.2.

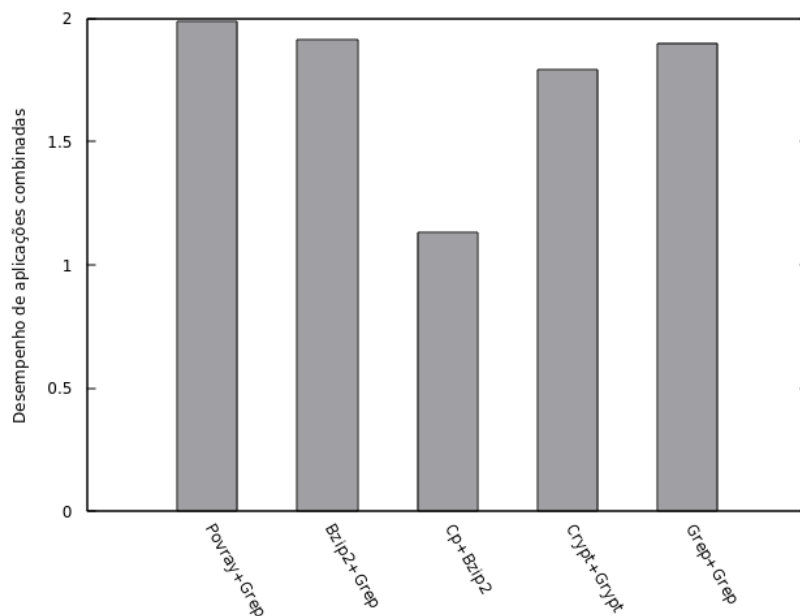


Figura 16: Pontuação combinada de aplicações selecionadas para o primeiro experimento

Levando-se em conta que cada aplicação consome exatamente metade dos recursos computacionais, o desempenho esperado seria 1. Entretanto, por conta da variação de interferência dada a combinação de aplicações, o desempenho também muda. Desse modo, aplicações que raramente interferem no desempenho uma da outra, a pontuação se aproxima de 2. Em contrapartida, aplicações que interferem substancialmente uma na outra, sua pontuação tende a ser baixa. Os resultados apresentados neste experimento, mostram por exemplo que *povray+grep* apresentam um grau de interferência praticamente nulo, enquanto que *cp + bzip2* o grau de interferência é maior a ponto da pontuação se aproximar de 1. Desse modo, era de se esperar, que dados os resultados referentes a interferência de aplicações apresentados no trabalho de Koh et al. (2007), que o desempenho combinado de aplicações iguais caísse drasticamente. Uma das hipóteses é que a configuração de *hardware* utilizada neste trabalho, típica para ambientes em produção, minimiza o grau

de degradação de algumas aplicações, mesmo quando executadas contra elas mesmas em máquinas virtuais diferentes. Em específico, foi utilizado nesse servidor uma configuração de redundância de disco *RAID 5*, o que pode minimizar ainda mais a degradação de desempenho para aplicações com perfis de *Entrada/Saída* como *Grep* e *Crypt*.

Um dos receios era que os procedimentos definidos na metodologia não fossem suficientes, dada a configuração de *hardware* do servidor, para observar a degradação de desempenho das aplicações. Entretanto, com resultados desse experimento demonstraram que com esses procedimentos é possível observar um grau de interferência para um conjunto de aplicações. O próximo passo então era observar o grau de interferência para um conjunto maior de aplicações afim de observar padrões de interferências dado o perfil de execução das aplicações (disco, *cpu* e memória).

4.2 Desempenho contra diferentes tipos de Aplicações

No segundo experimento foi construída uma matriz $n \times n$ com as combinações possíveis das aplicações sendo executadas tanto quanto *Background* como *Foreground*. Desse modo, contou-se com mais combinações de resultados sendo possível observar se existia ou não um padrão de degradação dado o tipo de aplicação que estava sendo executada. A figura 17 apresenta esses resultados.

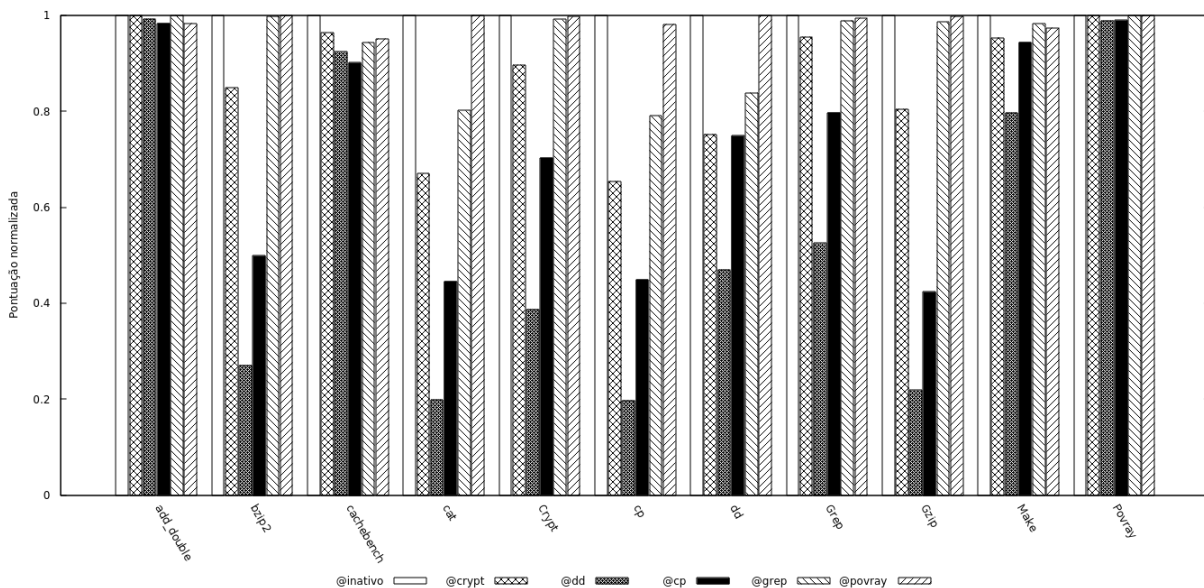


Figura 17: Pontuação normalizada contra um conjunto de aplicações

Os resultados deste segundo experimento mostram que de maneira geral algumas aplicações tendem a sofrer e a causar menos interferência do que outras aplicações. É o caso de *add_double*, *cachebench*, *make* e *povray*. Não por acaso, são ferramentas que não possuem perfis de cargas de trabalho voltados para operações de *entrada/saída* (

add_double possui perfil voltado para *cpu*, *make* e *povray* possuem perfil misto mas mais voltado para operações em *cpu* do que de *entrada/saída* e *cachebench* é mais voltado para operações no sistema de memória).

Assim, nota-se que essas aplicações, por possuírem um perfil de carga de trabalho voltado mais para *cpu* e memória (no caso do *cachebench*), praticamente não interferem em aplicações que possuem um perfil voltado para *operações de entrada/saída* e nem mesmo sofrem interferência significativa desse tipo de aplicações. Como pode ser observado, por exemplo, os resultados apresentados na execução de *povray* contra um conjunto de ferramentas, na Figura 17. Nota-se que a pontuação de *povray* contra esse conjunto de aplicações ficou próxima de 1 o que indica o grau de interferência bastante insignificante, o mesmo ocorrendo com *add_double*.

No caso do *cachebench*, mesmo tendo sido uma das ferramentas que menos sofreu interferência, percebe-se um grau de interferência mais elevado do que das outras duas (*povray* e *add_double*), isso se deve ao fato que o sistema de memória acaba atuando principalmente em operações de *cache* tanto em aplicações com perfil voltado para *cpu* quanto em aplicações com perfil voltado para operações de *entrada/saída*.

Para o restante das aplicações, observa-se um grau de interferência maior e com mais variações dependendo da aplicação que está sendo executada em *background*. Essas aplicações, são típicas aplicações de *entrada/saída*, com suas cargas de trabalho atuando em operações de escrita ou leitura em disco. Observa-se por exemplo, que *dd* e *cp* são as aplicações que causam maior degradação no desempenho das aplicações voltadas para *entrada/saída*. Dessa forma em uma das combinações apresentadas mostra que, executando *Bzip2* contra *dd*, a pontuação fica abaixo de 0.4, evidenciando um elevado grau de degradação.

Dada a variação de interferência de desempenho entre essas aplicações, vale ressaltar alguns casos. Entre eles, destaca-se por exemplo que a pontuação alcançada de *gzip* e *bzip2* contra *grep* indica um grau de degradação de desempenho pequeno ou praticamente nulo. Em contrapartida, a pontuação alcançada por *cp*, *cat* e *dd* contra *grep*, mostram uma degradação do desempenho maior se comparada com *gzip* e *bzip2* contra *grep*. Indicando, dessa forma que mesmo dentre as aplicações típicas de *entrada/saída* pode haver um padrão de interferência dado o seu tipo de execução(escrita ou leitura).

A Figura 18 apresenta os resultados de interferência para um outra combinação de aplicações. Nota-se que mesmo executando *add_double@add_double* a interferência apresentada é praticamente inexistente, evidenciando assim que as cargas de trabalhos aplicados para *cpu* não estavam sendo suficientes para que algum tipo de interferência fosse observável. Mais uma vez, nota-se maior degradação no desempenho geral em aplicações que possui um perfil de *entrada/saída*.

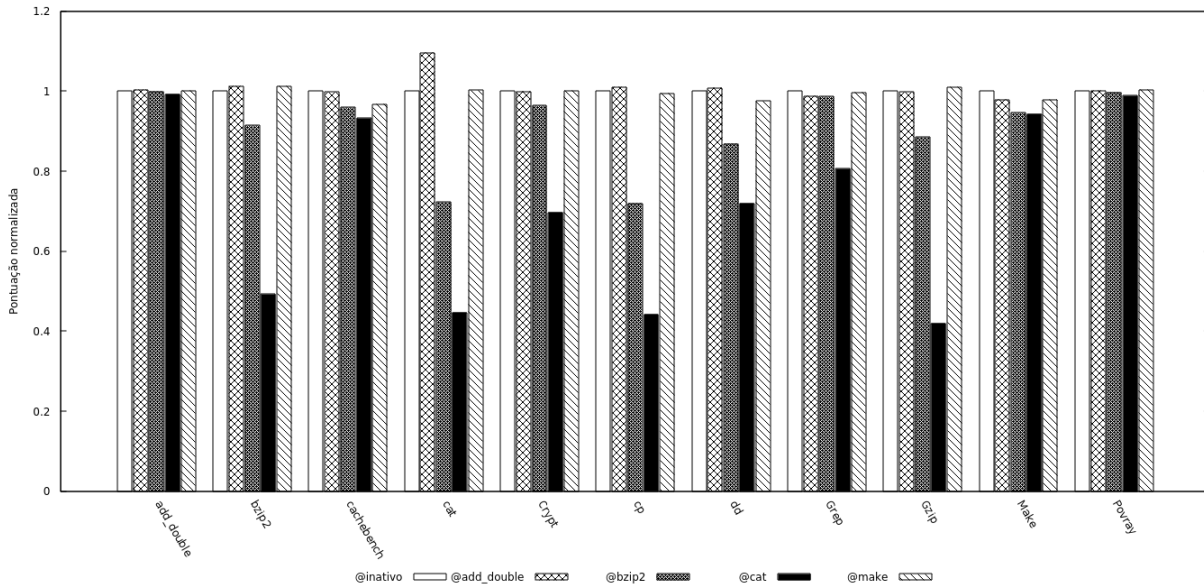


Figura 18: Variação de desempenho para um conjunto de aplicações

Os resultados deste segundo experimento demonstraram que o perfil de execução de aplicação influenciam de maneira considerável no desempenho de uma aplicação que está sendo executada em uma outra máquina virtual. Observa-se que a tendência é que aplicações com perfil voltadas para uso de disco sofram pouca interferência de aplicações com perfil de uso mais voltado para *cpu* e memória. Em contrapartida, seus desempenhos caem consideravelmente quando executadas contra outras aplicações com perfil de uso voltado para disco. Uma hipótese levantada neste experimento é pode haver um padrão de degradação dado o tipo de operação de disco ao qual as aplicações são voltadas (leitura ou escrita). Com esses resultados, chegou-se a conclusão que os testes definidos para *cpu* são insuficientes para que se observe algum tipo de degradação nesse perfil de execução.

4.3 Interferência nas Métricas de Desempenho a nível de Sistema

O terceiro experimento consistiu em coletar os dados referentes as métricas de desempenho a nível desse sistema. Desse modo, foi possível observar o perfil de execução das aplicações bem como avaliar como ocorre o impacto da degradação do desempenho nesses tipos de métricas. Como observado no experimento 2, algumas aplicações sofreram pouca degradação em seus desempenhos bem como interferiu muito pouco contra outras aplicações. Dessa forma, optou-se por restringir esse experimento às ferramentas aos quais foram observados o grau de interferência maior, sendo removidas então deste terceiro experimento *make*, *povray*, *add_double* e *cachebench*.

As imagens 19, 20 e 21 mostram o desempenho das aplicações quando executadas contra um servidor inativo. Os dados apresentados permite observar o perfil de execução

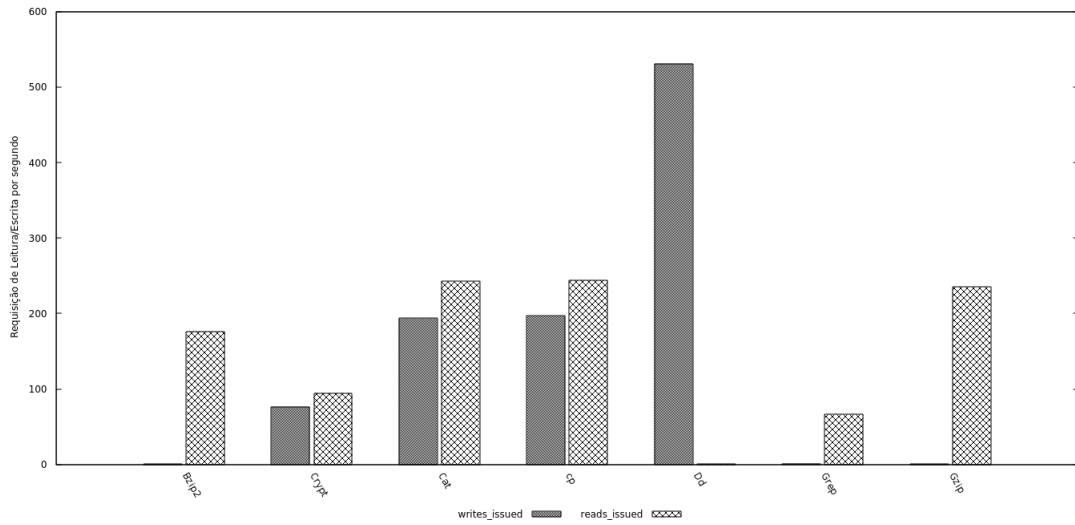


Figura 19: Desempenho alcançado das aplicações para requisições de escrita e leitura em disco

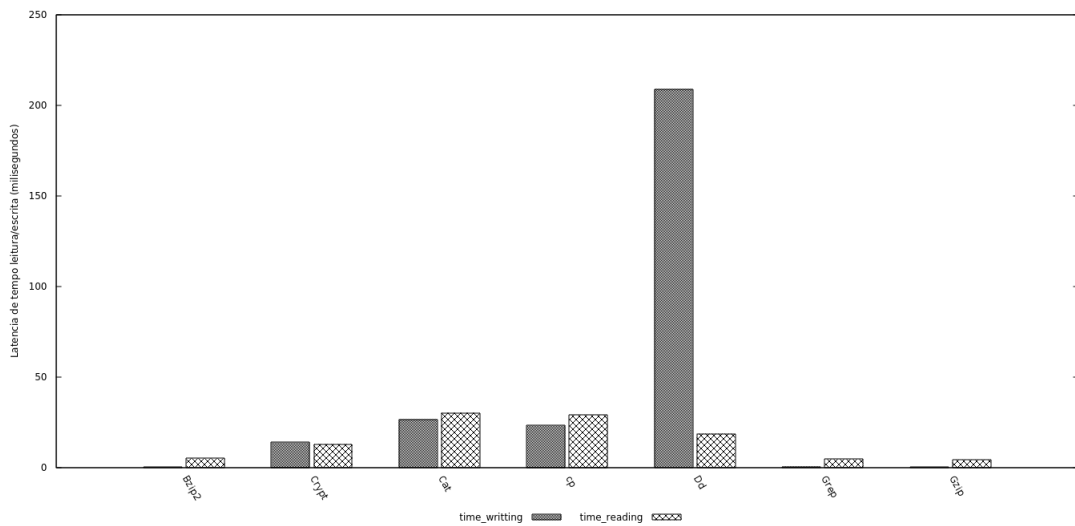


Figura 20: Tempo que as aplicações levam para executar uma operação de leitura e escrita em disco

de algumas aplicações a partir da pontuação alcançada nas métricas de desempenho a nível de sistema. Percebe-se dessa forma que aplicações como *crypt*, *cat*, *cp* possui um perfil misto nas operações em disco, ou seja, executam de forma quase igualitária tanto operações de leitura quanto operações de escrita em disco. Em contrapartida, aplicações como *dd*, *grep*, *gzip* e *Bzip2* possuem um perfil único no que tange as operações em disco, com *dd* tendo uma elevada pontuação para operações de escrita de disco, e as outras três possuindo um perfil voltado mais para leitura em disco. O perfil de execução dessas aplicações para as métricas de leitura e escrita em disco reflete o perfil para o tempo de escrita e de leitura em disco, mantendo dessa forma, o mesmo padrão de desempenho para essas duas últimas métricas. Para porcentagem de utilização de *cpu*, observa-se que as aplicações *bzip2*, *crypt*, *grep* e *gzip* possui taxas elevadas de utilização de *cpu*, mostrando

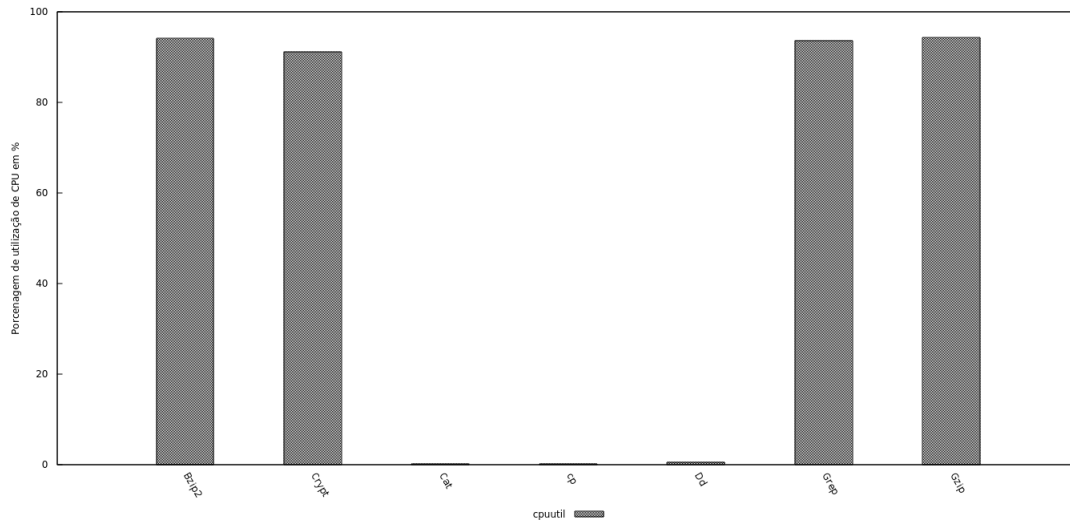


Figura 21: Porcentagem de utilização de cpu para cada aplicação

assim um perfil misto de execução com relação à utilização de *cpu* e *disco*. Já *cat*, *cp* e *dd* apresentam baixas taxas de utilização de *cpu* evidenciando dessa forma seu perfil voltado para utilização de disco.

No experimento 2 observou-se que algumas aplicações podem sofrer maior degradação dependendo da aplicação que está sendo executada como *background*. *Bzip2*, por exemplo, obteve uma queda de desempenho acentuada executando contra *cp*, *cat* e *dd*. Já contra *Grep* e *crypt*, obteve pontuação relativamente alta. A Figura 22 mostra como se dá essa degradação nas métricas a nível de sistema de *bzip2* contra um conjunto de aplicações.

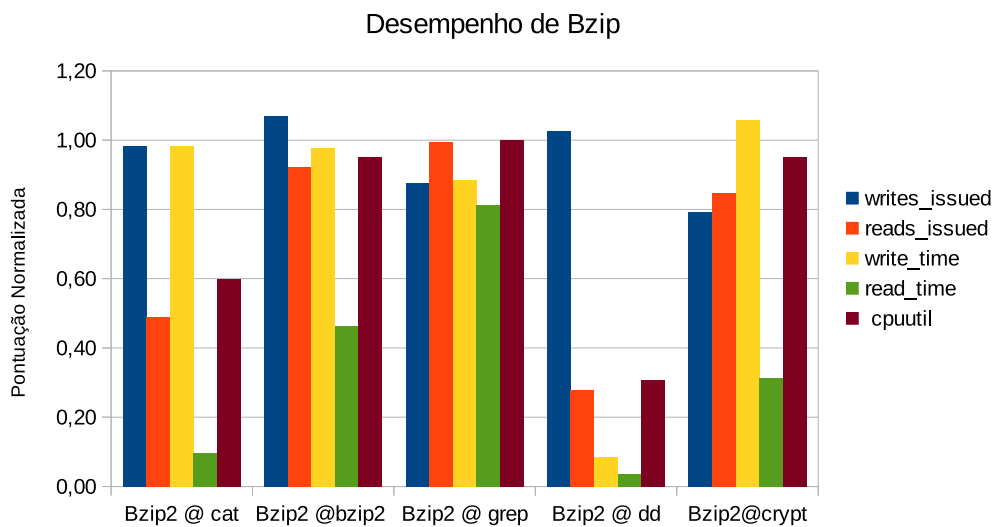


Figura 22: Pontuação normalizada das métricas a nível de sistema de *Bzip2* contra diferentes tipos de aplicações.

As pontuações normalizadas de algumas métricas ultrapassam o valor de 1, o que teoricamente seria impossível dado que o desempenho, contra uma aplicação *background*, tende a ser menor ou igual a 1 dependendo da aplicação. Entretanto, as medições $F@B$ e $F@Inativo$ são feitas em experimentos diferentes. Dessa forma, observou-se que, mesmo coletando uma amostra de 300 valores para cada métrica, pode haver uma pequena variação de experimento para experimento. Assim, uma pontuação de uma métrica que não sofre degradação contra uma determinada aplicação *background* ($F@B$), pode naquele momento alcançar um valor um pouco maior do que quando foi executando contra uma máquina virtual inativa ($F@Inativo$), resultando em uma pontuação normalizada maior que 1 de acordo com a equação 3.1.

Nota-se, portanto, que *bzip2* contra *cat* resulta uma interferência nula na métrica *writes_issued*, dada a pontuação normalizada desta, por outro lado a pontuação normalizada de *cpuutil*, *writes_issued* e *read_time* caíram drasticamente. A queda de *cpuutil* em *bzip2*, mesmo com *cat* possuindo baixa taxa de utilização de *cpu*, pode ser explicada pela pontuação alcançada por *read_time*. Uma pontuação normalizada de *read_time* mais baixa indica que *bzip2* agora está levando mais tempo para realizar poucas requisições de disco, fazendo que a *cpu* fique mais tempo ociosa esperando por operações de *entrada/saída*. Por possuir um perfil mais voltado para operações de leitura em disco, o desempenho de *bzip2* tende a ser fortemente influenciado com a queda de valores de *reads_issued* e *read_time*.

Executando *Bzip2*, *grep* e *crypt* como *background*, *Bzip2* tende a ter um desempenho melhor, tendo apenas como maior degradação a métrica *reads_issued*, sendo possivelmente a causa da pequena degradação sofrida por *Bzip2* quando executado contra essas aplicações, como apresentado nas imagens 17 e 18. Com *dd* como *background*, *Bzip2* possui uma degradação em seu desempenho semelhante à quando executada contra *cat*. Novamente, têm-se uma queda acentuada nas pontuações normalizadas de *cpuutil*, *reads_issued* e *read_time*.

A figura 23 apresenta a variação da degradação sofrida por *dd* para cada tipo de métrica. *cp* possui uma degradação de desempenho semelhante tanto para as aplicações com perfis mistos quanto para aplicações voltadas para operações em disco, com exceção do caso de quando ele é executado contra ele mesmo. Sua pontuação, como apresentado nas imagens 18 e 17, fica na faixa de 0.7 ~0.85 contra aplicações de perfil misto e de disco e próximo de 0.5 quando executado contra ele mesmo, e isso se reflete em suas métricas. Principalmente *writes_issued*, que mantém esse mesmo padrão de degradação evidenciando dessa forma que o desempenho de *dd* é fortemente influenciado por essa métrica, visto que, *dd* possui um perfil voltado para escrita de disco.

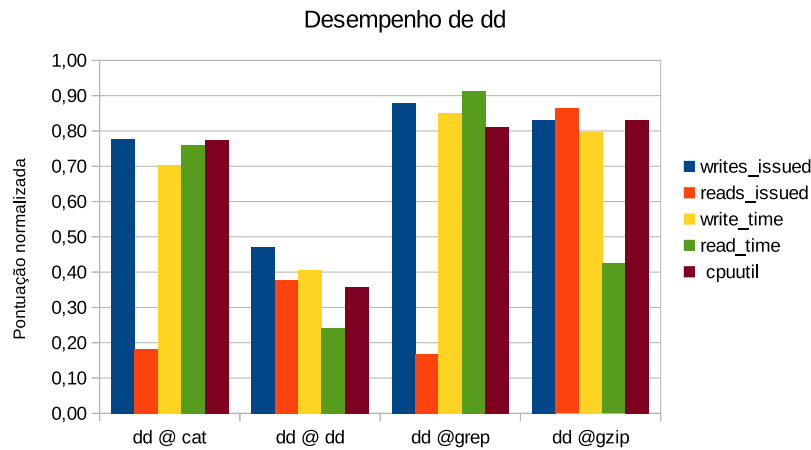


Figura 23: Pontuação normalizada das métricas a nível de sistema de *dd* contra diferentes aplicações.

A figura 24 apresenta a degradação sofrida por *grep* em suas métricas indicadoras de desempenho para cada aplicação. Observa-se que mais uma vez, que o desempenho de uma aplicação é determinado pelo grau de degradação sofrido pela sua métrica de maior uso. Assim, executando *grep* contra *dd*, sua pontuação normalizada de *writes_issued* alcança 0.76 enquanto que *reads_issued* atinge 0.54 e, levando-se em conta seu tempo de execução, a pontuação atinge 0.52 como mostrado na Figura 17, evidenciando que a degradação sofrida por *grep* advém de valores de *reads_issued* mais baixos. Nota-se, também que *grep* tem maiores perdas de desempenho para *reads_issued* quando executado contra, *cp*, *cat* e *dd* do que contra *Bzip2* e *crypt*. Isso se deve ao fato de que as três primeiras aplicações geram mais requisições de disco (leitura e escrita) do que as duas últimas.

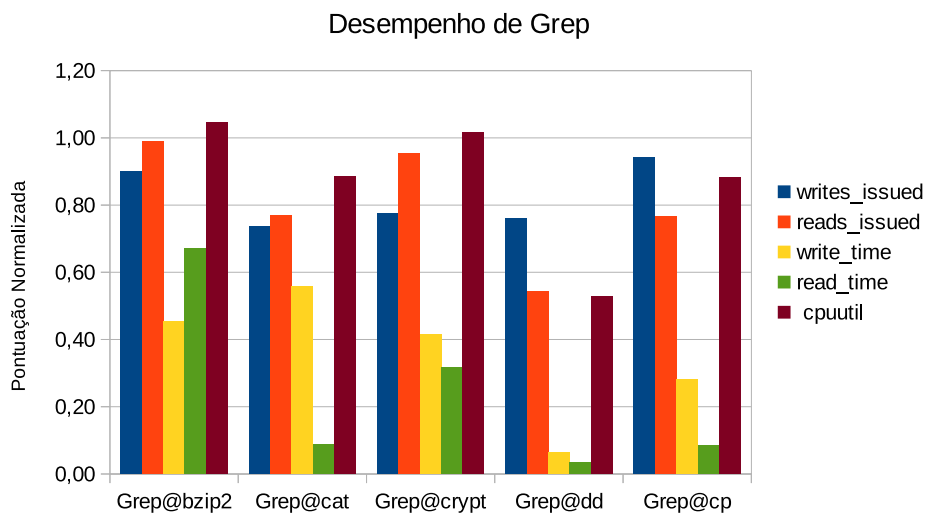


Figura 24: Pontuação normalizada das métricas a nível de sistema de *Grep* contra diferentes aplicações.

Com os resultados deste experimento foi possível traçar um perfil de execução para cada ferramenta, bem como observar como o seu perfil de execução influencia bastante em seu desempenho final quando executado contra uma aplicação em *background*. Uma das hipóteses levantadas na Seção 4.2 era de que poderia haver um padrão de degradação dentre as aplicações com perfil voltado para operações de disco. Com os resultados apresentados tal hipótese não pode ser confirmada dado que o que se observou foi que o que causa maior degradação para esse conjunto de aplicações são as taxas mais elevadas de requisições no disco que uma aplicação *background* pode gerar. De maneira geral, percebe-se que aplicações que possuem perfil misto para *cpu* e *disco*, possuem baixa degradação contra aplicações de perfil misto e de perfil voltado para *cpu*, e uma elevada perda de desempenho quando executadas contra aplicações de perfil apenas de disco. Aplicações com perfil de disco possuem baixa degradação apenas quando executadas contra aplicações de perfil *cpu*.

5 Conclusão

A primeira parte deste trabalho consistiu na implantação de um ambiente de computação em nuvem de modo que fosse possível um melhor aproveitamento dos recursos de hardware dos servidores físicos do *LAPPIS*, bem como a garantia de um ambiente estável para aplicação de estudo de interferência em ambientes virtualizados. Nesta etapa do trabalho, também foram feitas pesquisas bibliográficas referentes a interferência entre máquinas virtuais em um servidor físico, ao qual foram selecionados alguns trabalhos que poderiam servir de insumo para este estudo.

Na segunda parte deste trabalho teve como foco a aplicação dos experimentos feitos por [Koh et al. \(2007\)](#) no ambiente recém implantado, entretanto uma das dificuldades encontradas foi replicar os procedimentos feitos por [Koh et al. \(2007\)](#), dado que os mesmos não eram explicitamente descritos. Além disso, o *hypervisor* utilizado, o *KVM*, não possuía suporte para coleta de métricas de desempenho a nível de sistema. Desse modo a coleta de dados foi dividida em três ciclos de experimento, que permitiram que fosse obtida e avaliada de maneira gradativa os dados obtidos com relação a interferência ocasionada entre máquinas virtuais. Assim, umas das contribuições deste trabalho envolve a definição de procedimentos metodológicos, experimentais bem como no uso de ferramentas, que possibilitem a replicação deste experimento em trabalhos futuros, e em ambientes com *hypervisors* que não possuam suporte para coleta de dados referentes ao desempenho de máquinas virtuais.

Os resultados dos experimentos realizados permitem concluir que as tecnologias atuais de virtualização e de computação em nuvem, não garantem um isolamento efetivo entre máquinas virtuais em um mesmo servidor físico. A tendência é que aplicações que possuem perfil voltado para operações em disco interferem de maneira significativa no desempenho outras aplicações com o mesmo perfil, quando executadas em máquinas virtuais diferentes. Aplicações com o perfil voltado para *cpu* interferem muito pouco contra aplicações com o perfil voltado para aplicações em disco e vice-versa.

Um ponto que pode ser considerado uma limitação deste trabalho é que não foi possível avaliar a degradação de desempenho entre aplicações com o perfil voltado para utilização de *cpu*. Entende-se que para as configurações de *hardware* do servidor utilizado neste estudo, são necessárias um número maior de máquinas virtuais executando *benchmarking* de *cpu*, ao mesmo tempo, para que fosse possível observar algum tipo de interferência e degradação em seus desempenhos.

Os procedimentos adotados por esse trabalho promovem algumas possibilidades para trabalhos futuros. Dessa forma, um dos caminhos que podem ser adotados consiste

em um comparativo entre os diversos tipos de *hypervisors* utilizados, levando-se em conta a técnica de virtualização utilizada: paravirtualização e virtualização total, de modo que se possa avaliar qual tipo de ferramenta dado a, técnica de virtualização, trata melhor a interferência de desempenho entre as máquinas virtuais.

Durante o desenvolvimento deste trabalho foi publicado um artigo por [Caglar et al. \(2016\)](#) ao qual é apresentado o desenvolvimento de uma ferramenta responsável por mitigar a interferência no desempenho de aplicações, efetuando migrações de máquinas virtuais entre servidores dado o tipo de aplicação que está sendo executado nas mesmas. Para tal essa ferramenta coleta dados a partir de ferramentas de *benchmark* de modo que são traçados os perfis das aplicações. Assim, aplicando técnicas de *machine learning* em cima dos dados coletados essa ferramenta promove a migração de máquinas virtuais entre os servidores físicos dado o seu grau de degradação da aplicação. Para predição de desempenho dessas aplicações foi utilizada uma técnica de mineração de dados denominada *k-means*.

Já no trabalho de [Koh et al. \(2007\)](#), para predição de desempenho foi utilizada a média aritmética ponderada a partir de uma análise de componente principal efetuada em cima dos dados coletados. Dessa forma um possível trabalho futuro englobaria a aplicação de técnicas de *machine learning* utilizando os modelos preditivos aplicados por [Koh et al. \(2007\)](#) e comparando os com os resultados alcançados por [Caglar et al. \(2016\)](#) utilizando o *k-means*.

Referências

- CAGLAR, F. et al. Intelligent, performance interference-aware resource management for iot cloud backends. In: *IoTDI*. IEEE Computer Society, 2016. p. 95–105. ISBN 978-1-4673-9948-7. Disponível em: <<http://dblp.uni-trier.de/db/conf/iotdi/iotdi2016.html#CaglarSGK16>>. Citado na página 58.
- CARISSIMI, A. *Virtualização: da teoria a soluções*. 2016. <<http://www.gta.ufrj.br/ensino/CPE758/artigos-basicos/cap4-v2.pdf>>. Acesso em: 8 de Março de 2016. Citado na página 23.
- CARVALHO, F. L. de; BELLEZI, M. A. Avaliação de desempenho dos hypervisors xen e kvm utilizando administração simplificada através do libvirt. *Tecnologias, Infraestrutura e Software*, 2014. Citado 2 vezes nas páginas 29 e 30.
- CHEF SOFTWARE, INC. *An Overview of Chef*. 2016. <https://docs.chef.io/chef_overview.html>. Acesso em: 8 de Março de 2016. Nenhuma citação no texto.
- CLOUDSTACK. *About Cloudstack*. 2016. <<https://cloudstack.apache.org/about.html>>. Accessed: 2016-03-07. Citado 2 vezes nas páginas 31 e 32.
- HIGGINBOTTOM, G. *CloudStack 101 – Tudo O Que Você Precisa Em Apenas Um Artigo*. 2013. Acesso em: 7 de março de 2016. Disponível em: <<http://www.shapeblue.com/pt-br/cloudstack-101-2/>>. Citado na página 30.
- HUBER, N.; KONEV, S.; HAUCK, M. Evaluating and modeling virtualization performance overhead for cloud environments. 2011. Citado 5 vezes nas páginas 19, 20, 38, 39 e 40.
- ISMAEEL, S. et al. Open source cloud management platforms: A review. In: *IEEE 2nd International Conference on Cyber Security and Cloud Computing, CSCloud 2015, New York, NY, USA, November 3-5, 2015*. [S.l.: s.n.], 2015. p. 470–475. Citado 2 vezes nas páginas 34 e 35.
- JUNIOR, D. P. Q. *Virtualização: Conceitos, Técnicas aplicadas em um comparativo de desempenho entre as principais ferramentas sem custo de licenciamento*. Dissertação (Mestrado) — Instituto Superior Tupy, Joinville, 2008. Citado na página 25.
- KOH, Y. et al. An analysis of performance interference effects in virtual environments. *Performance Analysis of Systems and Software, 2007. ISPASS 2007. IEEE International Symposium on*, 2007. Citado 10 vezes nas páginas 19, 37, 39, 40, 42, 43, 44, 47, 57 e 58.
- LAPPIS. *Apresentação*. 2014. <<https://fga.unb.br/lappis/>>. Acesso em: 06 de Junho de 2016. Citado na página 21.
- LAUREANO, M. *Máquinas virtuais e emuladores*. [S.l.]: Novatec editora, 2006. Citado 2 vezes nas páginas 26 e 27.
- MCEWAN, W. Virtual machine technology and their application in the delivery of ict. In: *Accessed 14 March 2003 www.ddj.com/documents/s=882/ddj0008f/0008f.htm*. [S.l.: s.n.], 2002. p. 55–62. Citado na página 24.

- MENASCÉ, D. A. *Virtualization: Concepts, Applications, and Performance Modeling*. 2005. Citado na página 23.
- OPENNEBULA. *Administration Guide*. 2016. <<https://softwarepublico.gov.br/gitlab/softwarepublico/softwarepublico/wikis/manual-arquitetura>>. Acesso em: 12 de Março de 2016. Citado 4 vezes nas páginas 32, 33, 34 e 35.
- POPIOLEK, P. F.; MENDIZABAL, O. M. Monitoring and analysis of performance impact in virtualized environments. *Journal of Applied Computing Research*, 2012. Citado 5 vezes nas páginas 11, 19, 38, 39 e 40.
- PORTAL DO SOFTWARE PÚBLICO BRASILEIRO. *Manual de Operação*. 2016. <<https://softwarepublico.gov.br/gitlab/softwarepublico/softwarepublico/wikis/manual-arquitetura>>. Acesso em: 10 de Março de 2016. Citado na página 68.
- QUMRANET, INC. *KVM: Kernel-based Virtualization Driver. White Paper*. 2006. Citado na página 28.
- RASMUSSEN, L.; CORCORAN, D. Performance overhead of kvm on linux 3.9 on arm cortex-a15. *SIGBED Rev.*, ACM, New York, NY, USA, v. 11, n. 2, p. 32–38, set. 2014. ISSN 1551-3688. Disponível em: <<http://doi.acm.org/10.1145/2668138.2668143>>. Citado na página 29.
- RED HAT. *Manual de Operação*. 2015. <<https://www.redhat.com/pt-br/files/resources/en-rh-kvm-kernal-based-virtual-machine.pdf>>. Acesso em: 13 de Março de 2016. Citado 2 vezes nas páginas 28 e 29.
- SOUZA, F. B. de. *Uma arquitetura para monitoramento e medição de desempenho para ambientes virtuais*. Dissertação (Mestrado) — Universidade Federal de Minas Gerais, 2006. Citado na página 19.
- TANENBAUM, A. S. *Modern Operating Systems*. 3rd. ed. Upper Saddle River, NJ, USA: Prentice Hall Press, 2007. ISBN 9780136006633. Citado 3 vezes nas páginas 23, 27 e 28.
- TICKOO, O. et al. Modeling virtual machine performance: Challenges and approaches. In: *ACM SIGMETRICS Performance Evaluation Review*. [S.l.: s.n.]. Citado na página 20.
- VERAS, M.; CARISSIMI, A. *Virtualização de Servidores*. 3rd. ed. Rio de Janeiro: Escola Superior de Redes, 2015. Citado 4 vezes nas páginas 23, 24, 25 e 26.
- VMWARE INC. *Understanding Full Virtualization, Paravirtualization, and Hardware Assist*. 2008. <http://www.vmware.com/content/dam/digitalmarketing/vmware/en/pdf/techpaper/VMware_paravirtualization.pdf>. Acesso em: 13 de Maio de 2016. Citado 3 vezes nas páginas 26, 27 e 28.
- WALTERS, J. P. et al. A comparison of virtualization technologies for hpc. In: *Proceedings of the 22Nd International Conference on Advanced Information Networking and Applications*. Washington, DC, USA: IEEE Computer Society, 2008. (AINA '08), p. 861–868. ISBN 978-0-7695-3095-6. Disponível em: <<http://dx.doi.org/10.1109/AINA.2008.45>>. Citado na página 29.
- XEN PROJECT. *History*. 2016. <<http://www.xenproject.org/about/history.html>>. Acesso em: 15 de Março de 2016. Citado na página 29.

Apêndices

APÊNDICE A – Estudo de caso

A.1 Lappis

O LAPPIS possui uma infraestrutura de servidores que vinha possibilitando a disponibilização de serviços e ferramentas de utilidades para FGA. Destacam-se as ferramentas *Redmine* e *Dotproject*, utilizadas nas disciplinas de *Metodologias de Desenvolvimento de software* e *Gestão de Portifólio e Produto*, bem como possibilitava a disponibilização de máquinas virtuais utilizadas como ambientes de testes para desenvolvimento do Portal do Software Público, e para sistemas que estavam sendo desenvolvidos também pelo LAPPIS, tais como o SRA(sistema de registro de atendimento) e o SGD(sistema de gestão de desempenho). Entretanto, tais recursos físicos vinham sendo subutilizados, devido aos seguintes fatores:

- Versão do *hypervisor* desatualizada.
- Ausência de uma interface de gestão para máquinas virtuais.
- Centralização do conhecimento.

O *hypervisor* utilizado para disponibilização máquinas virtuais era o XEN na versão 4.1. Com o XEN nessa versão era inviável a disponibilização de máquinas virtuais com a versão de sistemas operacionais mais recentes tais como *Debian 7*, *Debian 8* e *Centos 7*. O que tornava difícil também, a tarefa de disponibilizar ambientes de testes com sistemas operacionais atualizados para sistemas em desenvolvimento pelo LAPPIS. A falta de uma interface de gerenciamento dificultava atividades triviais tais como instanciação, criação de imagens e migração de máquinas virtuais bem como visibilidade de uso de recursos. Por fim, a centralização do conhecimento impactava uma dependência problemática do profissional responsável pela implementação dessa infraestrutura. Assim, na sua ausência a equipe por parte do LAPPIS responsável por essa infraestrutura, encontrou sérias dificuldades em manter a disponibilização de ambientes virtuais. Essa baixa visibilidade dos procedimentos adotados na infraestrutura, também promovia insegurança por parte da equipe em arriscar no desenvolvimento de mudanças relacionadas à essa infraestrutura. Desse modo, o uso de recursos de *hardware* disponíveis para provimento de serviços úteis tanto para o LAPPIS quanto para a FGA estava comprometida.

A partir disso, dada a inviabilidade de continuar com essa infraestrutura, chegou-se a conclusão que o melhor caminho a ser adotado era a reformulação da mesma. Desse modo, adotou-se os seguintes procedimentos:

- Migração de máquinas virtuais para um dos servidores, de modo que o outro permanecesse liberado para a implementação inicial de uma plataforma em nuvem.
- Implementação de uma plataforma em nuvem que atendessem as necessidades do LAPPIS no servidor físico.
- Consolidação de toda infraestrutura física sob a solução de nuvem.

Desse modo, com os próprios colaboradores do LAPPIS desenvolvendo esse tipo de iniciativa, a expectativa era que o problema relacionado com a centralização do conhecimento fosse sanado. A implementação de uma solução de nuvem, e consequentemente, uso de outro *hypervisor* ou até mesmo o próprio XEN atualizado proporcionaria a solução dos problemas relacionados a falta de gerenciabilidade e a disponibilização de máquinas virtuais com sistemas operacionais atualizados, respectivamente.

A.2 Infraestrutura

A infraestrutura basicamente é composta de três servidores físicos e de máquinas virtuais que compartilham o uso de recursos desses servidores. Dois desses possuem a mesmas configurações:

- Servidor em *rack Dell PowerEdge r620*.
- 24 processadores *Intel Xeon* , 2.0GHz.
- 64 GB de Memória DDR3.
- 2 TB SATA HDD.
- 4 interfaces *EThernet 10/100/1000-BaseT*.

Esses servidores são identificados como *Solarian* e *Imperius* e estavam sendo utilizados como os provedores de máquinas virtuais. O terceiro servidor físico possui a seguinte configuração:

- Servidor *Dell T5500*
- 8 processadores *Intel Xeon* , 3.2GHz.
- 24 GB de Memória DDR3.
- 2TB SATA HDD.
- 1 interface *EThernet 10/100/1000-BaseT*.

Esse servidor é identificado como *Polaris* e seu uso se limitava a um ambiente de testes para ferramentas de integração contínua do Portal do Software Público. A Figura 25 apresenta um esquema onde mostra a alocação das máquinas virtuais nos servidores, em seguida é feita uma breve descrição dos serviços oferecidos.

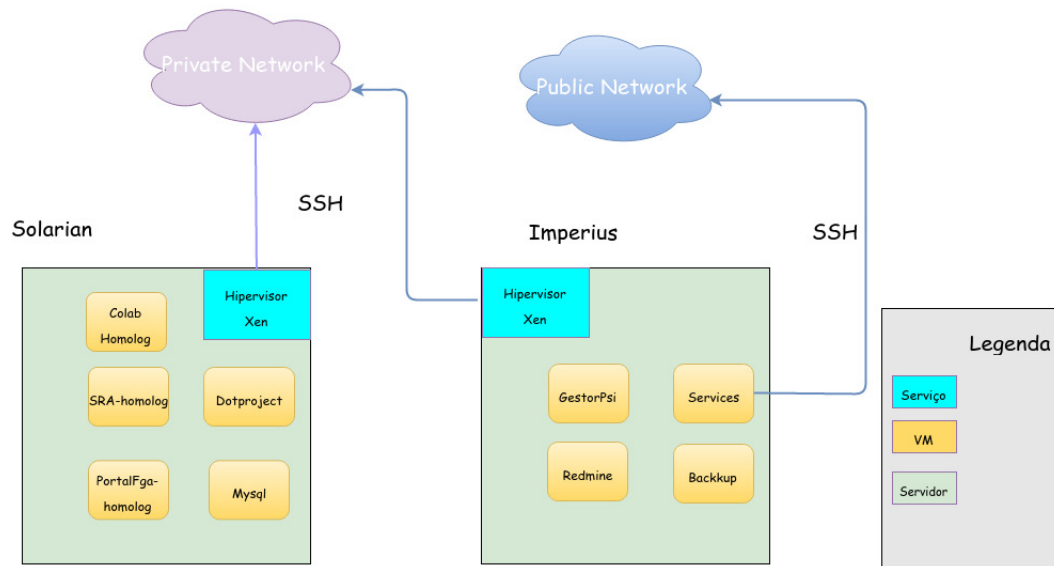


Figura 25: Visão geral da Infraestrutura

- **Services:** Ambiente utilizado para acesso externo, tinha função de servidor *DHCP* também.
- **Backup:** Ambiente utilizado para backups do *Redmine*, *Dotproject* e Portal da FGA.
- **MySQL:** Utilizada como banco de dados para o *Redmine* e *Dotproject*.
- **SRA-Homolog:** Ambiente utilizada para homologação do Sistema de Registro de Atendimentos.
- **SGD-Homolog:** Ambiente utilizada como homologação do Sistema de Gestão de Desempenho.
- **Dotproject:** Utilizada para provimento do *Dotproject*, empregado como ferramenta de Gestão nas disciplinas de Gestão de Portifólio e Produto e Metodologia de Desenvolvimento de Software.
- **Redmine:** Utilizado para disponibilização do *Redmine*, empregado também como ferramentas de Gestão nas disciplinas de Gestão de Portifólio e Produto e Metodologia de Desenvolvimento de Software.

- **PortalFga-homolog**: Ambiente de homologação para o Portal da FGA.
- **GestorPsi**: Ambiente de testes para o *GestorPsi*, software da disciplina Manutenção Evolução do Software.
- **Colab-homolog**: Ambiente de testes para a ferramenta *Colab*, software que compõe o Portal do Software Público.

A.3 Migração de Máquinas Virtuais

Para implementação de uma solução de plataforma em nuvem, os procedimentos adotados consistiam em usar inicialmente um dos servidores como ambiente físico para testes iniciais da plataforma em nuvem. Assim que a mesma estivesse estabilizada, o outro servidor seria agregado a essa solução. Desse modo, uma maneira encontrada para que esses procedimentos fossem feitos sem ter a indisponibilização dos serviços por um grande período de tempo foi a migração de todos os serviços para um único servidor. Assim, teria-se um servidor livre para uma implementação e investigação inicial dessa plataforma em nuvem, enquanto que o outro servidor estaria disponibilizando os serviços em uso. Com isso, o servidor escolhido para essa implementação e investigação inicial da plataforma em nuvem foi o *Solarian*.

Dado que as máquinas virtuais utilizavam discos *LVM*, o procedimento adotado foi:

- Criação de imagens das máquinas virtuais com auxílio *LVM* e do comando *dd*.
- Transferência dessas imagens para o servidor *Imperius*.
- Restauração dessas imagens em discos *LVM* no servidor *Imperius*.
- Criação de máquinas virtuais no servidor *Imperius* utilizando as imagens restauradas em discos *LVM*.

Assim, todas as máquinas virtuais que estavam no servidor *Solarian* foram transferidas para o servidor *Imperius*, possibilitando assim que fosse iniciada a implementação da plataforma em nuvem em um servidor físico.

A.4 Implementação da Plataforma em nuvem

Para implementação da plataforma em nuvem duas ferramentas foram previamente abordadas *Cloudstack* e *Opennebula*.

Dessa maneira, com o intuito de possuir ambiente limpo e com um sistema operacional atualizado, efetuou-se a formatação do disco do servidor *Solarian*, sendo em seguida instalado o sistema operacional *Centos 7*. Assim, com o auxílio da documentação do *cloudstack* a instalação procedeu sem muitos problemas. Com instalação concluída, já era possível acessar a interface de gerenciamento dando continuidade com as configurações necessárias para criação de máquinas virtuais. Entretanto, a obrigatoriedade de configuração de todos os níveis de abstração, apresentados na Subseção 2.6.1, mostrou-se dispendiosa e desnecessária para uma configuração mínima e também para a infraestrutura disponível no LAPPIS.

Desse modo, alguns elementos do *cloudstack*, necessários para o ambiente, não apresentaram o comportamento esperado. Um exemplo disso, eram problemas recorrentes relacionados com a máquina virtual de sistema, responsável pelas operações no *storage* secundário (referenciada por *SSVM*), o que impossibilitava a criação de máquinas virtuais a partir de templates. Em resumo, mesmo com os problemas enfrentados, foi possível a criação de máquinas virtuais nessa plataforma, entretanto a configuração de vários elementos decorrente de sua abstração voltada para uma infraestrutura mais complexa, acabou por tornar difícil o gerenciamento e estabilização do ambiente como um todo, sendo considerado, para o caso em específico do LAPPIS, não sustentável.

Para instalação do *OpenNebula*, foram adotados os mesmos procedimentos feitos para testes com o *Cloudstack*: formatação do disco e instalação do sistema operacional *Centos 7*. Dessa forma, destaca-se a clareza e objetividade da documentação do *OpenNebula*, ao qual, a partir da mesma, a instalação prosseguiu sem muitas dificuldades. Para testes iniciais, utilizou-se uma estação convencional de trabalho para ser usada como *front-end* e *datastore*. O servidor *Solarian*, foi então configurado como *worked-node*. Após a instalação e as configurações de redes devidamente concluídas no *front-end* e *worked-node*, já era possível acessar a interface de gerenciamento. Assim, através da interface de gerenciamento, adicionou-se o servidor *Solarian* como *worked-node*.

A partir de então, o *daemon ONED* do *OpenNebula* passou a realizar operações de reconhecimento e monitoramento no servidor *Solarian*, sendo possível a criação de máquinas virtuais sobre o mesmo. Não se teve qualquer tipo de problema ou instabilidade graves na criação de máquinas virtuais, tão logo criadas, já era possível acessá-las via *ssh*. Destaca-se que a integração da interface de gerenciamento com o *OpenNebula Marketplace*, possibilitou a disponibilização rápida de máquinas virtuais. O *OpenNebula Marketplace* é um catálogo *online* de imagens pré-configuradas para máquinas virtuais, assim através da sua integração com a interface de gerenciamento já era possível criar máquinas virtuais com sistemas operacionais como *Debian 8* e *Centos 7*, previamente disponibilizados no *OpenNebula Marketplace*.

Em seguida, a fim de se ter uma avaliação inicial de desempenho, tomou-se a deci-

são de disponibilizar um ambiente de testes para o desenvolvimento do Portal do Software Público (SPB). O SPB é composto de um conjunto de ferramentas com funcionalidades complementares, que são desenvolvidas de forma independentes pelas suas respectivas comunidades (PORTAL DO SOFTWARE PÚBLICO BRASILEIRO, 2016). Para isso, como apresentado na Figura 26 fazem uso de cinco máquinas virtuais, aos quais cada uma possui uma função ou serviço específico. A implantação de todo esse ambiente é automatizado, assim com poucos comandos os serviços necessários são instalados em suas respectivas máquinas virtuais. A Figura 26 apresenta uma arquitetura de implantação do ambiente do Portal do Software Público.

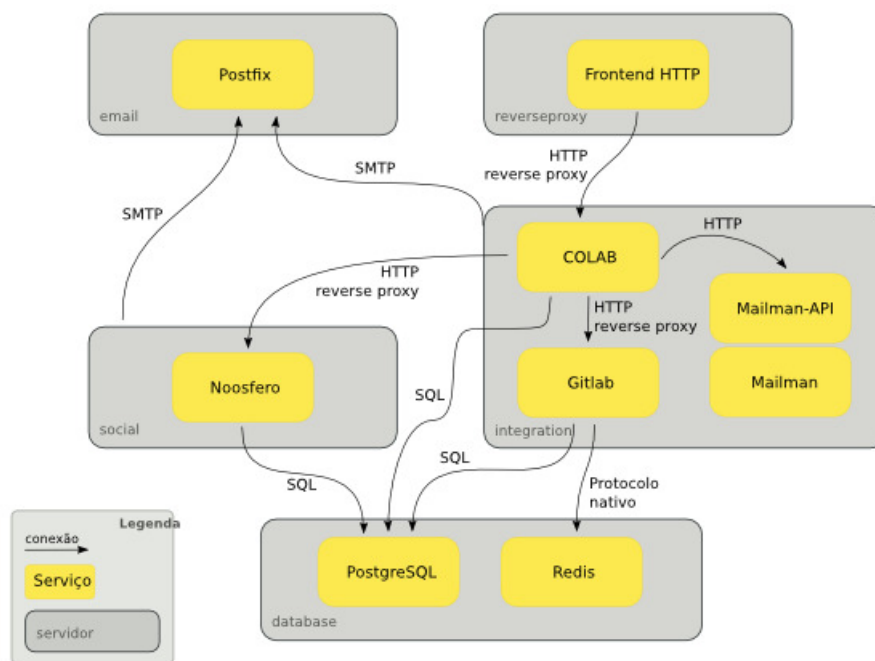


Figura 26: Arquitetura do Portal do Software Público
(PORTAL DO SOFTWARE PÚBLICO BRASILEIRO, 2016).

Assim, devido a quantidade ferramentas que seriam instaladas, só o processo de instalação já seria um ótimo meio de avaliar o desempenho. Portanto, cinco máquinas virtuais foram criadas com o sistema operacional *Centos 7* (necessário para a implantação do SPB), e em seguida, iniciou-se o procedimento de instalação do Portal do software Público nessas máquinas virtuais. Percebeu-se então que havia uma certa lentidão na instalação das ferramentas, e o procedimento que, em condições normais demoraria de 20 a 30 minutos, sob o ambiente virtual recém instanciado havia demorado por volta de 5 horas. Desse modo, alguns possíveis fatores foram listados como possíveis causas dessa lentidão:

- Uso de uma estação de trabalho de baixo desempenho como *datastore* e *front-end*.
- Uso do *System Datastore* na opção *shared*.

- Imagem de disco no formato *qcow2*

A estação de trabalho convencional havia sido escolhida apenas para testes iniciais, desse modo, por possuir uma configuração intermediária entre um servidor e estação de trabalho de alto desempenho, optou-se por utilizar a máquina *Polaris* como *datastore*, *front-end* e também como *worked-node*. Após isso, notou-se uma melhora significativa no tempo de disponibilização de máquinas virtuais mas ainda assim, continuava baixo o desempenho para a implantação do Portal do Software Público. Então a próxima medida adotada foi a mudança de configuração no tipo de *System Datastore*.

Inicialmente, havia se utilizado como configuração padrão do OpenNebula a opção *shared* para tipo do *System Datastore*. O que se mostrou um equívoco inicial, afinal o *LAPPIS* não possui uma infraestrutura de rede de armazenamento que comporte esse tipo de centralização de armazenamento para máquinas virtuais. Com as operações de escrita em disco das máquinas virtuais tendo que passar através da rede, isso viria a se tornar um grande gargalo. Portanto, a opção adequada nesse caso foi a configuração do *System Datastore* para opção *ssh*. Teria-se um aumento no tempo na disponibilização de máquinas, entretanto o desempenho melhoraria pois o uso do disco estava local agora.

Outro fator que impactava no desempenho era o uso de imagens *qcow2*. O *qcow2* é um formato de imagem de disco, ao qual sua vantagem reside no fácil gerenciamento. Afinal, possui suporte para *snapshot*, alocação de espaço no disco é dinâmica e o tamanho de suas imagens é bastante reduzido se comparado com o formato *raw*. Entretanto, possui um baixo desempenho, se comparado ao próprio *raw*. Uma alternativa inicial foi o uso de discos *LVM* para as máquinas virtuais ao invés de imagens de disco como *raw* e *qcow2*, entretanto teve-se problemas com o monitoramento efetuado pelo *ONED*, o que por si só foi um indicativo que para o uso de *LVM* o *OpenNebula* ainda não oferece um bom suporte. Então, após investigações sobre o problema de desempenho com imagens de disco no formato *qcow2*, descobriu-se que os problemas de desempenho com essas imagens podem ser minimizados utilizando opções de *cache* em disco. Dentre as opções disponibilizadas pelo *OpenNebula* (*writeback*, *writethrough*, *directsync*, *unsafe*) a que mostrou mais efetiva em termos de desempenho e segurança foi a opção *writeback*.

Após esses procedimentos, notou-se um aumento considerável no desempenho de escrita de disco nas máquinas virtuais, de modo que as aplicações passaram a serem instaladas sem qualquer problema relacionado com desempenho, possibilitando assim que a implantação Portal do Software Público para ambiente de testes pudesse ser feito sem grandes problemas. Com isso, finalmente havia-se alcançado um ponto de estabilização da infraestrutura como um todo, os próximos passos englobariam então a disponibilização dos serviços providos pelo *LAPPIS* sob a nova plataforma em nuvem, bem como utilização do servidor *Imperius* sob essa plataforma.

A.5 Consolidação da Infraestrutura

Após os testes iniciais e a disponibilização de alguns serviços, tinha-se uma infraestrutura estável e pronta para ser utilizada como provedora de serviços. Como etapa final, então era necessário incorporar o servidor *Imperius* sob a plataforma em nuvem. Antes disso, fazia-se necessário replicar os serviços, que atualmente estavam em execução no servidor *Imperius*. Com isso feito, o servidor *imperius* fora incorporado sob a nova plataforma consolidando assim toda a infraestrutura física de servidores sob a nova plataforma em nuvem. A Figura 27 apresenta o modelo de implantação final do *OpenNebula*.

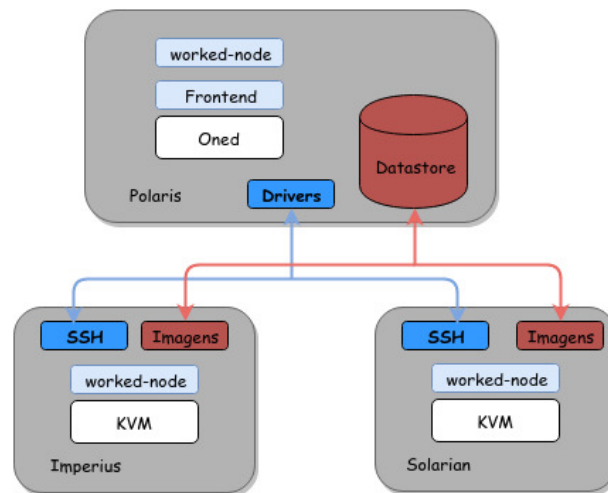


Figura 27: Implantação do *OpenNebula*

Serviços como *Redmine*, *Dotproject* se mantiveram, a *Services* que antes era utilizado tanto como *Firewall* como máquina para acesso externo, fora configurada apenas como máquina de acesso externo. Uma outra máquina virtual para *Firewall* fora criada, sendo nela colocados os IPs públicos. Outros serviços foram disponibilizados, a Figura 28 apresenta um visão geral de todos eles. No total, hoje são disponibilizadas 26 máquinas virtuais. A seguir é feita uma breve descrição dos principais serviços recém disponibilizados:

- **Proxy:** Utilizada como um *proxy* de cache, promovendo assim um melhor desempenho na transferência de pacotes que já foram baixados.
- **Boca:** Máquina virtual utilizada para disponibilização do *BOCA*, servindo dessa forma como *juiz online* para competições de programação para disciplina de *Tópicos Especiais em Programação*.
- **Moodle:** Responsável pela disponibilização do *Moodle* para disciplinas de Computação Básica.

- **Mirror:** Responsável por servir de *espelho* de alguns repositórios utilizados pelo Portal do Software Público.
- **PortalFGA-Homologação:** Ambiente de homologação do Portal da FGA.

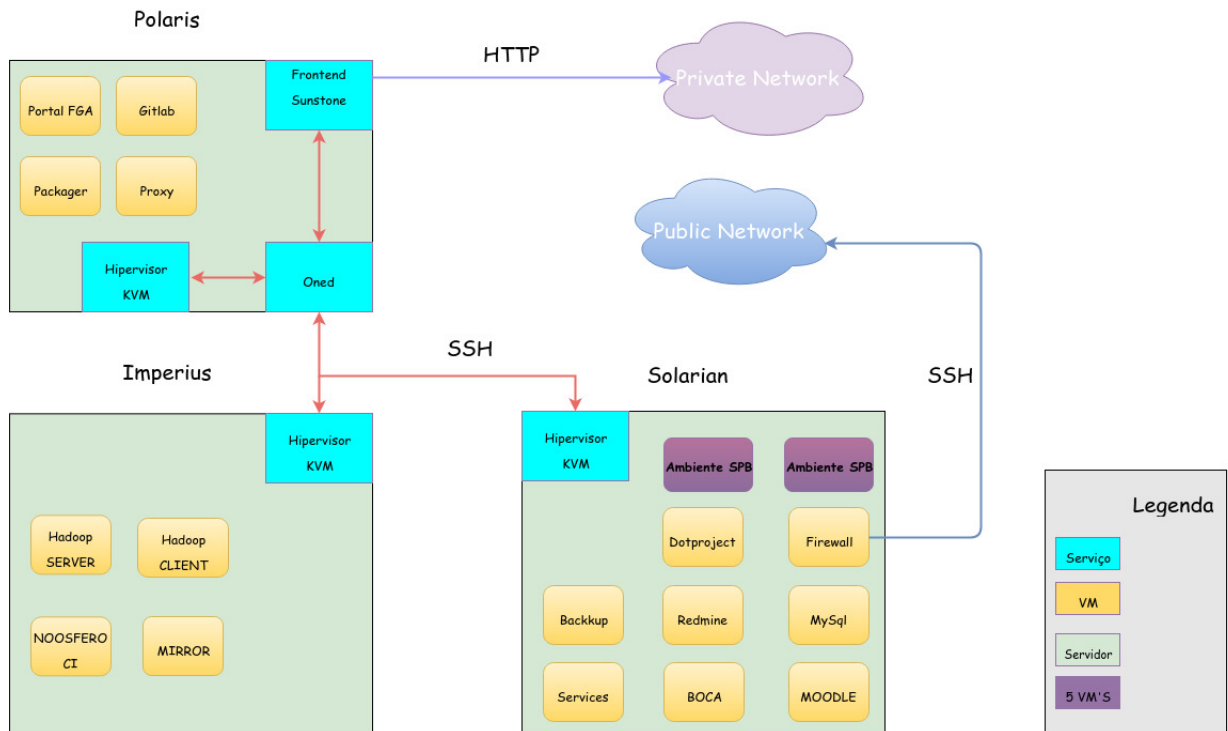


Figura 28: Visão geral da Infraestrutura após implantação do *OpenNebula*

Outro procedimento adotado foi o uso da ferramenta *Chef* em conjunto com o *Chake* para automatização, tanto no provimento da plataforma em nuvem, quanto na disponibilização de serviços. Com isso, como uma iniciativa inicial para automatização bem como documentação da infraestrutura, desenvolveu-se neste trabalho arquivos de configurações, utilizando o *Chake* e *chef solo*, para alguns serviços disponibilizados: *Proxy*, *Redmine*, *OpenNebula Front end*, *OpenNebula Worked Node*. Na seção de Apêndices são disponibilizados os códigos das receitas desenvolvidas para provimento e documentação desse serviços.

APÊNDICE B – Código fonte da receita do *OpenNebula Frontend*

```
1 repo_dir = '/etc/yum.repos.d/opennebula.repo'
2 config_ssh_dir = '/var/lib/one/.ssh/config'
3 package "epel-release"
4
5 template repo_dir do
6   source 'opennebula.repo.erb'
7 end
8
9 package 'opennebula-server'
10 package 'opennebula-sunstone'
11
12 execute 'disable-selinux' do
13   command 'sed -i "s/SELINUX=\w*/SELINUX=disable /g"
14             etc/sysconfig/selinux'
15 end
16
17 execute 'change_script' do
18   command 'sed -i "s/yum install/yum install -y/g"
19             /usr/share/one/install_gems'
20 end
21
22 execute 'sunstone' do
23   command 'echo -e 1 "\n" "\n| /usr/share/one/install_gems'
24 end
25
26 execute 'external_access' do
27   command 'sed -i "s/:host: 127.0.0.1/:host: 0.0.0.0/g"
28             /etc/one/sunstone-server.conf'
29 end
30
31 service 'opennebula' do
32   action [:enable, :start]
```

```
33  action [:enable, :start]
34 end
35
36 template config_ssh_dir do
37   source 'config.erb'
38   owner 'oneadmin'
39   group 'oneadmin'
40   mode '0600'
41 end
```

APÊNDICE C – Código fonte da receita do *OpenNebula Node*

```
1 repo_dir= '/etc/yum.repos.d/opennebula.repo'
2
3 #add opennebula repository
4 template repo_dir do
5   source 'opennebula.repo.erb'
6 end
7 #install node package for OpenNebula
8 package "opennebula-node-kvm"
9
10 #enable and starts services
11 service "messagebus.service" do
12   action [:enable, :start]
13 end
14
15 service "libvirtd.service" do
16   action [:enable, :start]
17 end
18
19 service "nfs-server.service" do
20   action [:enable, :start]
21 end
```


APÊNDICE D – Código fonte da receita do *Redmine*

```
1 version = '3.2.0'
2 url_redmine =
    "https://www.redmine.org/releases/redmine-#{version}.tar.gz"
3 redmine_dir = '/opt/redmine/'
4 apache_conf_dir = '/etc/apache2/sites-available/master.conf'
5 #install dependencies
6 execute 'update' do
7   command "apt-get update"
8   ignore_failure true
9   action :nothing
10 end
11 packages = %w(mysql-client libmysqlclient-dev gcc
    build-essential zlib1g zlib1g-dev zlibc ruby-zip libssl-dev
    libyaml-dev libcurl4-openssl-dev ruby gem
    libapache2-mod-passenger apache2-mpm-prefork apache2-dev
    libapr1-dev libxslt1-dev checkinstall libxml2-dev ruby-dev vim
    libmagickwand-dev imagemagick)
12
13
14 packages.each do |package_name|
15   package package_name
16 end
17
18 directory redmine_dir
19
20 remote_file redmine_dir + 'redmine.tar.gz' do
21   source url_redmine
22   mode '0755'
23 end
24
25 execute 'extract_redmine' do
26   command 'tar xzf redmine.tar.gz'
27   cwd redmine_dir
28 end
29
```

```
30 #installing gem bundler
31 gem_package 'bundler'
32
33 extracted_redmine_dir = redmine_dir+'redmine-'+version
34
35 execute 'bundle_install' do
36   command 'bundle install'
37   cwd extracted_redmine_dir
38 end
39
40 execute 'generate_redmine_secret_token' do
41   command 'bundle exec rake generate_secret_token'
42   cwd extracted_redmine_dir
43 end
44
45 template extracted_redmine_dir+'/'+'config/database.yml' do
46   source 'database.yml.erb'
47   variables({
48     redmine_passwd: node['passwd']['redmine']
49   })
50 end
51
52 execute 'database_migration' do
53   command 'RAILS_ENV=production bundle exec rake db:migrate ||
54           RAILS_ENV=production bundle exec rake
55           redmine:load_default_data'
56   cwd extracted_redmine_dir
57 end
58
59 execute "chown-data-www" do
60   command "sudo chown -R www-data files log tmp
61           public/plugin_assets"
62   user "root"
63   cwd extracted_redmine_dir
64 end
65
66 execute "symbolic_link" do
67   command 'sudo ln -s '+extracted_redmine_dir+'/public/'+
68           '/var/www/html/redmine'
69 end
70
71 template apache_conf_dir do
```

```
68   source 'master.conf.erb'
69 end
70
71 execute 'disable_default_apache' do
72   command 'sudo a2dissite 000-default.conf'
73 end
74
75 execute 'enable_master_conf' do
76   command 'sudo a2ensite master.conf'
77 end
78
79 execute 'passenger_permission' do
80   command "echo 'PassengerUser www-data' >>
            /etc/apache2/mods-available/passenger.conf"
81 end
82
83 execute 'enable_passenger' do
84   command "sudo a2enmod passenger"
85
86 end
87 service 'apache2' do
88   action :restart
89 end
```


APÊNDICE E – Código fonte da receita do servidor *Proxy* usando *squid*

```
1 # Recipe for squid server
2
3 # Config:
4 squid_maximum_object_size = "600" #MB
5 squid_minimum_object_size = "0" #MB
6 squid_cache_size = "30000" #MB
7
8 # Yum update
9 execute "yum_update" do
10   command "yum -y update"
11 end
12
13 # Install Squid
14 package 'squid'
15
16 # Configure squid.conf
17 squid_conf_file = "/etc/squid/squid.conf"
18 template squid_conf_file do
19   source 'squid.conf.erb'
20   variables({
21     cache_size: squid_cache_size,
22     maximum_object_size: squid_maximum_object_size,
23     minimum_object_size: squid_minimum_object_size
24   })
25 end
26
27 # Restart squid service
28 service "squid" do
29   action :restart
30 end
31
32 # Enable iptables to listen port 3128
33 execute "add_squid_iptables_rule" do
34   command "iptables -I INPUT -p tcp --dport 3128 -j ACCEPT && " +
35     "iptables-save > /etc/sysconfig/iptables"
```

```
36 end
37
38 # Install iptables service
39 package 'iptables-services'
40
41 # Enable iptables on boot and start it
42 service "iptables" do
43   action [:enable, :start]
44 end
```