UNIVERSIDADE DE CAXIAS DO SUL
CENTRO DE CIÊNCIAS EXATAS E DA TECNOLOGIA
CIÊNCIA DA COMPUTAÇÃO

BRUNO EMER

Implementação de alta disponibilidade em uma empresa prestadora de serviços para Internet

André Luis Martinotto
Orientador

Caxias do Sul
Junho de 2016

Implementação de alta disponibilidade em uma empresa prestadora de serviços para Internet

por

Bruno Emer

Projeto de Diplomação submetido ao curso de Bacharelado em Ciência da Computação do Centro de Ciências Exatas e da Tecnologia da Universidade de Caxias do Sul, como requisito obrigatório para graduação.

Projeto de Diplomação

Orientador: André Luis Martinotto
Banca examinadora:

Maria de Fatima Webber do Prado Lima
CCTI/UCS
Ricardo Vargas Dorneles
CCTI/UCS

Projeto de Diplomação apresentado em
x de julho de 2016

Daniel Luís Notari
Coordenador

SUMÁRIO

LISTA DE SIGLAS

ADSL	*Asymmetric Digital Subscriber Line*
API	*Application programming interface*
ASP	*Active Server Pages*
CRM	*Cluster resource management*
DNS	*Domain name system*
DRBD	*Distributed replicated block device*
ECC	*Error correction code*
FTP	*File transfer protocol*
IIS	*Internet information services*
IMAP	*Internet message access protocol*
IP	*Internet protocol*
IPv6	*Internet protocol version 6*
ISA	*Instruction set architecture*
JVM	*Java virtual machine*
KVM	*Kernel-based Virtual Machine*
LTS	*Long term support*
MAC	*Media access control*
MTBF	*Mean time between failures*
MTTR	*Mean time to repair*
NAT	*Network address translation*
PC	*Personal computer*
PHP	*Personal Home Page*
POP	*Post office protocol*
PPPoE	*Point-to-point protocol over ethernet*
PRTG	*Paessler router traffic grapher*
RAID	*Redundant array of independent disks*
RAM	*Random access memory*
SLA	*Service level agreement*
SMTP	*Simple mail transfer protocol*
SNMP	*Simple Network Management Protocol*
SPOF	*Single point of failure*
SSH	*Secure shell*
SVN	*Subversion*
TCP	*Transmission control protocol*
TI	Tecnologia da Informação

UDP	*User datagram protocol*
VM	*Virtual machine*
VMM	*Virtual machine monitor*
WHM	*WebHost manager*
XMPP	*Extensible messaging and presence protocol*

LISTA DE FIGURAS

LISTA DE TABELAS

RESUMO

O número de serviços oferecidos através da Internet vem crescendo a cada dia que passa. Sendo assim, a alta disponibilidade é um fator crucial para empresas que prestam serviços desse tipo. O aumento da disponibilidade é um grande diferencial para essas empresas, pois atualmente existe uma grande concorrência.

Além do estudo da disponibilidade apresentado neste trabalho, será feito um estudo sobre virtualização, pois é um dos principais recursos utilizados para se obter alta disponibilidade, sendo bastante relevante para empresas de hospedagens.

O principal objetivo deste trabalho é aumentar a disponibilidade de uma empresa prestadora de serviços para Internet, utilizando ferramentas de código aberto, de forma a garantir que seus serviços mais críticos fiquem o menor tempo indisponível. Para isso, será feito uma análise dos serviços ofereciados pela empresa bem como um levantamento do ambiente de servidores da mesma.

Além disso será pesquisado e testado ferramentas para possibilitar a criação de uma proposta e de uma implementação de alta disponibilidade no ambiente atual da empresa.

Palavras-chave: Alta disponibilidade, Virtualização, Tolerância a falhas, Código aberto.

Title

ABSTRACT

Keywords: keywords.

1 INTRODUÇÃO

O crescente avanço tecnológico e o desenvolvimento da Internet, provocou um aumento no número de aplicações ou serviços que dependem da infraestrutura de Tecnologia da Informação (TI). Além disso, percebe-se um aumento significativo no número de operações *on-line* que são realizados, tanto por organizações públicas ou privadas, quanto por grande parte da população.

Desta forma, a sociedade está cada vez mais dependente da tecnologia, de computadores e de sistemas. De fato, pode-se observar sistemas computacionais desde em uma farmácia, até em uma grande indústria. Sendo assim, a estabilidade e a disponibilidade desses sistemas apresenta um grande impacto em nosso dia-a-dia, pois um grande número de atividades cotidianas dependem deles.

Uma interrupção imprevista em um ambiente computacional poderá causar um prejuízo financeiro para a empresa que fornece o serviço, além de interferir na vida das pessoas que dependem de forma direta ou indireta deste serviço. Essa interrupção terá maior relevância para as corporações cujo o serviço ou produto final é fornecido através da Internet, como por exemplo, o comércio eletrônico, *web sites*, sistemas corporativos, entre outros. Em um ambiente extremo, pode-se imaginar o caos e o possível risco de perda de vidas que ocorreria em caso de uma falha em um sistema de controle aéreo (COSTA, 2009).

Para essas empresas um plano de contingência é fundamental para garantir uma boa qualidade de serviço, além de otimizar o desempenho das atividades, e também para fazer uma prevenção de falhas e uma recuperação rápida caso essas ocorram (COSTA, 2009). De fato, hoje em dia a confiança em um serviço é um grande diferencial para a empresa fornecedora deste, sendo que a alta disponibilidade é fundamental para atingir este objetivo.

A alta disponibilidade consiste em manter um sistema disponível por meio da tolerância a falhas, isto é, utilizando mecanismos que fazem a detecção, mascaramento e a recuperação de falhas, sendo que esses mecanismos podem ser implementados a nível de *software* ou de *hardware* (REIS, 2009). Para que um sistema seja altamente disponível ele deve ser tolerante a falhas, sendo que a tolerância a falhas é frequentemente implementada utilizando redundância. No caso de uma falha em um dos componentes evita-se a interrupção do sistema, uma vez que o sistema poderá continuar funcionando utilizando o outro componente (BATISTA, 2007).

Neste trabalho será realizado um estudo sobre a implementação de um sistema de alta disponibilidade em uma empresa de hospedagens. Essa empresa oferece serviços pela Internet, como por exemplo hospedagens de sites, *e-mail*, sistemas de gestão, *e-mail marketing*, entre outros. A empresa possui aproximadamente 60 servidores físicos e virtuais, e aproximadamente 9000 clientes, sendo que em períodos de pico

atende em torno de 1000 requisições por segundo.

Atualmente, a empresa possui redundância de conexões de acesso à Internet, refrigeração e energia, com *nobreaks* e geradores. Porém, essa empresa não possui nenhuma redundância nos serviços que estão sendo executados nos servidores. Desta forma, caso ocorra uma falha de *software* ou de *hardware*, os serviços ficarão indisponíveis. Neste trabalho será realizada uma análise dos serviços oferecidos pela empresa, sendo que mecanismos de alta disponibilidade serão desenvolvidos para os serviços mais críticos. Para a redução dos custos serão utilizadas ferramentas gratuitas e de código aberto.

1.1 Objetivos

Atualmente a empresa estudada não possui nenhuma solução de alta disponibilidade para seus serviços críticos. Desta forma, neste trabalho será desenvolvida uma solução de alta disponibilidade para estes serviços, sendo que essa solução será baseada no uso de ferramentas de código aberto e de baixo custo. Para que o objetivo geral seja atendido os seguintes objetivos específicos deverão ser realizados:

- Identificar os serviços críticos a serem integrados ao ambiente de alta disponibilidade;
- Definir as ferramentas a serem utilizadas para implementar tolerância a falhas;
- Realizar testes para a validação do sistema de alta disponibilidade que foi desenvolvido.

1.2 Estrutura do trabalho

O trabalho foi estruturado em cinco capítulos, que são:

- Capítulo 2: apresenta o conceito de alta disponibilidade e conceitos relacionados;
- Capítulo 3: é apresentado um breve histórico da virtualização, bem como o conceito de máquinas virtuais e também as suas classificações e estratégias de implementação;
- Capítulo 4: descreve o ambiente atual da empresa com os serviços que são fornecidos;
- Capítulo 5: é feito a identificação dos serviços críticos e apresenta a solução de alta disponibilidade utilizando virtualização no cenário da empresa;
- Capítulo 6: apresenta a conclusão e o cronograma para a implementação da solução proposta.

2 ALTA DISPONIBILIDADE

Alta disponibilidade é uma conhecida técnica que está sendo cada vez mais empregada em ambientes computacionais. O objetivo de prover alta disponibilidade resume-se em garantir que um serviço esteja sempre disponível quando o cliente solicitar ou acessar (COSTA, 2009). A alta disponibilidade geralmente é implementada com uma redundância de *hardware* ou de *software*, sendo que quanto maior for a disponibilidade desejada maior deverá ser a redundância no ambiente, assim reduzindo os pontos únicos de falha, que em inglês são chamados de *Single point of failure* (SPOF). A alta disponibilidade está diretamente relacionada aos conceitos de:

- Dependabilidade: indica a qualidade do serviço fornecido e a confiança depositada neste serviço. A dependabilidade envolve atributos como segurança de funcionamento, segurança de acesso, manutenabilidade, testabilidade e comprometimento com o desempenho (WEBER, 2002);
- Confiabilidade: é o atributo mais importante, pois transmite a ideia de continuidade de serviço (PANKAJ, 1994). A confiabilidade refere-se a probabilidade de um serviço estar funcionando corretamente durante um dado intervalo de tempo;
- Disponibilidade: é a probabilidade de um serviço estar operacional no instante em que for solicitado (COSTA, 2009);
- Tolerância a falhas: procura garantir a disponibilidade de um serviço utilizando mecanismos capazes de detectar, mascarar e recuperar falhas. O seu principal objetivo é alcançar a dependabilidade, assim indicando uma boa qualidade de serviço (COSTA, 2009). A tolerância a falhas é um dos principais conceitos da alta disponibilidade, sendo descrita na Seção 2.1.

2.1 Tolerância a falhas

Sabe-se que o *hardware* tende a falhar, principalmente devido a fatores físicos, por isso utiliza-se métodos para a prevenção de falhas. A abordagem de prevenção de falhas é realizada na etapa de projeto, ou seja, consiste em definir mecanismos que impeçam que que as falhas ocorram. Além disso, a prevenção de falhas melhora a disponibilidade e a confiabilidade de um serviço, uma vez que esta tem como objetivo diminuir a possibilidade de falhas antes de colocar o sistema em uso.

A prevenção de falhas não eliminará todas as possíveis falhas. Sendo assim, a tolerância a falhas procura fornecer a disponibilidade de um serviço mesmo com a presença de falhas. De fato, enquanto a prevenção de falhas tem foco nas fases de

projeto, teste e validação, a tolerância a falhas apresenta como foco a utilização de componentes replicados para mascarar as falhas (PANKAJ, 1994).

O objetivo da tolerância a falhas é aumentar a disponibilidade de um sistema, ou seja, aumentar o intervalo de tempo em que os serviços fornecidos estão disponíveis aos usuários. Um sistema é dito tolerante a falhas se ele for capaz de mascarar a presença de falhas ou recuperar-se de uma sem afetar o funcionamento do sistema.

A tolerância a falhas frequentemente é implementada utilizando redundância (Seção 2.2). Um exemplo muito utilizado para tornar um sistema tolerante a falhas é a virtualização. Nestes ambientes normalmente existem dois servidores físicos onde máquinas virtuais são executadas, sendo que no caso de um dos servidores falhar, o *software* de monitoramento fará a transferência das máquinas virtuais para o outro servidor, de forma transparente aos usuários, evitando assim a indisponibilidade do serviço. Os principais conceitos de virtualização, são apresentados no Capítulo 3.

A tolerância a falhas pode ser dividida em dois tipos. O primeiro tipo, o mascaramento, não se manifesta na forma de erro ao sistema, pois as falhas são tratadas na origem. O mascaramento é utilizado principalmente em sistemas críticos e de tempo real. Um exemplo são os códigos de correção de erros, em inglês *Error correction code* (ECC), que são utilizados em memórias para detecção e correção de erros.

O segundo tipo de tolerância a falhas consiste em detectar, localizar a falha, e reconfigurar o *software* ou *hardware* de forma a corrigi-la. Esse tipo de tolerância a falha é dividido nas seguintes etapas (WEBER, 2002).

- Detecção: realiza o monitoramento e aguarda uma falha se manifestar em forma de erro, para então passar para a próxima fase. Um exemplo de detecção de erro é um cão de guarda (*watchdog timer*), que recebe um sinal do programa ou serviço que esta sendo monitorado e caso este sinal não seja recebido, o *watchdog* irá se manifestar na forma de erro. Um outro exemplo é o esquema de duplicação e comparação, onde são realizadas operações em componentes replicados com os mesmos dados de entrada, e então os dados de saída são comparados. No caso de diferenças nos dados de saída um erro é gerado.
- Confinamento: responsável pela restrição de um erro para que dados inválidos não se propaguem para todo o sistema, pois entre a falha e a detecção do erro há um intervalo de tempo. Neste intervalo pode ocorrer a propagação do erro para outros componentes do sistema, sendo assim, antes de executar medidas corretivas é necessário definir os limites da propagação. Na fase de projeto essas restrições devem ser previstas e tratadas. Um exemplo de confinamento é o isolamento de alguns processos que estão em execução em um sistema operacional. Neste caso o sistema faz o gerenciamento dos processos para isolar e impedir que as falhas de um processo gerem problemas em outros processos.
- Recuperação: após a detecção de um erro ocorre a recuperação, onde o estado de erro é alterado para estado livre de erros. A recuperação pode ser feita de duas formas, que são:

  - *forward error recovery* (recuperação por avanço): ocorre uma condução para um estado que ainda não ocorreu. É a forma de recuperação mais eficiente, porém mais complexa de ser implementada.
  - *backward error recovery* (recuperação por retorno): ocorre um retorno para um estado anterior e livre de erros. Para retornar ao estado anterior

podem ser utilizados pontos de recuperação (*checkpoints*). Assim, quando ocorrer um erro, um *rollback* é executado, ou seja, o sistema retornará a um estado anterior a falha.

- Tratamento: procura prevenir que futuros erros aconteçam. Nesta fase ocorre a localização da falha para descobrir o componente que originou a falha. A substituição do componente danificado pode ser feita de forma manual ou automática. O reparo manual é feito por um operador que é responsável pelo reparo ou substituição de um componente. Como exemplo pode-se citar a troca de um disco rígido de um servidor. Já o reparo automático é utilizado quando existe um componente em espera para a substituição, como por exemplo, um disco configurado como *hot spare*, ou seja, um componente de *backup* que assumirá o lugar do outro imediatamente após o componente principal falhar. Em *storages* ou servidores, o *hot spare* pode ser configurado através de um *Redundant array of independent disks* (RAID) (ROUSE, 2013).

Definir failover ??

2.2 Redundância

A redundância pode ser implementada através da replicação de componentes, e apresenta como objetivo reduzir o número de SPOF e garantir o mascaramento de falhas. Na prática, se um componente falhar ele deve ser reparado ou substituído por um novo, sem que haja uma interrupção no serviço. Além disso, a redundância pode ser implementada através do envio de sinais ou *bits* de controle junto aos dados, servindo assim para detecção e correção de erros (WEBER, 2002). Segundo (NøRVåG, 2000) existem quatro tipos diferentes de redundância que são:

- *Hardware*: utiliza-se a replicação de componentes, sendo que no caso de falha em um deles o outro possa assumir seu lugar. Para fazer a detecção de erros a saída de cada componente é constantemente monitorada e comparada à saída do outro componente. Um exemplo prático de redundância de *hardware* são os servidores com fontes redundantes. Neste caso são utilizadas duas fontes ligadas em paralelo, sendo que, caso uma falhe a outra suprirá a necessidade de todo o servidor;
- Informação: ocorre quando uma informação extra é enviada ou armazenada para possibilitar a detecção e a correção de erros. Um exemplo são os *checksums* (soma de verificação). Esses são calculados antes da transmissão ou armazenamento dos dados e recalculados ao recebê-los ou recuperá-los, assim sendo possível verificar a integridade dos dados. Outro exemplo bastante comum são os *bits* de paridade que são utilizados para detectar falhas que afetam apenas um *bit* (WEBER, 2002);
- *Software*: pode-se definir redundância de *software* como a configuração de um serviço ou *software* em dois ou mais locais diferentes. Pode-se citar como exemplo um sistema gerenciador de banco de dados *MySQL*, que pode ser configurado com um modelo de replicação do tipo *master-slave*, onde um servidor principal (*master*) grava as operações em um arquivo, para que então os servidores *slaves*, possam recuperar e executar essas operações, com isso mantendo os dados sincronizados. Neste caso, tanto o servidor *master* quanto os *slaves*

executam o serviço *MySQL*, caracterizando uma redundância (SILVA VIANA, 2015). A redundância de *software* também pode ser implementada com o objetivo de tolerar falhas e *bugs* em um *software* crítico. Existem algumas técnicas que podem ser utilizadas para isso, como por exemplo, a programação de n-versões, que consiste no desenvolvimento de n versões de um mesmo *software*, desta forma, possibilita-se o aumento da disponibilidade, uma vez que elas provavelmente não apresentarão os mesmos erros. A programação de n-versões possui um custo muito elevado, não sendo muito utilizada.

- Tempo: este é feito através da repetição de um conjunto de instruções em um mesmo componente, assim detectando uma falha caso ocorra. Essa técnica necessita tempo adicional, e é utilizada em sistemas onde o tempo não é crítico. Por exemplo, um *software* de monitoramento de serviços que faz um teste em cada serviço. No caso de ocorrência de uma falha em um serviço, uma ação corretiva será executada para reestabelecer este serviço. Essa técnica, diferentemente da redundância de *hardware*, não requer um *hardware* extra para sua implementação (COSTA, 2009).

2.3 Cálculo da alta disponibilidade

Um aspecto importante sobre alta disponibilidade é como medi-la. Para isso são utilizados os valores de *uptime* e *downtime*, que são respectivamente o tempo em que os serviços estão em execução e o tempo em que não estão executando. A alta disponibilidade pode ser expressa pela quantidade de "noves", isto é, se um serviço possui quatro noves de disponibilidade, este possui uma disponibilidade de 99,99% (PEREIRA FILHO, 2004).

A Tabela 2.1 apresenta alguns níveis de disponibilidade, e os seus percentuais de *Uptime* e os *Downtime* por ano. Já na última coluna tem-se alguns exemplos de serviços relacionados ao nível de disponibilidade. Pode-se observar que para alguns serviços, como por exemplo, sistemas bancários ou sistemas militares é necessário um alto nível de disponibilidade (PEREIRA FILHO, 2004).

Nível	Uptime	Downtime por ano	Exemplos
1	90%	36.5 dias	computadores pessoais
2	98%	7.3 dias	
3	99%	3.65 dias	sistemas de acesso
4	99.8%	17 horas e 30 minutos	
5	99.9%	8 horas e 45 minutos	provedores de acesso
6	99.99%	52.5 minutos	CPD, sistemas de negócios
7	99.999%	5.25 minutos	sistemas de telefonia ou bancários
8	99.9999%	31.5 minutos	sistemas de defesa militar

Tabela 2.1: Níveis de alta disponibilidade e exemplos de sistemas

A porcentagem de disponibilidade (d) pode ser calculada através da equação

$$d = \frac{MTBF}{(MTBF + MTTR)} \tag{2.1}$$

onde o *Mean time between failures* (MTBF) corresponde ao tempo médio entre falhas, ou seja, corresponde ao tempo médio entre as paradas de um serviço. Já

o *Mean time to repair* (MTTR) é o tempo médio de recuperação, isto é, o tempo entre a queda e a recuperação de um serviço (GONçALVES, 2009).

A alta disponibilidade é um dos principais fatores que fornece confiança aos clientes ou usuários de um serviço, sendo extremante importante em empresas que fornecem serviços *on-line*. Por isso, as empresas desenvolveram o *Service level agreement* (SLA), que é um acordo de nível de serviço, o qual garante que o serviço fornecido atenda as expectativas dos clientes. Um SLA é um documento contendo uma descrição e uma definição das características mais importantes do serviço que será fornecido. Esse acordo apresenta também o percentual de disponibilidade do serviço. Além disso, um SLA deverá conter descrição do serviço, requerimentos, horário de funcionamento, *uptime* do serviço, *downtime* máximo do serviço, entre outros (SMITH, 2010).

2.4 Considerações finais

Neste capítulo foram descritos os principais conceitos de alta disponibilidade e conceitos relacionados. Como mencionado anteriormente, um dos principais recursos utilizados para a obtenção de alta disponibilidade é a virtualização, uma vez que essas são utilizadas para implementar a redundância de *software*. Desta forma, no próximo capítulo será feita uma breve definição de virtualização, com as vantagens e as estratégias de implementação de máquinas virtuais.

3 VIRTUALIZAÇÃO

O conceito virtualização surgiu na década de 60, onde o usuário muitas vezes necessitava de um ambiente individual, com suas próprias aplicações e totalmente isolado dos demais usuários. Esse foi um dos principais motivos para a criação das máquinas virtuais, que na época eram conhecidas como *Virtual machine* (VM). As VMs apresentaram uma forte expansão com o sistema operacional *370*, que foi desenvolvido pela *IBM*, e foi um dos principais sistemas comerciais com suporte à virtualização da época. Esse sistema operacional era executado em *mainframes*, que eram grandes servidores capazes de processar um grande volume de informações (LAUREANO; MAZIERO, 2008).

Na década de 80 houve uma redução no uso da virtualização devido a popularização do *Personal computer* (PC). Na época era mais vantajoso disponibilizar um PC para cada usuário do que investir em *mainframes*. Devido à crescente melhora na performance do PC e ao surgimento da linguagem *Java*, no início da década de 90, a tecnologia de virtualização retornou com o conceito de virtualização de aplicação (LAUREANO; MAZIERO, 2008).

A virtualização foi definida nos anos 60 e 70 como uma camada entre o *hardware* e o sistema operacional que possibilitava a divisão e a proteção dos recursos físicos. Porém, atualmente ela abrange outros conceitos, como por exemplo a *Java virtual machine* (JVM), que não virtualiza um *hardware*. De fato, esta permite que uma aplicação convidada execute em diferentes tipos de sistemas operacionais.

Atualmente, define-se virtualização como uma camada de *software* que utiliza os serviços fornecidos por uma determinada interface de sistema para criar outra interface de mesmo nível. Assim, a virtualização permite a comunicação entre interfaces distintas, de forma que uma aplicação desenvolvida para uma plataforma X possa também executar em uma plataforma Y (LAUREANO; MAZIERO, 2008).

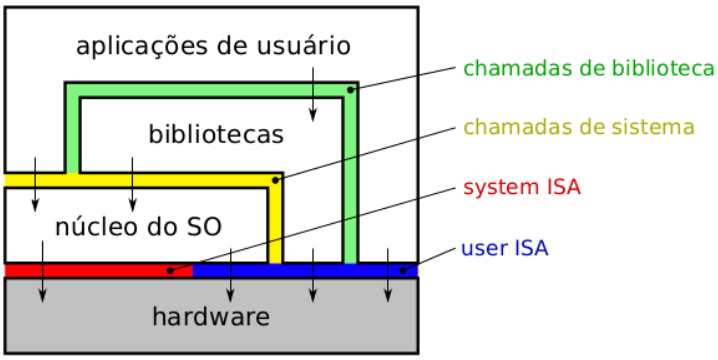
Como mencionado, a virtualização permite a comunicação entre diferentes interfaces, sendo que existem diferentes tipos de interfaces nos sistemas de computação (MAZIERO, 2013):

- Conjunto de instruções ou *Instruction set architecture* (ISA): é a interface básica, que fica entre o *software* e o *hardware*, e é composta por instruções de código de máquina. Essa interface é dividida em dois grupos:
  - Instruções de usuário ou *User ISA*: são instruções disponíveis à aplicações de usuários. Essas instruções executam em modo usuário, sendo que nesse modo há restrições de modo a garantir um controle e segurança no acesso aos recursos de *hardware*. Instruções de usuário são instruções

não privilegiadas, ou seja, são instruções que podem ser executadas sem interferir em outras tarefas, porque elas não acessam recursos compartilhados. Este grupo de interface contém, por exemplo, instruções de operações aritméticas e instruções de ponto flutuante (BUYYA; VECCHIOLA; SELVI, 2013);
  - Instruções de sistema ou *System ISA*: essas instruções geralmente são disponibilizadas para o núcleo do sistema operacional. Elas são instruções privilegiadas, ou seja, são instruções que acessam recursos compartilhados. Essas instruções são executadas em modo supervisor (ou modo *kernel*), que permite realizar operações sensíveis[1] no *hardware* (BUYYA; VECCHIOLA; SELVI, 2013). Como exemplo de instruções de sistema pode-se citar as instruções que alteram o estado dos registradores do processador;

- Chamadas de sistema ou *syscalls*: são operações oferecidas pelo núcleo do sistema operacional para as aplicações dos usuários. Essas operações permitem um acesso controlado aos dispositivos, a memória e ao processador. As instruções privilegiadas não podem ser executadas no modo usuário, por isso, as aplicações de usuários utilizam chamadas de sistemas em seu lugar, e então o sistema operacional determina se essas operações poderão comprometer a integridade do sistema (MARINESCU, 2013). Um exemplo de chamada de sistema é uma operação de escrita em disco rígido ou qualquer operação de entrada e saída feito por aplicações de usuários.

A Figura 3.1 mostra as diferentes interfaces entre aplicações de usuários, bibliotecas, núcleo do sistema operacional e o *hardware*.



Fonte: MAZIERO (2013)

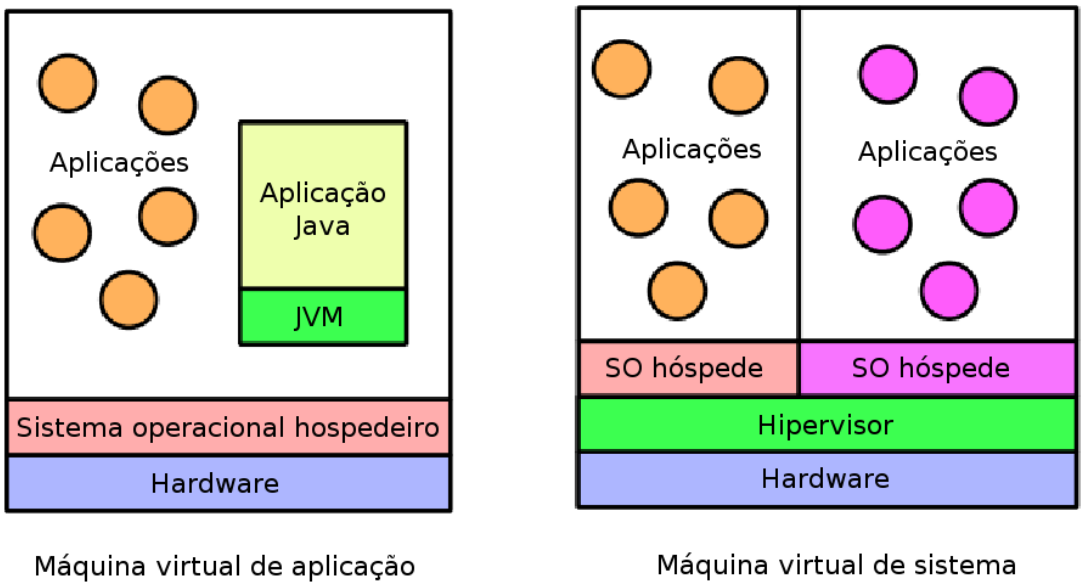
Figura 3.1: Interfaces de sistemas de computação.

Máquinas virtuais podem ser divididas em dois grupos principais, que são: as máquinas virtuais de aplicação (Seção 3.1), e máquinas virtuais de sistema (Seção 3.2). As máquinas virtuais de aplicação fazem a virtualização de uma aplicação e suportam apenas uma aplicação, ou seja, elas provêm um ambiente que permite a execução de uma aplicação convidada. Um exemplo de máquina virtual de aplicação é a JVM. Já uma máquina virtual de sistema suporta um sistema operacional convidado, com suas aplicações executando sobre ele. Uma máquina virtual executando

[1] Operações sensíveis são instruções que podem alterar o estado do processador.

sobre o hipervisor *Kernel-based Virtual Machine* (KVM) é um exemplo de máquina virtual de aplicação (LAUREANO; MAZIERO, 2008).

Na Figura 3.2 tem-se o modelo de máquina virtual de aplicações, onde uma JVM, juntamente com aplicações, está executando sobre um sistema operacional. O outro modelo é de máquina virtual de sistema, que possui dois sistemas operacionais executando sobre um único *hardware* por meio do hipervisor.



Fonte: LAUREANO; MAZIERO (2008)

Figura 3.2: Máquinas virtuais de aplicação e de sistema.

3.1 Máquinas virtuais de aplicação

As máquinas virtuais de aplicação, também chamadas de máquinas virtuais de processos, são responsáveis por prover um ambiente que permite a execução de uma aplicação convidada, sendo que esta aplicação possui um conjunto de instruções, ou de chamadas do sistema, diferentes da arquitetura do sistema hospedeiro. Neste caso, quando temos uma chamada de sistema ou instruções de máquina, será necessário uma tradução dessas interfaces, que será feita pela camada de virtualização. Os dois principais tipos de máquinas virtuais de aplicação são:

- Máquinas virtuais de linguagem de alto nível: esse tipo de máquina virtual foi criado levando em consideração uma linguagem de programação e o seu compilador. Neste caso, o código compilado gera um código intermediário que não pode ser executado em uma arquitetura real, mas pode ser executado em uma máquina virtual. Sendo assim, para cada arquitetura ou sistema operacional deverá existir uma máquina virtual que permita a execução da aplicação. Como exemplo deste tipo de máquina virtual pode-se citar a máquina virtual Java (JVM) e a *Microsoft Common Language Infrastructure*, que é a base da plataforma *.NET* (CARISSIMI, 2008);

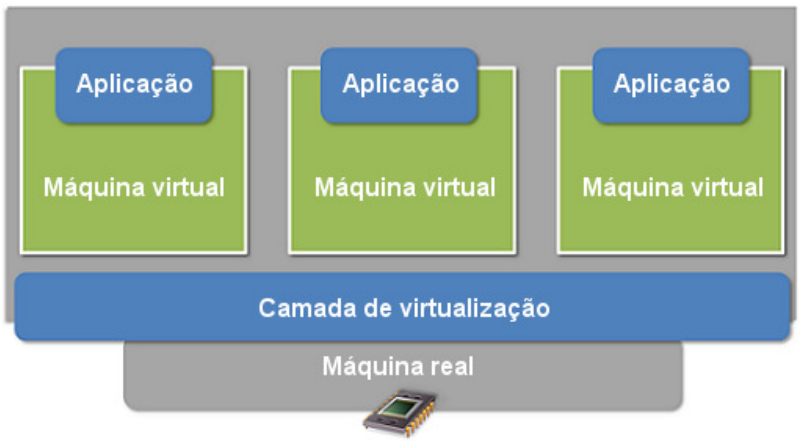
- Emulação no sistema operacional: nesse caso é feito um mapeamento entre as chamadas de sistema que são utilizadas pela aplicação e as chamadas do sistema operacional hospedeiro. A virtualização de aplicação pode ser encontrada em ferramentas que emulam uma aplicação que foi desenvolvida para uma plataforma em uma outra plataforma. Como exemplo, pode-se citar o *Wine* (WineHQ, 2016), que permite executar aplicações *Windows* em plataformas *Linux*.

3.2 Máquinas virtuais de sistema

As máquinas virtuais de sistema, também chamadas de hipervisor ou *Virtual machine monitor* (VMM), são uma camada de *software* que possibilita que múltiplos sistemas operacionais convidados executem sobre um mesmo computador físico, ou seja, o hipervisor provê uma interface ISA virtual, que pode ou não ser igual a interface real, e virtualiza outros componentes de *hardware*, para que cada máquina virtual convidada possa ter seus próprios recursos. Para tanto, a virtualização de sistema utiliza abstrações em sua arquitetura. Por exemplo, ela transforma um disco rígido físico em dois virtuais menores, sendo que esses discos virtuais são arquivos armazenados no disco físico (SMITH; NAIR, 2005).

Nesse modelo, o ambiente de virtualização de sistema é composto basicamente por três componentes (Figura 3.3):

- Máquina real: também chamada de hospedeiro, que é o *hardware* onde o sistema de virtualização irá executar;
- Camada de virtualização: é conhecida como hipervisor ou também chamada de VMM. Essa camada tem como função criar interfaces virtuais para a comunicação da máquina virtual com a máquina real;
- Máquina virtual: também conhecida como sistema convidado, sendo executado sobre a camada de virtualização. Geralmente, tem-se várias máquinas virtuais executando simultaneamente sobre esta camada.



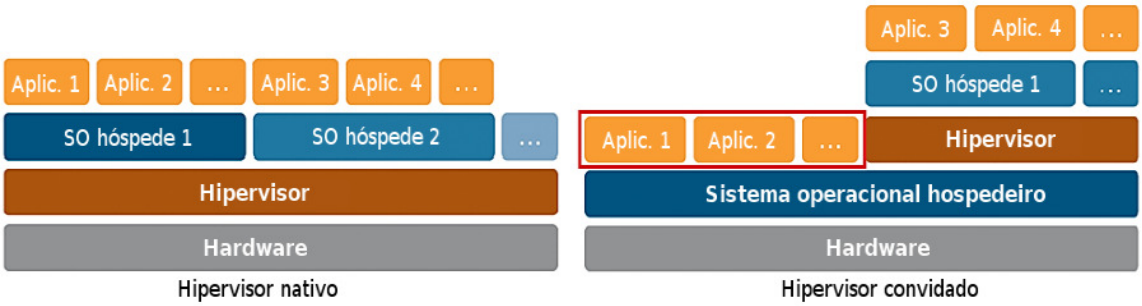
Fonte: ANDRADE (2011)

Figura 3.3: Componentes da virtualização.

3.2.1 Arquiteturas de máquinas virtuais de sistema

Existem basicamente duas arquiteturas de hipervisor de sistema, que são apresentadas na Figura 3.4 (MAZIERO, 2013):

- Hipervisores nativos: esse hipervisor executa diretamente sobre o *hardware*, ou seja, sem um sistema operacional hospedeiro. Neste caso, o hipervisor nativo faz a multiplexação dos recursos do *hardware* (memória, disco rígido, interface de rede, entre outros) e diponibiliza esses recursos para as máquinas virtuais. Alguns exemplos de sistemas que utilizam essa arquitetura de hipervisor são o *IBM 370* (IBM, 2016), o *Xen* (Citrix, 2016a) e o *VMware ESXi* (VMware, 2016a);
- Hipervisores convidados: esse tipo de hipervisor executa sobre um sistema operacional hospedeiro e utiliza os recursos desse sistema para gerar recursos para as máquinas virtuais. Normalmente esse tipo de hipervisor suporta apenas um sistema operacional convidado para cada hipervisor. Exemplos de *softwares* que possuem esse tipo de arquitetura são o *VirtualBox* (Oracle, 2016a), o *VMware Player* (VMware, 2016b) e o *QEmu* (QEmu, 2016).



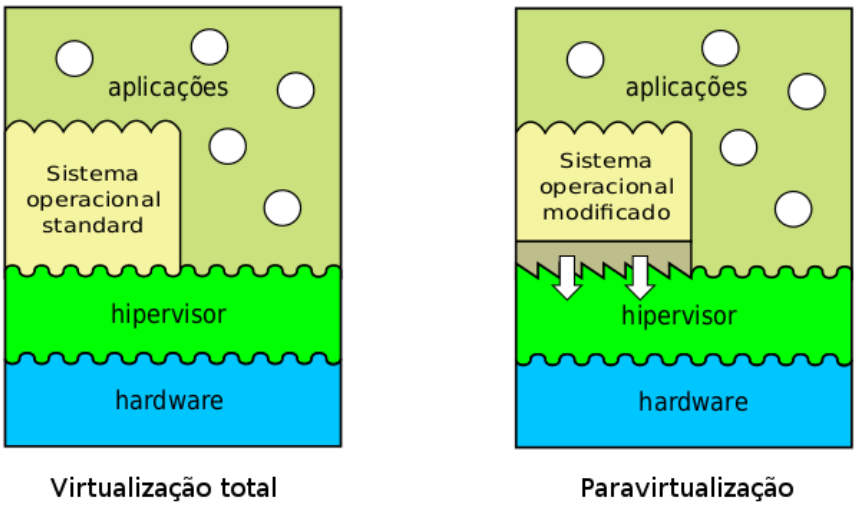
Fonte: SANTOS MACEDO; SANTOS (2016)

Figura 3.4: Arquiteturas de máquinas virtuais de sistema.

Conclui-se que hipervisores convidados são mais flexíveis que os nativos, pois podem ser facilmente instalados ou removidos de um sistema operacional existente. Já os hipervisores nativos possuem melhor desempenho pois acessam o *hardware* diretamente.

3.2.2 Implementações de máquinas virtuais de sistema

As máquinas virtuais de sistema podem ser implementadas usando diferentes estratégias. Atualmente as estratégias mais utilizadas são a virtualização total e a paravirtualização (Figura 3.5):



Fonte: MAZIERO (2013)

Figura 3.5: Implementações de máquinas virtuais de sistema.

- Virtualização total: nesta estratégia todas as interfaces de acesso ao *hardware* são virtualizadas. Desta forma, possibilita-se que os sistemas operacionais convidados executem como se estivessem diretamente sobre o *hardware*. Na virtualização total o conjunto de instruções do processador é acessível somente ao hipervisor, sendo que essa estratégia utiliza tradução dinâmica[1] para executar as instruções do sistema convidado. A grande vantagem dessa estratégia é a possibilidade de um sistema convidado ser executado sem a necessidade de ser modificado. Porém, essa estratégia possui um desempenho inferior devido ao fato do hipervisor intermediar todas as chamadas de sistemas e operações do sistema convidado. Um exemplo de ferramenta que utiliza a virtualização total é o *QEmu* (QEmu, 2016);
- Paravirtualização: nesta estratégia a interface entre o hipervisor e o sistema operacional convidado foi modificado para se obter uma eficiência significativamente maior. Destaca-se que essa estratégia de implementação utiliza uma arquitetura de hipervisor nativo. As modificações na interface de sistema (*system ISA*) exigem que o sistema operacional convidado seja adaptado para o hipervisor, para possibilitar a execução sobre a plataforma virtual. Para essa adaptação, o hipervisor disponibiliza uma *Application programming interface* (API), para que os sistemas convidados possam acessar o ISA de sistema. Contudo, a interface de usuário é mantida, assim, as aplicações do sistema convidado não precisam ser modificadas (MAZIERO, 2013).

[1] A tradução dinâmica analisa e reorganiza as instruções de um sistema convidado para melhorar o desempenho da execução, além disso a tradução dinâmica adapta as instruções do sistema convidado para o sistema real.

A paravirtualização possui um desempenho superior se comparada a virtualização total, pois acessa alguns recursos de forma direta, sendo que o hipervisor é reponsável somente por impedir que o sistema convidado execute operações indevidas. Como exemplo pode-se citar o controle de acesso à memória feito pelo hipervisor. Na virtualização total o hipervisor reserva um espaço para cada sistema convidado, que por sua vez, acessa a memória como se fosse uma memória física, ou seja, inicia o seu endereçamento na posição zero. Sendo assim, cada vez que o sistema convidado acessar a memória, o hipervisor precisará converter os endereços do sistema convidado para os endereços reais de memória. Já na paravirtualização, o hipervisor informa ao sistema convidado a área de memória que ele poderá utilizar, assim, não sendo necessário nenhuma conversão de endereços.

Apesar de apresentar um desempenho inferior, a virtualização total possui uma maior portabilidade, ou seja, permite que sistemas operacionais executem como convidados sem a necessidade de serem modificados. Assim, permitindo por exemplo, que um sistema operacional qualquer possa ser instalado em um ambiente de virtualização total. Além disso, permite virtualizar um sistema operacional já instalado em uma máquina física apenas copiando o conteúdo de seu disco rígido, sem a necessidade de reinstalar esse sistema operacional e reconfigurar todas as aplicações.

3.3 Vantagens das máquinas virtuais

De modo geral, a principal vantagem das máquinas virtuais de aplicação é a possibilidade de executar uma mesma aplicação em diversos sistemas operacionais sem a necessidade de recompilar a mesma. Já para máquinas virtuais de sistema, destaca-se a possibilidade de executar mais de um sistema operacional sobre um mesmo *hardware*. Nas próximas seções serão descritas as principais utilizações e vantagens da virtualização de *desktops* e de servidores.

3.3.1 Virtualização de Desktop

A portabilidade é uma das grandes vantagens da virtualização, que também pode ser aplicada em *desktops*. Pode-se citar como exemplo, o desenvolvimento de *software* para diversos sistemas operacionais sem a necessidade de aquisição de um computador físico para a instalação de cada sistema operacional. Assim, a virtualização de *desktops* pode ser utilizada em ambientes de desenvolvimento, pois possibilitam a execução de múltiplas plataformas sem comprometer o sistema operacional original (CARISSIMI, 2008). Um exemplo é o *VMware Workstation*, que possibilita a virtualização em PC para fins de desenvolvimento de *software* (WMware, 2016).

Em empresas pode-se ainda utilizar virtualização de *desktops*, através da configuração de terminais remotos nos computadores e um servidor para centralizar as máquinas virtuais. Com isso torna-se mais simples a manutenção dos *desktops*, além disso, estes necessitam de um *hardware* de menor valor, uma vez que esses executarão apenas um terminal remoto. Por fim, essa técnica possibilita uma maior segurança dos dados, pois os dados serão armazenados em um local seguro, como por exemplo, um *datacenter*. Exemplos desse tipo de virtualização são o *Xen Desktop* (Citrix, 2016b) e o *VMware Horizon View* (VMware, 2016c).

Para usuários de computadores em geral a virtualização também é interessante, uma vez que esses podem necessitar de um *software* que não está disponível para a plataforma utilizada. Deste modo, a virtualização possibilita executar diferen-

tes sistemas operacionais no computador do usuário. Por exemplo, para usuários *MacOS* é comum a necessidade de executar aplicações que não existem para a sua plataforma, sendo assim, pode-se utilizar uma máquina virtual para executar essas aplicações.

Pode-se também encontrar virtualização de *desktops* em laboratórios de ensino, devido a necessidade de executar diferentes sistemas operacionais para determinadas disciplinas. Isso é necessário quando pretende-se configurar e executar aplicações para fim de experimentos ou aprendizagem, com isso, essas ações não afetarão o sistema hospedeiro. A grande vantagem da utilização de máquinas virtuais nesse tipo de ambiente é a facilidade na manutenção, pois as máquinas virtuais podem ser restauradas de forma simples.

3.3.2 Virtualização de servidores

Em muitos casos as empresas utilizam serviços distribuídos entre diferentes servidores físicos, como por exemplo, servidores de *e-mail*, hospedagens e banco de dados. Essa estrutura faz com que alguns recursos fiquem ociosos, pois em muitos casos esses serviços necessitam de menos recursos que o servidor físico fornece. Por exemplo, um serviço de transmissão de *streaming* de áudio utiliza pouco acesso ao disco rígido, porém utiliza um poder de processamento e memória RAM maior. Portanto, uma das grandes vantagens da virtualização é um melhor aproveitamento dos recursos. De fato, alocando vários serviços em um único servidor físico tem-se um melhor aproveitamento do *hardware* (MOREIRA, 2006), e consequentemente, tem-se uma redução de custos com a administração e a manutenção dos servidores físicos.

Em um ambiente heterogênio pode-se também utilizar virtualização, pois ela permite a instalação de diversos sistemas operacionais em um único servidor. Esse tipo de virtualização favorece a implementação do conceito "um servidor por serviço", que consiste em ter um servidor para cada serviço. Além disso, tem-se o isolamento de serviços, ou seja, caso ocorra uma falha de segurança em um serviço, essa falha não comprometerá todo o sistema, uma vez que cada serviço estará executando em seu próprio sistema operacional (CARISSIMI, 2008).

Outra motivação para a utilização de virtualização em servidores consiste na redução de custos com energia elétrica e equipe. Essa redução de custos pode ser obtida através da implantação de servidores mais robustos para substituir dezenas de servidores comuns. Além disso, pode-se obter uma redução nos custos com refrigeração, uma vez que essa estrutura proporciona uma redução no número de servidores e de espaço físico necessitado para esses.

Por fim, existe uma técnica chamada *live migration*, ou migração em tempo real. Essa técnica possibilita que uma máquina virtual, que está executando em um servidor físico, seja transferida através da rede para outro servidor sem ser reiniciada. Nesse processo a máquina virtual fica no estado suspenso (por um período curto de tempo) até que o servidor de destino receba os dados necessários para continuar a execução da máquina virtual (SILVA, 2009). Essa técnica possibilita a utilização de redundância de *software* e fará parte da implementação deste trabalho.

3.4 Considerações finais

Neste capítulo foi apresentado um breve histórico da virtualização e os dois principais grupos de máquinas virtuais que são: máquinas virtuais de aplicação e máquinas virtuais de sistema. Também foram apresentadas as vantagens e as estratégias de implementação de máquinas virtuais de sistema, dando ênfase para as máquinas virtuais de sistema, uma vez que essas serão utilizadas no desenvolvimento deste trabalho. De fato, essas serão utilizadas para a implementação da redundância de *software*. No próximo capítulo será feito o levantamento e análise dos serviços fornecidos pela empresa que é o objetivo do estudo deste trabalho. Posteriormente, serão apresentados os serviços que são considerados críticos e a proposta de solução de alta disponibilidade.

4 ESTUDO DO AMBIENTE DA EMPRESA

Este capítulo apresentará a estrutura da empresa que será objetivo de estudo neste trabalho. Esta é uma empresa que fornece serviços de hospedagens e também está associada a um provedor de Internet[1]. A empresa possui grande parte de seus clientes localizados na serra do Rio Grande do Sul, sendo que atualmente possui aproximadamente 9000 clientes. A sede da empresa está localizada na cidade de Garibaldi, além disso possui quatro filiais no estado, atendendo aproximadamente 30 cidades.

A empresa oferece serviços pela Internet aos seus clientes, sendo eles: hospedagens de sites, banco de dados, *e-mail*, sistemas de gestão, *e-mail marketing*, *backup*, *máquinas virtuais*, autenticação via *Asymmetric Digital Subscriber Line* (ADSL), rádio *online* e telefonia. Além disso, o provedor associado fornece aos seus clientes acesso à Internet via rádio e acesso à Internet por meio de fibra óptica. A maioria dos serviços são fornecidos por meio de *softwares* de código aberto.

Atualmente, a empresa possui redundância de refrigeração e de energia, como pode ser observado na Figura 4.1. A redundância de refrigeração é composta por dois ares-condicionados (destacados com a cor azul).

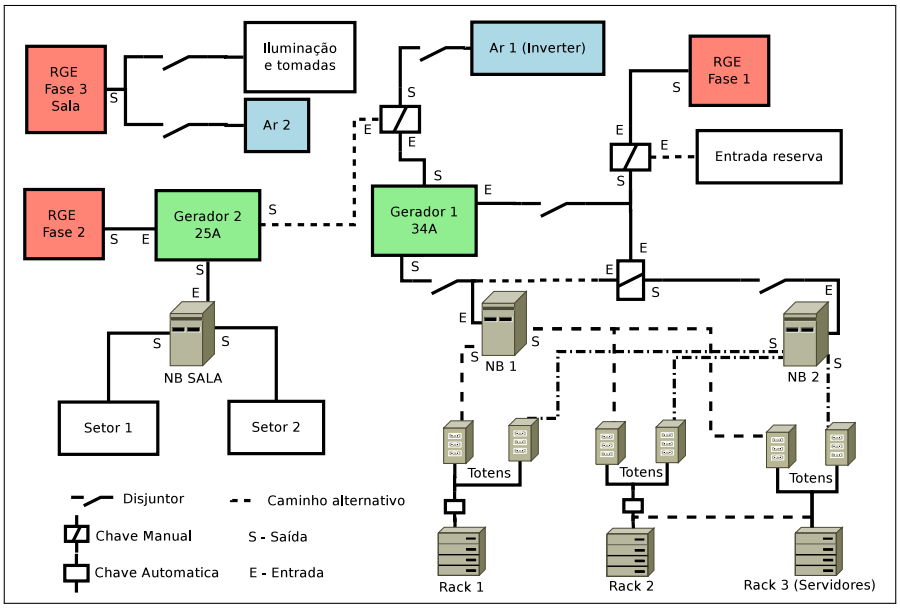


Figura 4.1: Diagrama de instalação elétrica.

[1]É importante salientar que esse provedor utiliza a maior parte dos serviços da empresa, pois possui um número de clientes maior que a empresa.

A redundância de energia é feita através de três *nobreaks*, sendo que dois deles (identificados como NB 1 e NB 2) são utilizados para alimentação dos servidores e outros equipamentos, como por exemplo, roteadores. Assim, se um *nobreak* falhar o outro assume a alimentação de todos os equipamentos. O terceiro *nobreak* (identificado como NB Sala) é utilizado para alimentar os computadores dos funcionários de dois setores da empresa. Conectado aos *nobreaks* estão seis *totens*[1], sendo que nesses *totens* são ligados os equipamentos e servidores que estão montados nos *racks*. Além disso, existe a entrada de energia, com três fases (destacadas na cor vermelho), e dois geradores (destacados na cor verde) que suprem a necessidade de consumo de energia elétrica do ambiente.

Nas próximas seções será feita uma descrição da estrutura da empresa. Na Seção 4.1 será descrito a instalação física dos servidores, com suas estruturas e suas configurações. Na Seção 4.2 será descrito os servidores que não utilizam virtualização, ou seja, possuem os seus serviços instalados diretamente sobre o sistema operacional. Na Seção 4.3 será descrito a estrutura dos servidores que utilizam de virtualização e todos os serviços fornecidos por esses servidores.

4.1 Instalação física

A estrutura atual da empresa é composta por quatorze servidores físicos. A configuração de *hardware* desses servidores pode ser encontrada na Tabela 4.1, onde tem-se o nome do servidor, o modelo, a configuração dos processadores, quantidade de memória *Random access memory* (RAM), número de discos rígidos e a capacidade unitária de cada disco.

Servidor	Modelo	Processador	Memória RAM	Disco
Bello		1 x Intel Core 2 Duo E6750 2.66 GHz	2 GB DDR2	5,5 TB SATA
Cacti	Dell PowerEdge 2950	2 x Intel Xeon E5310 1.60 GHz	12 GB DDR2	2 x 73 GB SAS
Dati	Dell PowerEdge 1850	2 x Intel Xeon 3.20 GHz	4 GB DDR2	2 x 146 GB SCSI
Monit		1 x Intel Core 2 Quad Q9550 2.83 GHz	4 GB DDR2	120 GB SSD
Nino		1 x Intel Core 2 Duo E4500 2.20 GHz	4 GB DDR2	500 GB SATA
Sfrunhon		1 x Intel Xeon X3330 2.66 GHz	8 GB DDR2	750 GB SATA
Vigilante		1 x Intel Pentium Dual E2180 2.00 GHz	4 GB DDR2	2,5 TB SATA
Brina	Dell PowerEdge 2950	2 x Intel Xeon E5410 2.33 GHz	24 GB DDR2	6 x 300 GB SAS
Fulmine	IBM System x3650 M4	1 x Intel Xeon E5-2650 2.00 GHz	32 GB DDR3	6 x 2 TB SATA
Piova	Dell PowerEdge R410	2 x Intel Xeon E5530 2.40 GHz	32 GB DDR3	4 x 500G SATA
Raggio	HP ProLiant DL360 G7	2 x Intel Xeon E5630 2.53 GHz	32 GB DDR3	4 x 300 GB SAS
Tempesta	Dell PowerEdge R620	2 x Intel Xeon E5-2620 2.00 GHz	32 GB DDR3	5 x 1 TB SATA 3 x 1,2 TB SAS
Tuono	HP ProLiant DL380 G7	2 x Intel Xeon E5649 2.53 GHz	32 GB DDR3	6 x 300 GB SAS 2 x 146 GB SAS
Venti	Dell PowerEdge R210 II	1 x Intel Xeon E3-1220 3.10 GHz	16 GB DDR3	2 x 3 TB SATA

Tabela 4.1: Configuração dos servidores físicos.

Todos os servidores estão ligados ao *switch*, que provê aos servidores acesso à Internet através de um roteador. Para os servidores mais importantes são utilizados dois cabos de rede que estão ligados a um *switch gigabit*, assim possibilitando a configuração de um *link aggregation*, que permite configurar mais de uma interface de rede física em uma interface agregada. Através deste *link aggregation* pode-se dobrar a capacidade de tráfego de dados e obter redundância de interfaces. O

[1] *Totens* são torres que possuem tomadas para plugar os equipamentos

diagrama da Figura 4.2 demonstra uma visão geral da estrutura física dos servidores da empresa.

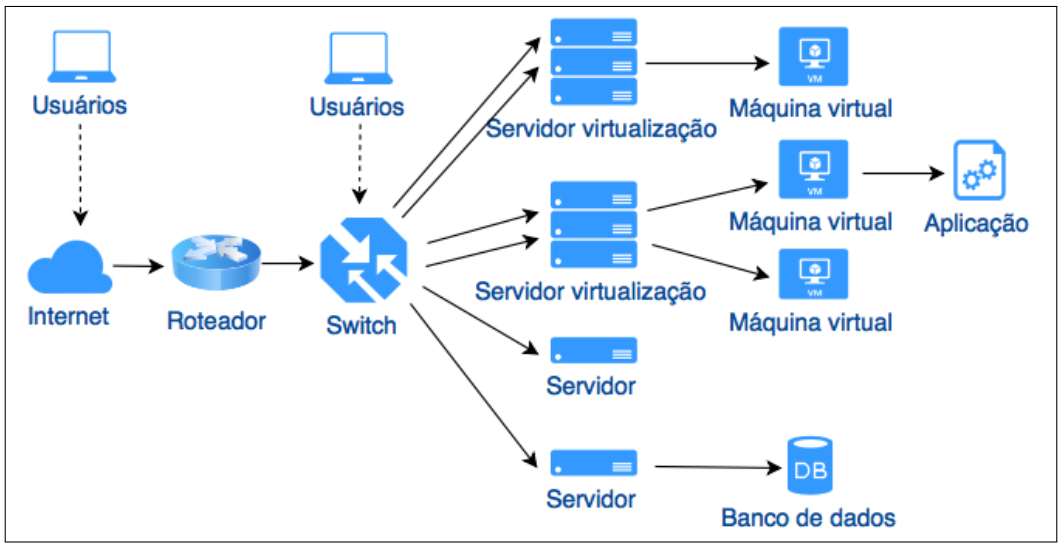


Figura 4.2: Modelo de estrutura física.

A Figura 4.3 demonstra, através de uma foto, o *rack* e todos os servidores, inclusive o *switch*.

Figura 4.3: Imagem do *rack* e dos servidores.

4.2 Servidores sem virtualização

Atualmente existem sete servidores que possuem serviços que são executados sobre o sistema operacional nativo, ou seja, sem virtualização. Eles são os sete primeiros servidores da Tabela 4.1, e estão descritos abaixo:

- Bello: esse servidor possui o sistema operacional *Ubuntu 14.04 Long term support (LTS)* (Canonical, 2016). Sua função é armazenar todos os dados de *backup*, para isso ele possui instalado a ferramenta *Bacula Storage 5.2.6* (Bacula, 2016). Destaca-se que a ferramenta *Bacula director* é o *software* que faz a gerência e controle do *backup* e encontra-se instalado no servidor *Quebei*, que será detalhado na Seção 4.3.7;
- Cacti: um dos servidores de monitoramento da rede do provedor. Esse utiliza a distribuição *CentOS 6.6* (CentOS, 2016) e executa a aplicação *Cacti 0.8.8b* (Cacti, 2016), que é uma ferramenta de código aberto desenvolvida para monitorar equipamentos de rede que suportem o protocolo *Simple Network Management Protocol* (SNMP). Essa ferramenta monitora a maior parte da rede *core* e da rede *backbone*, tanto dos clientes de Internet via rádio, como de fibra óptica;
- Dati: é o servidor de banco de dados principal, sendo que esse possui o sistema operacional *Ubuntu 14.04 LTS* (Canonical, 2016). O serviço que executa sobre esse servidor é um sistema gerenciador de banco de dados *MySQL 5.5.49* (Oracle, 2016b), que armazena os dados das aplicações *ZoneMinder* (ZoneMinder, 2016) (servidor de câmeras), *Icewarp Server* (Icewarp, 2016) (servidor de *e-mail*) e *Ejabberd* (ProcessOne, 2016) (servidor de mensagens instantâneas), que serão detalhados na Seção 4.3;
- Monit: esse servidor faz o monitoramento dos demais servidores. Ele possui o sistema operacional *Ubuntu 12.04 LTS* (Canonical, 2016), e executa as aplicações *Nagios 3.2.3* (Nagios, 2016) e *Munin 1.4.6* (Munin, 2016), ambos *softwares* livres. O *Nagios* monitora o *hardware* e os serviços que estão executando em cada servidor. O segundo, *Munin*, é responsável por gerar gráficos de monitoramento. Com ele pode-se criar, por exemplo, gráficos com a utilização do processador, da memória RAM, do disco, da temperatura e da velocidade dos *fans*;
- Nino: esse é o servidor utilizado pelo setor de desenvolvimento de *software*. Suas aplicações executam sobre o sistema operacional *Ubuntu 14.04 LTS* (Canonical, 2016), sendo que os serviços fornecidos pelo servidor são: um servidor *Web Apache 2.4.7* (Apache Software Foundation, 2016a); *Personal Home Page (PHP) 5.5.9* (PHP Group, 2016); sistema gerenciador de banco de dados *MySQL 5.5.49* (Oracle, 2016b) e *PostgreSQL 9.3.13* (PostgreSQL Group, 2016); compartilhamento de arquivos *Samba 4.3.9* (Samba Team, 2016); controle de versões de *software Subversion (SVN) 1.8.8* (Apache Software Foundation, 2016b); gerenciador de *bugs Trac 1.0.1* (Edgewall Software, 2016); e o gerenciador de mensagens instantâneas para ambiente de testes *Ejabberd 2.1.11* (ProcessOne, 2016);
- Sfrunhon: é um outro servidor de monitoramento da rede do provedor. Esse utiliza a distribuição *CentOS 6.3* (CentOS, 2016) e executa a aplicação *Cacti 0.8.8a* (Cacti, 2016). Esse servidor monitora o tráfego dos clientes, tanto de Internet via rádio, como de fibra óptica;

- Vigilante: esse servidor é responsável por capturar e armazenar o *streaming* de vídeo das câmeras de segurança do provedor. Esse possui o sistema operacional *Ubuntu 14.04 LTS* (Canonical, 2016) e executa a aplicação *ZoneMinder 1.29* (ZoneMinder, 2016), que é o *software* responsável pela captura e armazenamento das imagens das câmeras do provedor.

4.3 Servidores com virtualização

Os servidores de virtualização possuem suas respectivas VMs, sendo que, para a virtualização utiliza-se o hipervisor KVM e a ferramenta *QEmu*, que são projetos de *software* livre. Procurou-se manter um ambiente homogêneo com o objetivo de facilitar a manutenção, para isso utilizou-se o mesmo hipervisor e o mesmo sistema operacional para os servidores de virtualização (hospedeiros). Esse sistema operacional é o sistema de código aberto *Ubuntu 14.04 LTS* (Canonical, 2016). Além disso, esses servidores possuem redundância de *hardware*, com fonte de alimentação duplicada e discos rígidos configurados através de um RAID. Em servidores com mais de dois discos é utilizado RAID 5. Já em servidores que possuem apenas dois discos é utilizado RAID 1 (espelhamento de discos). O ambiente também possui uma redundância do cabeamento, como visto anteriormente.

A empresa fornece serviços diversos, desde hospedagens de sites até *Domain name system* (DNS) recursivo para o provedor de Internet. Atualmente sete servidores são utilizados com virtualização (os últimos sete servidores da Tabela 4.1), sendo que existem quarenta e seis VMs distribuídas entre os sete servidores de virtualização. Nas próximas seções serão descritos esses servidores, suas respectivas máquinas virtuais e serviços.

4.3.1 Servidor Brina

O servidor Brina possui duas VMs, como pode ser visto na Figura 4.4, sendo que os serviços executados nessas VMs são os seguintes:

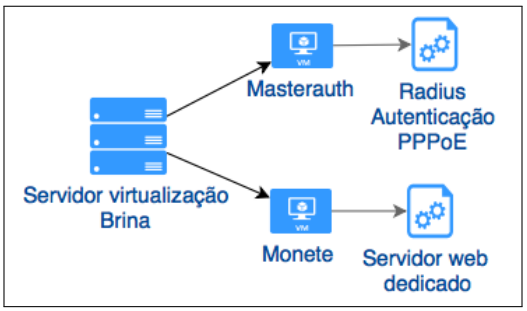


Figura 4.4: Servidor de virtualização Brina.

- *Masterauth*: sua configuração é 1 *core* de *2.33 GHz* para processamento, 1,5 GB de memória RAM e 8 GB de disco. O sistema operacional é o *Ubuntu 14.04 LTS* (Canonical, 2016), sendo que esse servidor fornece um serviço de autenticação *Point-to-point protocol over ethernet* (PPPoE) para uma parte dos clientes do provedor. Para essa autenticação é utilizada o *software Freeradius 2.1.12* (FreeRADIUS, 2016);
- *Monete*: sua configuração é 1 *core* de *2.33 GHz* para processamento, 3 GB de

memória RAM e 50 GB de disco. Esse servidor possui o sistema operacional *Ubuntu 14.04 LTS* (Canonical, 2016), e é um servidor *Web* dedicado para o site do provedor, ou seja, armazena o *Web* site do provedor. Para isso ele utiliza os *softwares Apache 2.4.7* (Apache Software Foundation, 2016a), *PHP 5.5.9* (PHP Group, 2016) (com *PHP-FPM 5.59*) e *MySQL 5.5.49* (Oracle, 2016b).

4.3.2 Servidor Fulmine

O servidor Fulmine possui dez VMs, como pode ser visto na Figura 4.5, sendo que os serviços executados nessas VMs são os seguintes:

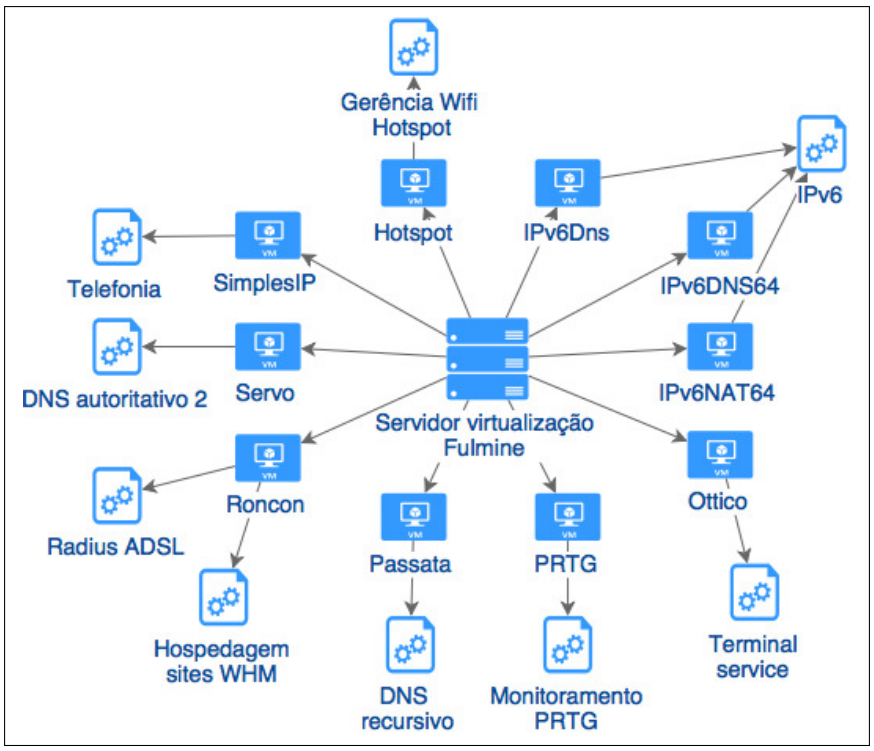


Figura 4.5: Servidor de virtualização Fulmine.

- *Hotspot*: sua configuração é 1 *core* de *2.0 GHz* para processamento, 1,5 GB de memória RAM e 8 GB de disco. Esse servidor possui o sistema operacional *Ubuntu 14.04 LTS* (Canonical, 2016), sendo que esse servidor faz a gerência de equipamentos da *Ubiquiti* que fazem *hotspot*. Esses equipamentos disponibilizam a tecnologia *Wi-fi* para prover acesso à Internet em ambientes públicos e são utilizados pelo provedor;
- *IPv6Dns*, *IPv6Dns64* e *IPv6Nat64*: suas configurações são 1 *core* de *2.0 GHz* para processamento, 1 GB de memória RAM e 8 GB de disco. O sistema operacional é o *Ubuntu 14.04 LTS* (Canonical, 2016), sendo que esses servidores fornecem o serviço de DNS e *Network address translation* (NAT) para navegação *Internet protocol version 6* (IPv6) do provedor;
- *Ottico*: esse servidor possui 2 *cores* de *2.0 GHz* para processamento, 4 GB de memória RAM e 50 GB de disco. O servidor possui o sistema operacional *Windows 2007 Server Standard* e possui o serviço de *terminal service* para suporte e gerência da infraestrutura de fibra óptica do provedor;

- *Paessler router traffic grapher* (PRTG): esse servidor possui 2 *cores* de *2.0 GHz* para processamento, 4 GB de memória RAM e 100 GB de disco. O servidor possui o sistema operacional *Windows 2008 Server R2* e sua função é fazer o monitoramento de tráfego dos equipamentos da rede *core* do provedor;
- *Passata*: esse servidor possui 2 *cores* de *2.0 GHz* para processamento, 3 GB de memória RAM e 20 GB de disco. O servidor possui o sistema operacional *Ubuntu 14.04 LTS* (Canonical, 2016) e fornece o serviço de DNS recursivo, através do *software Bind 9.9.5* (ISC, 2016). Esse é o servidor primário de DNS, sendo o mais importante para navegação dos clientes de todo o provedor;
- *Roncon*: esse servidor possui 4 *cores* de *2.0 GHz* para processamento, 6 GB de memória RAM e 400 GB de disco. Ele possui o sistema operacional *Red Hat 5.11* (Red Hat Inc, 2016) e provê hospedagem de sites *Web* desenvolvidos com a linguagem PHP. Nele está instalado o *software WebHost manager* (WHM) (cPanel and WHM, 2016), que faz a gerência dos serviços de hospedagens desses sites e banco de dados. Além disso, encontra-se disponível a ferramenta *cPanel*, que faz parte do WHM e fornece acesso aos desenvolvedores desses sites. Para fornecer essa hospedagem os seguintes *softwares* estão instalados: *Apache 2.2.26* (Apache Software Foundation, 2016a), *PHP 5.3.27* (PHP Group, 2016), *MySQL 5.1.73* (Oracle, 2016b) e *PostgreSQL 8.4.20* (PostgreSQL Group, 2016). Além da hospedagem, esse servidor fornece o serviço de autenticação ADSL de terceiros utilizando o *software Freeradius 1.1.3* (FreeRADIUS, 2016);
- *Servo*: sua configuração é 1 *core* de *2.0 GHz* para processamento, 2 GB de memória RAM e 30 GB de disco. Esse servidor possui o sistema operacional *CentOS 6.8* (CentOS, 2016), sendo que esse servidor fornece, através do *software Bind 9.8.2* (ISC, 2016), o serviço de DNS autoritativo. Esse é o servidor DNS secundário dos domínios hospedados pela empresa;
- *SimplesIP*: esse servidor possui 2 *cores* de *2.0 GHz* para processamento, 3 GB de memória RAM e 80 GB de disco. O servidor possui o sistema operacional *CentOS 6.6* (CentOS, 2016) e é o servidor de telefonia sobre *Internet protocol* (IP) do provedor. Esse utiliza como base o *software* livre *Asterisk 1.8.32* (Digium, 2016).

4.3.3 Servidor Piova

O servidor Piova possui nove VMs, como pode ser visto na Figura 4.6, sendo que os serviços executados nessas VMs são os seguintes:

- *ASP*: esse servidor possui 1 *core* de *2.40 GHz* para processamento, 1 GB de memória RAM e 50 GB de disco. O servidor possui o sistema operacional *Windows 2008 Server R2* e provê acesso a sites *Web* desenvolvidos com a linguagem *Active Server Pages* (ASP) (Microsoft, 2016a), através do *software Internet information services (IIS) 7.5* (Microsoft, 2016b). Destaca-se que esse servidor hospeda um número baixo de sites;
- *CactiBackbone*: esse servidor possui 1 *core* de *2.40 GHz* para processamento, 1 GB de memória RAM e 20 GB de disco. Ele é um servidor de monitoramento da rede do provedor. Esse utiliza a distribuição *CentOS 6.3* (CentOS, 2016) e executa a aplicação *Cacti 0.8.8a* (Cacti, 2016). Essa aplicação monitora atualmente uma parte da rede *backbone* do provedor;

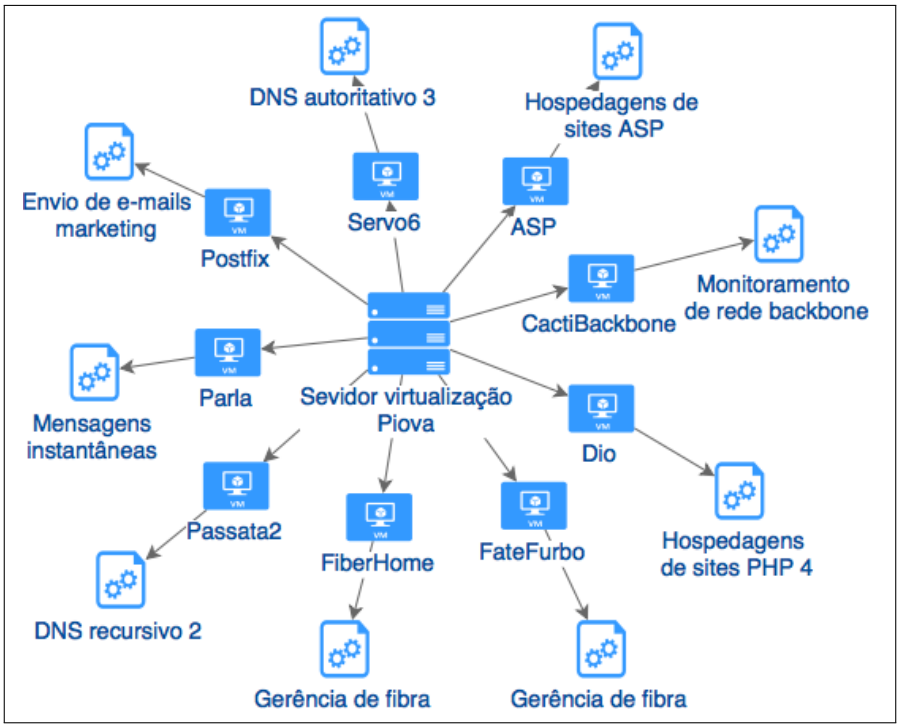


Figura 4.6: Servidor de virtualização Piova.

- *Dio*: esse servidor possui 1 *core* de *2.40 GHz* para processamento, 1 GB de memória RAM e 17,8 GB de disco. O servidor possui o sistema operacional *Ubuntu 6.06 LTS* (Canonical, 2016) e fornece serviço de hospedagens de sites desenvolvidos com a linguagem PHP versão 4.4.2. Esses sites são mantidos em um servidor separado devido a incompatibilidade com a versão 5. Esse servidor armazena aproximadamente 10 sites;
- *FateFurbo*: sua configuração é 2 *cores* de *2.40 GHz* para processamento, 4 GB de memória RAM e 80 GB de disco. O sistema operacional é o *Ubuntu 14.04 LTS* (Canonical, 2016) e esse possui um *software* proprietário da empresa *Padtec*, que faz a gerência de uma parte da rede de fibra óptica do provedor;
- *FiberHome*: sua configuração é 2 *cores* de *2.40 GHz* para processamento, 2 GB de memória RAM e 60 GB de disco. Esse servidor possui o sistema operacional *Windows XP* e possui o *software ANM 2000* instalado. Esse *software* é utilizado a gerência da fibra óptica do provedor;
- *Parla*: sua configuração é 1 *core* de *2.40 GHz* para processamento, 1 GB de memória RAM e 8 GB de disco. Ele possui o sistema operacional *Ubuntu 14.04 LTS* (Canonical, 2016) e provê um serviço de mensagens instantâneas, baseado no protocolo *Extensible messaging and presence protocol* (XMPP). Esse serviço é utilizado para comunicação entre funcionários da empresa e do provedor, e também entre clientes e os funcionários. O *software* utilizado é o *Ejabberd 2.1.11* (ProcessOne, 2016), que também é um *software* de código aberto;
- *Passata2*: esse servidor possui 1 *core* de *2.40 GHz* para processamento, 2 GB de memória RAM e 20 GB de disco. Esse servidor possui o sistema operacional *Ubuntu 14.04 LTS* (Canonical, 2016) e fornece o serviço de DNS recursivo, através do *software Bind 9.9.5* (ISC, 2016). Esse é o servidor secundário de DNS do provedor;

- *Postfix*: sua configuração é 1 *core* de *2.40 GHz* para processamento, 768 MB de memória RAM e 50 GB de disco. Esse servidor possui o sistema operacional *Ubuntu 14.04 LTS* (Canonical, 2016) e é responsável pelo envio de *e-mails*, através do *software Postfix 2.11* (Postfix, 2016). Os *e-mails* enviados por esse servidor são gerados por uma ferramenta de *e-mail marketing*, que foi desenvolvida pela empresa. É esse servidor que faz o envio de *e-mails* em massa para divulgação de informações ou produtos;
- *Servo6*: sua configuração é 1 *core* de *2.40 GHz* para processamento, 1,5 GB de memória RAM e 30 GB de disco. Esse servidor possui o sistema operacional *CentOS 6.8* e fornece, através do *software Bind 9.8.2* (ISC, 2016), o serviço de DNS autoritativo. Essa máquina virtual é o servidor de DNS terciário dos domínios hospedados pela empresa.

4.3.4 Servidor Raggio

O servidor chamado Raggio executa doze VMs (Figura 4.7), que fornecem os serviços de virtualização para algumas empresas, sendo que as máquinas virtuais são instaladas com o sistema operacional de preferência do cliente, e são disponibilizadas através de um serviço de acesso remoto, como por exemplo, através de *Secure shell* (SSH).

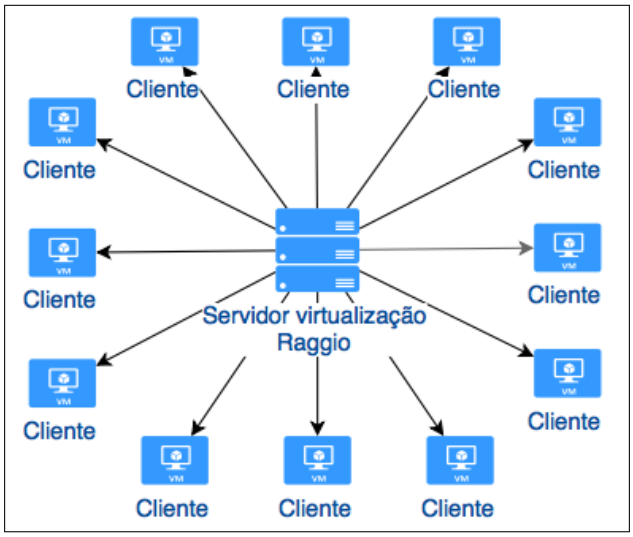


Figura 4.7: Servidor de virtualização Raggio.

4.3.5 Servidor Tempesta

O servidor Tempesta possui quatro VMs, como pode ser visto na Figura 4.8, sendo que os serviços executados nessas VMs são os seguintes:

- *Ledriovardar*: esse servidor possui 2 *cores* de *2.40 GHz* para processamento, 2 GB de memória RAM e 80 GB de disco. O servidor possui o sistema operacional *Windows 2008 Server R2* e possui o serviço de *terminal service* para suporte e gerência da rede do provedor;
- *Merak*: esse servidor fornece serviço de *e-mail* para envio e recebimento de *e-mails*. Ele possui uma configuração de 6 *cores* de *2.00 GHz* para processamento, 10 GB de memória RAM e 1 TB de disco. O servidor possui o

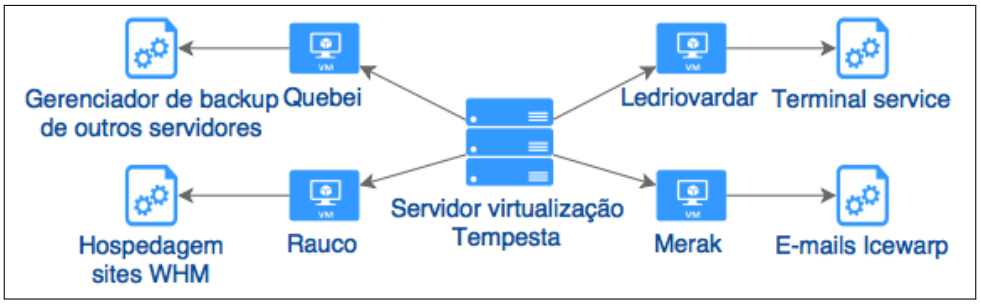


Figura 4.8: Servidor de virtualização Tempesta.

sistema operacional *Windows 2008 Server R2* e executa o *software Icewarp Server 10.4.4* (Icewarp, 2016). Essa aplicação fornece os serviços de envios de *e-mails* através do protocolo *Simple mail transfer protocol* (SMTP), recebimentos de *e-mails* através dos protocolos *Post office protocol* (POP) e *Internet message access protocol* (IMAP), o serviço de *Webmail* (PHP) e Anti-spam. Destaca-se que grande parte das contas de *e-mail* estão ociosas pois são oferecidas juntamente com a Internet vendida pelo provedor;

- *Quebei*: sua configuração é 1 *core* de *2.00 GHz* para processamento, 3 GB de memória RAM e 140 GB de disco. Esse servidor possui o sistema operacional *Ubuntu 14.04 LTS* (Canonical, 2016) e sua função é gerenciar o *backup* dos outros servidores. Para isso ele utiliza a ferramenta *Bacula 5.2.6* (Bacula, 2016) (pacote *bacula-director-common 5.2.6*). Destaca-se que esse *backup* é armazenado no servidor físico *Bello* detalhado na Seção 4.2. Além disso, esse servidor possui o sistema gerenciador de banco de dados *MySQL 5.5.49* (Oracle, 2016b) instalado, que está configurado no modelo *master-slave*, sendo que esse servidor é o *slave* e o servidor *Dati* (Seção 4.2) é o *master*;
- *Rauco*: esse servidor possui 2 *cores* de *2.00 GHz* para processamento, 6 GB de memória RAM e 600 GB de disco. Ele possui o sistema operacional *CentOS 6.8* (CentOS, 2016), sendo que esse servidor fornece o mesmo serviço do servidor *Roncon*, que foi descrito na Seção 4.3.2). Esse servidor fornece acesso a sites *Web* desenvolvidos com a linguagem PHP. Para suportar um número maior de hospedagens foram criados esses dois servidores, que possuem os mesmos *softwares*.

4.3.6 Servidor Tuono

O servidor Tuono possui quatro VMs, como pode ser visto na Figura 4.9, sendo que os serviços executados nessas VMs são os seguintes:

- *Mondoperso*: sua configuração é 1 *core* de *2.53 GHz* para processamento, 512 MB de memória RAM e 8 GB de disco. Esse servidor possui o sistema operacional *Ubuntu 14.04 LTS* (Canonical, 2016) e fornece *streaming* de áudio para *Web* rádio. Esse serviço é feito através do *software* livre *Icecast 2.3.3* (Xiph.Org Foundation, 2016);
- *Ns*: esse servidor possui 1 *core* de *2.53 GHz* para processamento, 2 GB de memória RAM e 30 GB de disco. O servidor possui o sistema operacional *CentOS 6.8* (CentOS, 2016) e fornece, através do *software Bind 9.9.3* (ISC, 2016), o serviço de DNS autoritativo. Esse é o servidor de DNS primário dos domínios hospedados pela empresa;

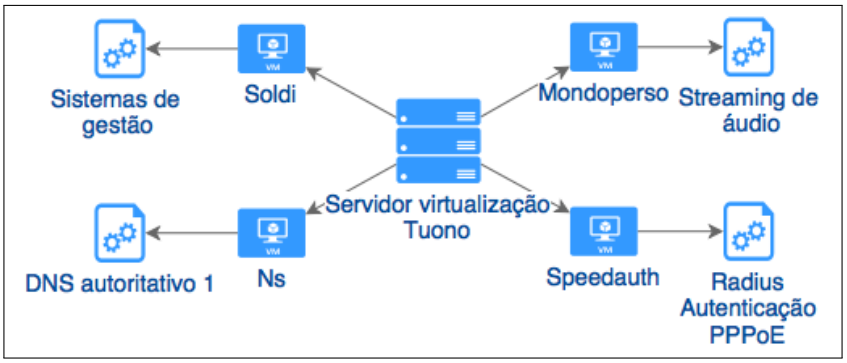


Figura 4.9: Servidor de virtualização Tuono.

- *Soldi*: sua configuração é 4 *cores* de *2.53 GHz* para processamento, 4 GB de memória RAM e 40 GB de disco. Esse servidor possui o sistema operacional *Ubuntu 14.04 LTS* (Canonical, 2016) e é um servidor *Web* exclusivo para *softwares* de gestão que são desenvolvidos pela empresa. Os seguintes *softwares* são utilizados: *Apache 2.4.7* (Apache Software Foundation, 2016a), *PHP 5.5.9* (PHP Group, 2016) (com *PHP-FPM 5.59*) e *MySQL 5.5.49* (Oracle, 2016b);
- *Speedauth*: sua configuração é 2 *cores* de *2.53 GHz* para processamento, 1,5 GB de memória RAM e 8 GB de disco. O sistema operacional é o *Ubuntu 14.04 LTS* (Canonical, 2016), sendo que esse servidor fornece o mesmo serviço do servidor *Masterauth* (Seção 4.3.1), que é autenticação PPPoE dos clientes do provedor. Esse servidor armazena os usuarios de autenticação da maior parte dos clientes.

4.3.7 Servidor Venti

O servidor Venti possui cinco VMs, como pode ser visto na Figura 4.10, sendo que os serviços executados nessas VMs são os seguintes:

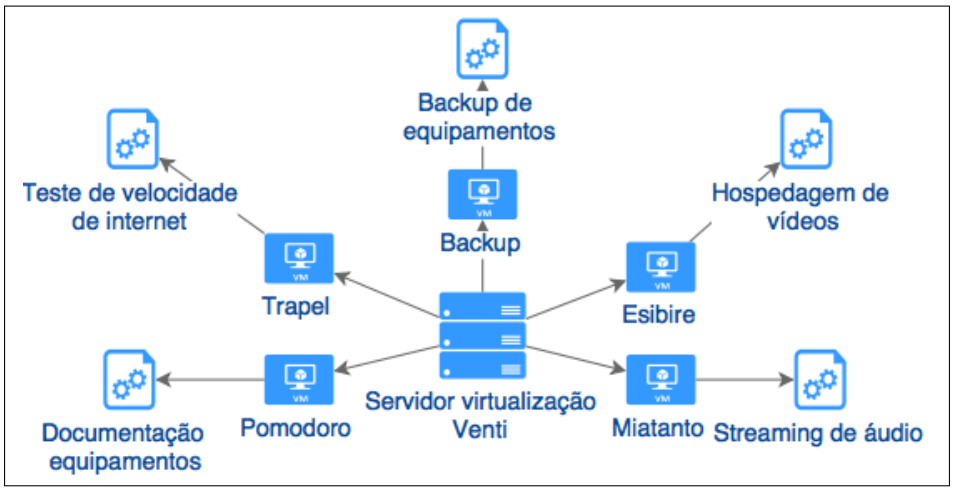


Figura 4.10: Servidor de virtualização Venti.

- *Backup*: sua configuração é 1 *core* de *3.10 GHz* para processamento, 1 GB de memória RAM e 15 GB de disco. Esse servidor possui o sistema operacional *Ubuntu 14.04 LTS* (Canonical, 2016) e executa o serviço de *backup* dos equipamentos do provedor. Esse utiliza *scripts* que foram desenvolvidos inter-

namente e que buscam e copiam os dados através do protocolo *File transfer protocol* (FTP);

- *Esibire*: sua configuração é 1 *core* de *3.10 GHz* para processamento, 1 GB de memória RAM e 50 GB de disco. Esse servidor possui o sistema operacional *Ubuntu 14.04 LTS* (Canonical, 2016) e faz a hospedagem de vídeos e a reprodução de *streaming* utilizando o protocolo FTP e um servidor *Web Apache 2.4.7*;
- *Miatanto*: sua configuração é 1 *core* de *3.10 GHz* de processamento, 1 GB de memória RAM e 8 GB de disco. Esse servidor possui o sistema operacional *Ubuntu 14.04 LTS* (Canonical, 2016) e fornece *streaming* de áudio para uma *Web* rádio. Esse serviço é feito através do *software* livre *Icecast 2.3.3* (Xiph.Org Foundation, 2016);
- *Pomodoro*: sua configuração é 1 *core* de *3.10 GHz* para processamento, 2 GB de memória RAM e 28 GB de disco. Esse servidor possui o sistema operacional *Ubuntu 14.04 LTS* (Canonical, 2016) e armazena a documentação dos equipamentos do provedor. Para esse armazenamento ele utiliza o *software* de código aberto *Sakai 2.9* (Apereo Foundation, 2016);
- *Trapel*: sua configuração é 1 *core* de *3.10 GHz* para processamento, 768 MB de memória RAM e 8 GB de disco. Esse servidor possui o sistema operacional *Ubuntu 14.04 LTS* (Canonical, 2016) e fornece um serviço de teste de velocidade da conexão à Internet. Ou seja, os usuários do provedor utilizam esse serviço para testar a velocidade da sua Internet. Para isso ele executa as aplicações *Apache 2.4.7* (Apache Software Foundation, 2016a) e *PHP 5.5.9* (PHP Group, 2016), e um *software* chamado *SpeedTest* (Ookla, 2016).

4.4 Considerações finais

Neste capítulo foi apresentado a empresa e feito uma análise de seus serviços. Com isso, no próximo capítulo será feito a identificação dos serviços mais críticos para a empresa e desenvolvido uma proposta para implementação de um ambiente de alta disponibilidade nos servidores de virtualização, além de identificar as ferramentas que serão utilizadas para essa implementação.

5 PROJETO DE IMPLEMENTAÇÃO

Neste capítulo será feito a identificação dos serviços mais críticos da empresa (Seção 5.1) para posteriormente ser possível propor uma solução de implementação de alta disponibilidade desses serviços que são fornecidos pela empresa, sendo que eles foram detalhados no Capítulo 4. Essa solução será detalhada na Seção 5.2. Com isso pretende-se atingir o objetivo deste trabalho.

5.1 Levantamento dos serviços críticos

No capítulo anterior foram detalhados todos os serviços que estão disponíveis na empresa. Sendo assim, nesta seção será feito uma análise desses serviços, destacando os serviços considerados mais críticos para a empresa.

O principal objetivo da empresa sempre foi priorizar a estabilidade e qualidade dos serviços fornecidos pela mesma. Deste modo, será definido alguns critérios para possibilitar a identificação dos serviços mais críticos para a empresa. Esses critérios são:

- A quantidade de clientes ou funcionários que utilizam o serviço: esse é o item mais relevante, pois impacta diretamente no faturamento da empresa. De fato, se um cliente ficar sem acesso à Internet, o cliente terá um desconto proporcional, na sua fatura, ao tempo que ficou sem acesso;
- O número de requisições em um determinado tempo: esse número é importante, uma vez que, indicam a quantidade de usuários que dependem do serviço. São exemplos dessa medida a quantidade de acessos por minuto em um servidor de hospedagens de sites e a quantidade de requisições DNS em um servidor recursivo;
- O volume de elementos do serviço: essa medida demonstra a abrangência do serviço, ou seja, quantos objetos existem em um serviço e consequentemente quantos clientes são dependentes deste. Como, por exemplo, a quantidade de contas de *e-mail* em um servidor de *e-mail* e a quantidade de equipamentos monitorados por um servidor de monitoramento. Esse critério é importante pois com ele pode-se comparar diferentes serviços possibilitando perceber o grau de relevância que cada um possui;
- O número de conexões *Transmission control protocol* (TCP) ou o número de transmissões *User datagram protocol* (UDP): essa medida pode demonstrar a quantidade de acessos simultâneos que um serviço pode atingir. Esse critério, como o anterior, também permite comparar diferentes serviços para ter a possibilidade de ordená-los de acordo com sua relevância, por isso o torna

importante para esta análise.

Considerando esses critérios, pode-se identificar os seguintes serviços:

- DNS recursivo primário: esse serviço foi classificado como o mais importante pois possui um impacto direto nos clientes do provedor. Esse é o único serviço que todos os clientes e funcionários utilizam. O objetivo de um provedor é fornecer uma navegação de qualidade aos seus clientes, sendo assim, o DNS é fundamental para essa navegação. O primeiro critério desse serviço é a quantidade de clientes que ele possui, que é aproximadamente 9000, além disso possui 65 funcionários. Outro importante critério que pode ser aplicado ao DNS recursivo é o número de requisições por segundo, que chega aproximadamente a 1150 (Figura 5.1). De acordo com a Figura 5.2, que compara os principais servidores da empresa através de transmissões UDP, onde pode-se claramente perceber que o servidor *Passata* possui um grande número de transmissões UDP, o qual fornece o serviço de DNS recursivo primário. Assim, esse serviço se torna o maior entre todos os outros;

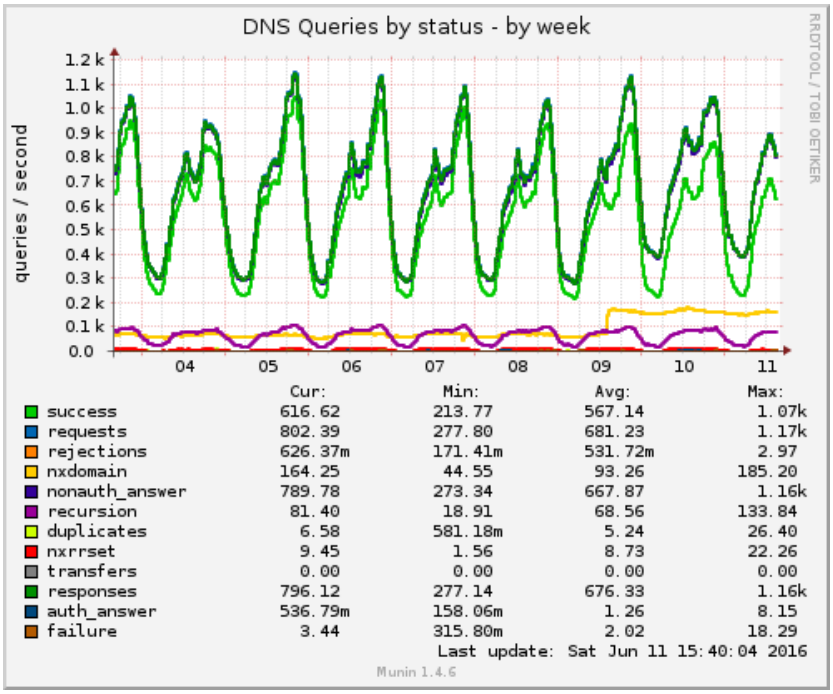


Figura 5.1: Gráfico de requisições DNS.

- *Radius*: esse serviço é importante para a navegação dos clientes do provedor, pois esse é o responsável pela autenticação de todos os clientes. Caso esse serviço fique indisponível, os clientes não conseguirão estabelecer conexão para sua navegação. Os servidores *Speedauth* e *Masterauth* recebem uma média de 1,6 requisições de autenticação por segundo como pode ser visto na Figura 5.3 e na 5.4, esse número se torna expressivo pois demonstra que a cada um segundo aproximadamente um cliente requisita a autenticação para iniciar a sua navegação. Além disso, esse serviço armazena alguns dados relacionados à conexão dos clientes, que são os seguintes dados: o endereço de IP que é utilizado por um cliente em um determinado período, o tráfego de dados da conexão, o tempo da conexão de cada cliente, o endereço *Media access control*

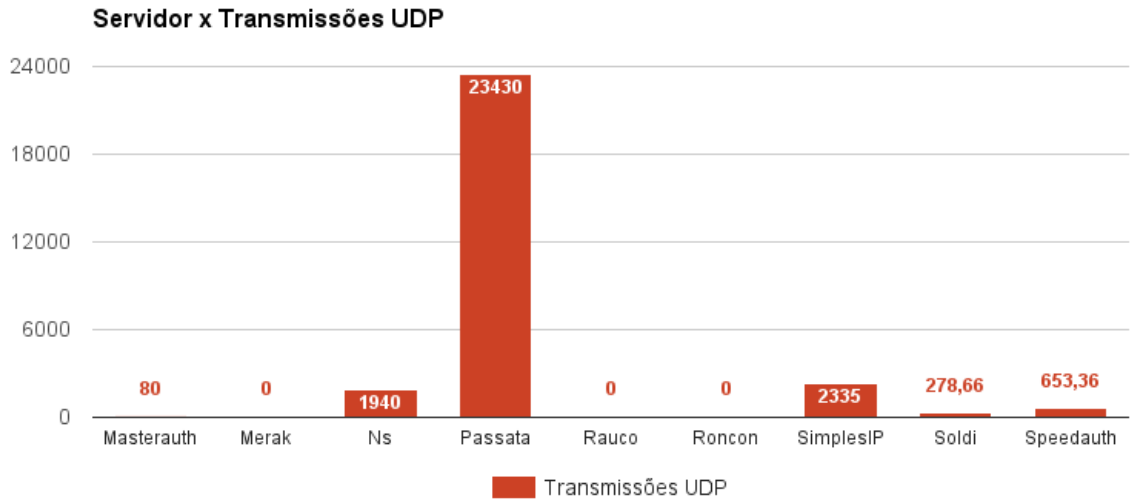


Figura 5.2: Gráfico de comparação de transmissões UDP entre os principais servidores.

(MAC) dos equipamentos dos clientes, entre outros. Essas operações resultam em um número de requisições por segundo (em média 23 requisições). Outro critério que é relevante para esses servidores é o volume de elementos do serviço, que neste caso, representa a quantidade de *logins* de autenticação dos clientes de acesso a Internet do provedor. De acordo com a Figura 5.5, os servidores *Speedauth* e *Masterauth* estão entre os maiores, sendo que essa figura compara os principais servidores da empresa através do volume de elementos que foi descrito no terceiro critério. Além disso, existe um destaque desses servidores no número de conexões TCP, como pode ser observado na Figura 5.6;

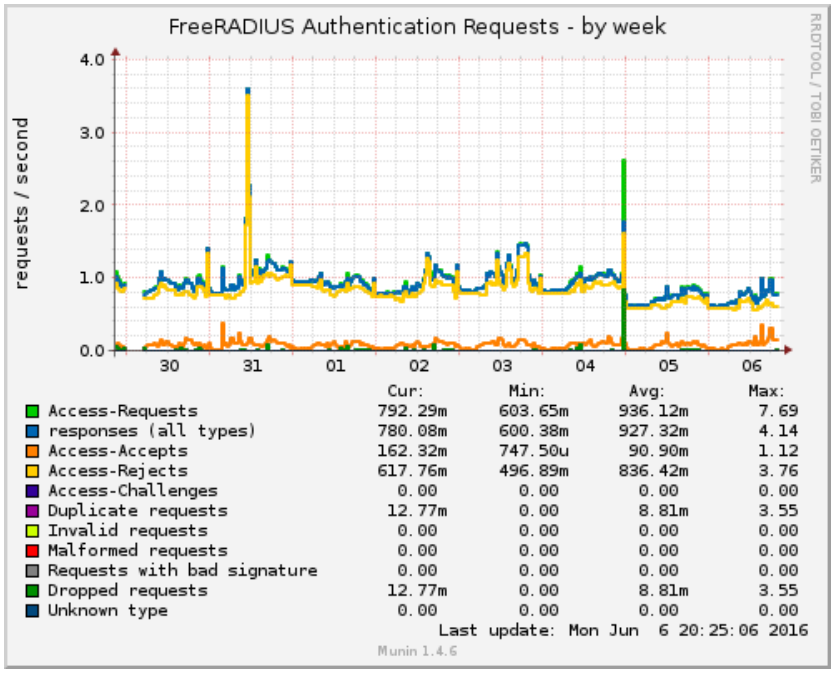


Figura 5.3: Gráfico de requisições de autenticação do servidor Speedauth.

- Sistemas da empresa e do provedor: o sistema do provedor é responsável pela maior parte das operações do provedor. De fato, esse sistema é responsável

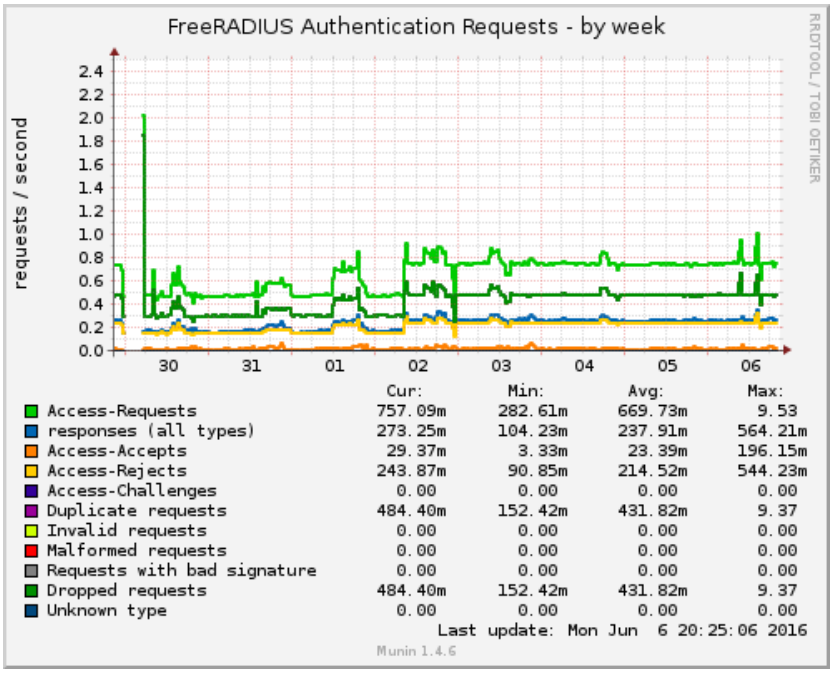


Figura 5.4: Gráfico de requisições de autenticação do servidor Masterauth.

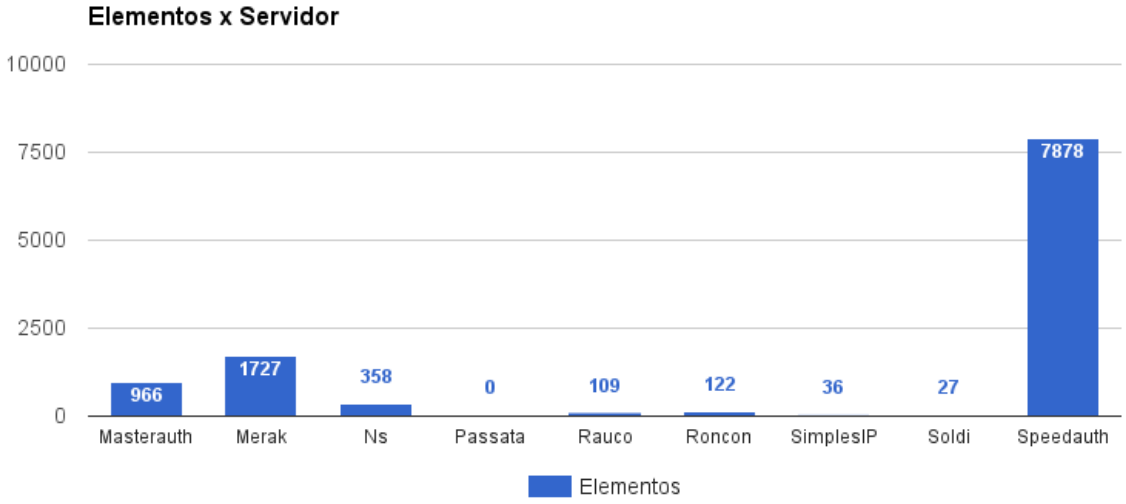


Figura 5.5: Gráfico de comparação de elementos entre os principais servidores.

pela emissão de boletos e envio para clientes, atendimento de clientes, comunicação interna da empresa, vendas, ativações de novos clientes, entre outros. Esse sistema não tem um impacto direto para os clientes, porém é fundamental para os funcionários da empresa e do provedor. Caso haja uma indisponibilidade desses sistemas a maior parte dos funcionários ficarão impossibilitados de trabalhar, sendo que a empresa possui no total 65 funcionários. Pode-se aplicar o critério de número de requisições por segundo nesse serviço, sendo que o principal sistema do provedor, que está executando sobre o servidor *Soldi*, que recebe aproximadamente 3 requisições por segundo (Figura 5.7). Além disso, a empresa mantém outros 27 sistemas, de outros clientes, nesse mesmo servidor. Outro critério que aplica-se a esse serviço é o número de conexões TCP, onde pode-se observar que o servidor *Soldi* está entre os mais relevantes, como pode

ser observado na Figura 5.6. Sendo que nesta figura há uma comparação entre os principais servidores da empresa através do critério de conexões TCP que foi descrito anteriormente no quarto critério;

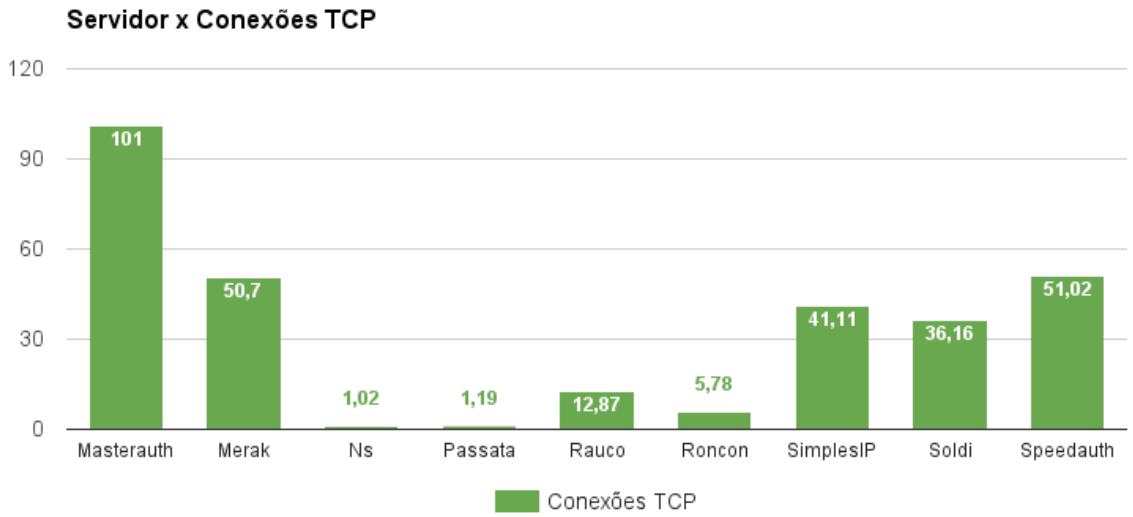


Figura 5.6: Gráfico de comparação de conexões TCP entre os principais servidores.

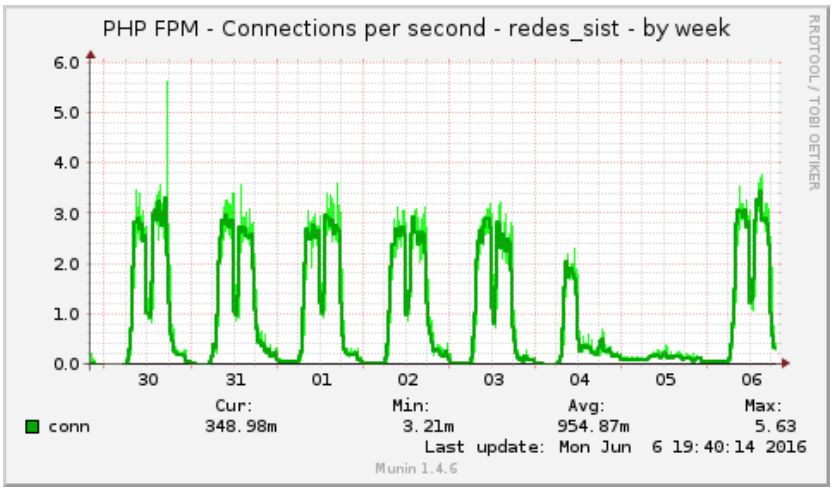


Figura 5.7: Gráfico de requisições por segundo do maior sistema.

- Telefonia: esse serviço tem relevância para a empresa e para o provedor, pois permite a comunicação entre os clientes e os funcionários, ou seja, o servidor *SimplesIP* é responsável por permitir o atendimento a clientes para fins de suporte técnico, comunicação interna entre funcionários, comunicação com técnicos externos, vendas, cobranças a clientes, entre outros. Para quantificar, no mês de maio de 2016 a empresa recebeu 15922 ligações, com duração total de 67 horas e 40 minutos, sendo que essas ligações foram recebidas por todos os setores do provedor e também da empresa. Além disso, no mesmo mês foram efetuadas 674 ligações entre funcionários. O gráfico da Figura 5.8 mostra a quantidade de canais ativos no servidor de telefonia. Observa-se na Figura que são realizadas de 20 a 30 ligações durante o horário comercial, das 08:00 às 12:00 e das 13:00 às 18:00. Outro critério que é relevante para este serviço é o número de conexões TCP e o número de transmissões UDP, pois

esse servidor também encontra-se entre os maiores. Ele pode ser observado na Figura 5.6 e na Figura 5.2.

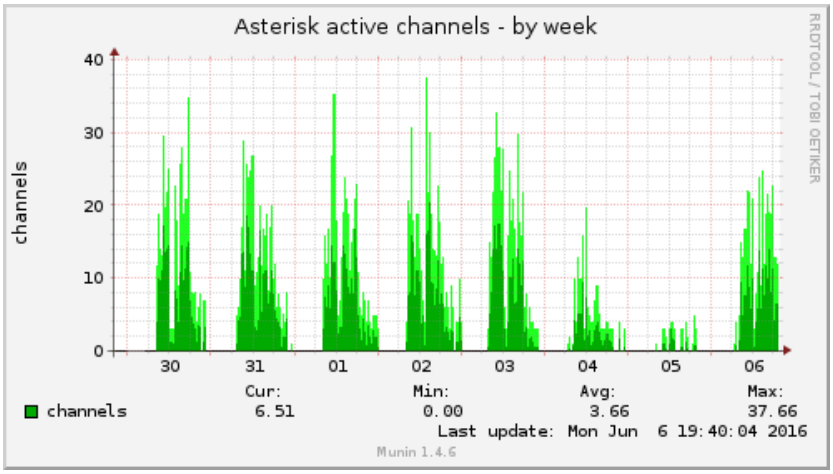


Figura 5.8: Gráfico de requisições por segundo do maior sistema.

Sobre os serviços descritos anteriormente, pode-se identificar quais máquinas virtuais deverão ser incluídas no ambiente de alta disponibilidade, sendo elas:

- *Passata*
- *Speedauth*
- *Masterauth*
- *Soldi*
- *SimplesIP*

5.2 Proposta de solução

O primeiro passo desta implementação é fazer uma reorganização das máquinas virtuais entre os servidores atuais, liberando assim o *hardware* suficiente para possibilitar a implementação do ambiente de alta disponibilidade. Para isso será feito as seguintes mudanças:

- *Passata* e *SimplesIP*: movidas do *Fulmine* para *Tuono*;
- *Monete*: movida do *Brina* para o *Piova*;
- *Ns*: movido do *Tuono* para o *Tempesta*;
- *Mondoperso*: movido do *Tuono* para o *Venti*.

Após ter sido feito a reorganização das máquinas virtuais será iniciado a implementação, montando um ambiente com dois servidores e os configurando de uma forma que caso houver alguma falha, em um servidor físico ou no seu sistema operacional hospedeiro, as máquinas serão tranferidas para o outro servidor físico. A configuração dos servidores deverá ser de 11 *cores* de processamento, 12 GB de memória e 156 GB de disco, sendo que essa configuração foi calculada a partir da soma dos recursos atuais das máquinas virtuais que possuem os serviços críticos, que foram apresentadas na Seção 5.1.

As ferramentas necessárias para essa implementação podem ser divididas em dois grupos: ferramenta de replicação de dados (Seção 5.2.1) e ferramenta que faz

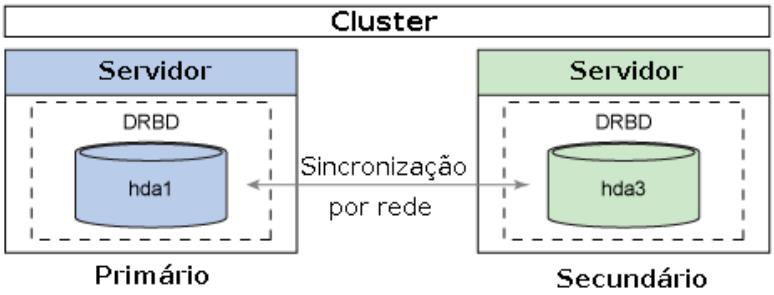
o monitoramento e a tranferências das máquinas virtuais no caso de alguma falha (Seção 5.2.2).

5.2.1 Ferramentas de replicação de dados

Replicação de dados pode ser feita de diversas maneiras, pode ser a nível de aplicação ou até mesmo a nível de *hardware*. Dependendo do objetivo e da aplicação pode-se usar ferramentas como por exemplo o *rsync*, que faz o sincronismo de dados de uma origem para um destino. Sendo assim, não é possível utilizar essa ferramenta, pois ela não faz a replicação em tempo real, ou seja, se for necessário utilizar os dados de destino ocorrerá perda de dados devido ao tempo de sincronismo. Outra forma de replicação é a de discos com RAID por exemplo, essa solução é eficaz para garantir que o sistema não fique indisponível em caso de falha de discos[1], porém não garante a disponibilidade quando *software* ou algum outro componente de *hardware* falhar (ZAMINHANI, 2008).

A solução de replicação ideal para esta implementação é um espelhamento de dados através da rede, sendo que essa solução permite a cópia dos dados para uma máquina remota em tempo real. Essa solução além de fazer a replicação dos dados, faz a redundância de todo *hardware*.

A ferramenta escolhida para replicação de dados na solução de alta disponibilidade desse trabalho foi o *Distributed replicated block device* (DRBD). Essa ferramenta é de código aberto, e permite a replicação de dados de um dispositivo local em tempo real. Os dispositivos DRBD, que geralmente ficam em */dev/drbd0*, possuem os estados primário e secundário. Entretanto, apenas um nó do *cluster* pode ser o nó primário, sendo que esse nó é o único que terá permissão para acessar o dispositivo, ou seja, somente ele poderá fazer operações de leitura e escrita. Na Figura 5.9 tem-se dois servidores com seus respectivos discos, *hda1* e *hda3*, formando um *cluster*, sendo que esses discos estão sincronizados através da rede. Com isso, todas as operações de escritas feitas no nó primário serão replicadas para o disco do nó secundário através do sincronismo por rede (ZAMINHANI, 2008).



Fonte: M. Tim Jones (2010)

Figura 5.9: Modelo DRBD

Neste projeto será utilizado o DRBD para replicar os discos das máquinas virtuais, para possibilitar a alta disponibilidade caso um dos servidores falhar.

[1]Lembrando que essa solução é utilizada no ambiente atual para aumentar a disponibilidade dos servidores

5.2.2 Ferramentas de gerenciamento de cluster

Para ser possível implementar uma solução de alta disponibilidade é necessário uma ferramenta que monitora os recursos, fazendo a detecção e recuperação do serviço utilizando mensagens entre os servidores, que é denominada como um *cluster* (PERKOV; PAVKOVIć; PETROVIć, 2011). Pode-se definir *cluster* como um grupo de computadores interligador por rede com o objetivo de aumentar o desempenho ou disponibilidade de um serviço (JUNIOR; FREITAS, 2005).

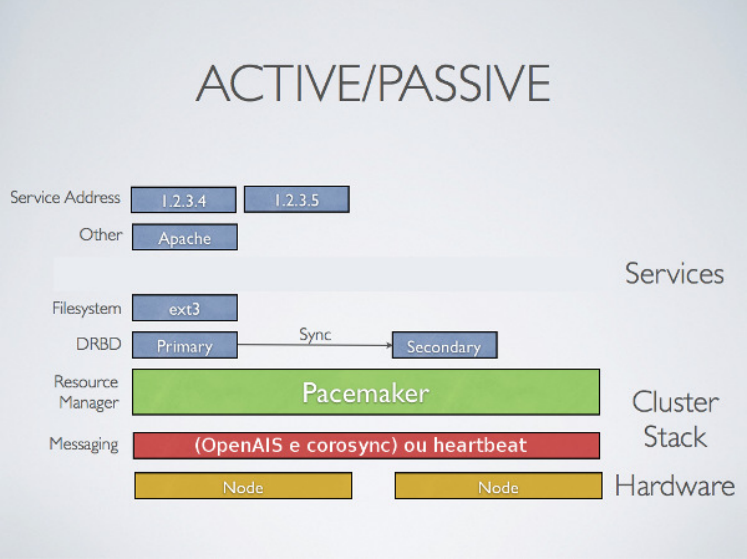
Essas ferramentas de gerenciamento de *cluster* permitem detecção de falhas a nível de nó ou a nível de recursos, ou seja, pode detectar falhas de *hardware* e falhas de serviços. Elas são conhecidas como *Cluster resource management* (CRM), cuja função é fazer a gerência de recursos de um *cluster*.

O CRM escolhido para gerenciar o ambiente que será criado neste trabalho foi o *Pacemaker*, pois foi a ferramenta de código aberto mais completa e flexível encontrada. Contudo, foi encontrada outra forma para implementar este ambiente, que seria utilizando apenas a ferramenta *Heartbeat*, porém, desta forma seria necessário criar um conjunto de *scripts* para fazer a migração das máquinas virtuais, o que dificultaria a implementação. O *Heartbeat* é um subprojeto do *Linux-HA* (Linux-HA, 2016a), que desenvolve aplicações para soluções de alta disponibilidade. Esse subprojeto é uma aplicação que envia pacotes *keepalive UDP* através da rede para outras instâncias do *Heartbeat*, para informar se ele está ativo (REIS, 2009).

O *Pacemaker* (ClusterLabs, 2016a) pode ser definido como uma ferramenta de recuperação de falhas a nível de serviço (PERKOV; PAVKOVIć; PETROVIć, 2011). Essa ferramenta é um projeto de código aberto mantido pela (ClusterLabs, 2016b), e teve origem com a necessidade de aperfeiçoar o gerenciador de recursos, que possuía algumas limitações, da ferramenta *Heartbeat* (Linux-HA, 2016b). Esse CRM funciona juntamente com ferramentas que fazem troca de mensagens entre os nós do *cluster* e ferramentas que fazem a afiliação, ou seja, fazem o processo de registro dos nós, de *failover* e de bloqueios distribuídos, quando utilizado sistema de arquivos distribuídos. As ferramentas encontradas para integrar com o *Pacemaker* são:

- *Corosync*: é responsável pelo processo de registro dos nós, de *failover* e de bloqueios distribuídos (utilizados para implementar cLVM, GFS/GFS2, e OCFS2), entre outros. Essa ferramenta derivou do projeto *OpenAIS*;
- *Heartbeat*: é um mecanismo que faz envio de mensagens entre os nós do *cluster* com objetivo de notificar esse *cluster* quando houver alguma alteração de estado dos recursos ou dos nós (ClusterLabs, 2016b).

Na Figura 5.10 pode-se observar a camada inferior com os nós do *cluster*, logo acima as ferramentas de envio de mensagens, acima delas fica o *Pacemaker*, e por fim o *DRBD* e os serviços dos servidor.



Fonte: ClusterLabs (2016a)

Figura 5.10: Pacemaker estrutura

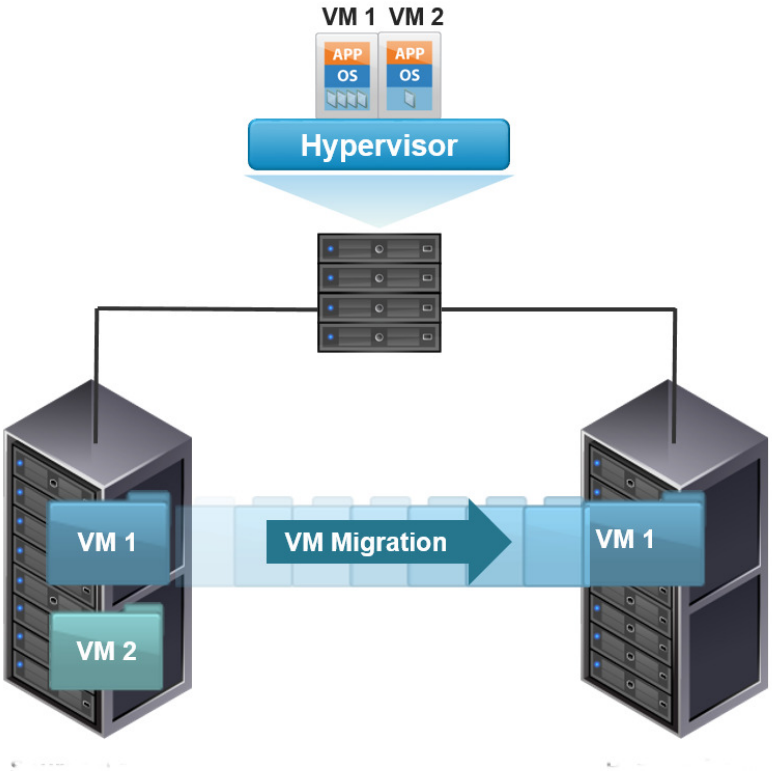
O CRM *Pacemaker* possui algumas funções principais que estão listadas abaixo:

- Inicia e para serviços dos nós do *cluster*, esses serviços podem ser desde um servidor *web* até uma interface de rede, por exemplo;
- Replica a configuração do *cluster* para todos os nós de forma transparente, com isso a configuração de todo o *cluster* pode ser feita em qualquer nó;
- Elege um nó como primário para centralizar todas as decisões, sendo que, se esse nó falhar outro será eleito primário.

Após ter os dados replicados, como foi descrito na seção anterior, deve-se criar o recurso do DRBD no *Pacemaker*, esse recurso fará o monitoramento e fará a troca dos estados, primário e secundário, dos nós do DRBD. Também será necessário criar um recurso para montar o sistema de arquivos do dispositivo DRBD no local onde ficarão os discos das máquinas virtuais. Esse recurso executará juntamente com o recurso do DRBD, assim, quando um nó do DRBD for escolhido como primário o sistema de arquivos será montado automaticamente.

E por fim, deve-se configurar um recurso para cada máquina virtual no *Pacemaker* para possibilitar que as máquinas virtuais migrem de um nó para outro de forma automática e transparente quando houver falhas. Para isso será necessário utilizar a opção de migração em tempo real, mais conhecida como *live migration*, que é fornecida pelo hipervisor KVM. A Figura 5.11 demonstra dois servidores com um hipervisor e duas VMs executando, sendo que elas inicialmente estão executando sobre o primeiro servidor, então a VM 1 está sendo migrada para o segundo servidor.

Neste capítulo, durante a escolha das ferramentas, foi efetuado alguns testes experimentais para melhor entender e avaliar as ferramentas e assim possibilitando escolha mais adequada para o ambiente da empresa. Sendo que esses testes foram apenas para entendimento.



Fonte: SPANIOL (2015)

Figura 5.11: Live migration

6 CONCLUSÃO

Neste trabalho foi feito um estudo de uma empresa prestadora de serviços para Internet, analisando sua estrutura física, bem como a configuração de todos os servidores, além do detalhamento de todos serviços fornecidos pela empresa. Após isso, foi feito o levantamento dos serviços mais críticos para o funcionamento da empresa com base em alguns critérios relevantes para a empresa. Esse estudo foi feito com objetivo de encontrar uma solução de alta disponibilidade para o ambiente atual da empresa, para aumentar a disponibilidade e a confiabilidade dos serviços vitais desta empresa.

Para atingir tal objetivo, estudou-se os conceitos básicos de alta disponibilidade, bem como os conceitos de tolerância a falhas, redundância de *hardware* e *software*. Como o ambiente atual da empresa possui grande parte dos serviços em máquinas virtuais, foi estudado os principais conceitos sobre virtualização. Sendo que na virtualização, foi estudado os dois grupos de máquinas virtuais o qual inclui máquinas virtuais de aplicação e máquinas virtuais de sistema, além das arquiteturas e estratégias de implementação de máquinas virtuais. Também foram descritas as vantagens do uso da virtualização em alguns cenários distintos.

E por fim, pesquisou-se ferramentas para possibilitar a implementação de um ambiente tolerante a falhas com o uso de máquinas virtuais. Sendo que, deu-se foco na busca por ferramentas de código aberto, para redução de custos. Para enfim propor uma solução de alta disponibilidade compatível com o ambiente atual da empresa, que é composto por servidores e ferramentas de código aberto, em sua maioria.

Sendo assim, pode-se concluir que é possível criar um *cluster* de alta disponibilidade com ferramentas de código aberto, e além disso pode-se utilizar virtualização em conjunto com essas ferramentas, para que, deste modo o ambiente se torne mais flexível e disponível.

Entretanto, como neste trabalho ainda será feito a implementação em si, não pode-se afirmar que a solução irá funcionar corretamente na empresa, devido a fatores como versão do sistema operacional, versão dos *softwares* de virtualização, entre outros.

6.1 Cronograma

A implementação fará parte da segunda etapa deste trabalho, bem como a elaboração da solução definitiva para o ambiente de alta disponibilidade. Além disso, as tarefas da segunda etapa deste trabalho estão detalhadas na Tabela 6.1. Para uma melhor organização, foi criado um cronograma com a distribuição das tarefas ao longo do próximo semestre, esse cronograma está presente na Tabela 6.2.

	Descrição
1	Redação do TCC II
2	Elaboração da solução
3	Reorganização do ambiente de virtualização
4	Realização de testes
5	Implementação da solução
6	Análise dos resultados e medições
7	Apresentação do TCC II

Tabela 6.1: Tarefas do trabalho de conclusão II

	Jul	Ago	Set	Out	Nov	Dez
1	X	X	X	X	X	X
2	X	X	X			
3	X	X				
4		X	X	X		
5			X	X		
6				X	X	
7						X

Tabela 6.2: Cronograma do trabalho de conclusão II

REFERÊNCIAS

ANDRADE, L. **Visão geral sobre virtualização**. <Disponível em: https://tecnologiasemsegredos.wordpress.com/category/virtualizacao/>. Acesso em 21 de maio de 2016.

Apache Software Foundation. **The Apache HTTP Server Project**. <Disponível em: http://httpd.apache.org/>. Acesso em 05 de junho de 2016.

Apache Software Foundation. **Apache Subversion**. <Disponível em: https://subversion.apache.org/>. Acesso em 05 de junho de 2016.

Apereo Foundation. **Introducing Sakai 11**. <Disponível em: https://sakaiproject.org/>. Acesso em 05 de junho de 2016.

Bacula. **Open Source Backup, Enterprise ready, Network Backup Tool for Linux, Unix, Mac, and Windows**. <Disponível em: http://blog.bacula.org/>. Acesso em 05 de junho de 2016.

BATISTA, A. C. **Estudo teórico sobre cluster linux**. 2007. Pós-Graduação (Administração em Redes Linux) — Universidade Federal de Lavras, Minas Gerais.

BUYYA, R.; VECCHIOLA, C.; SELVI, S. **Mastering Cloud Computing**: foundations and applications programming. [S.l.]: Elsevier, 2013.

Cacti. **Cacti - The Complete RRDTool-based Graphing Solution**. <Disponível em: http://www.cacti.net/>. Acesso em 05 de junho de 2016.

Canonical. **Ubuntu Server - for scale out workloads**. <Disponível em: http://www.ubuntu.com/server>. Acesso em 05 de junho de 2016.

CARISSIMI, A. Virtualização: da teoria a soluções. In: **Minicursos do Simpósio Brasileiro de Redes de Computadores**. Porto Alegre: XXVI Simpósio Brasileiro de Redes de Computadores e Sistemas Distribuídos, 2008.

CentOS. **The CentOS Project**. <Disponível em: https://www.centos.org/>. Acesso em 05 de junho de 2016.

Citrix. **The Xen Project, the powerful open source industry standard for virtualization**. <Disponível em: http://www.xenproject.org/>. Acesso em 22 de maio de 2016.

Citrix. **XenApp and XenDesktop - Virtual Apps and Desktops**. <Disponível em: https://www.citrix.com/products/xenapp-xendesktop/>. Acesso em 22 de maio de 2016.

ClusterLabs. **Pacemaker - ClusterLabs**. <Disponível em: http://clusterlabs.org/wiki/Pacemaker>. Acesso em 08 de junho de 2016.

ClusterLabs. **Cluster Labs - The Home of Linux Clustering**. <Disponível em: http://clusterlabs.org/>. Acesso em 08 de junho de 2016.

COSTA, H. L. A. **Alta disponibilidade e balanceamento de carga para melhoria de sistemas computacionais críticos usando software livre**: um estudo de caso. 2009. Pós-Graduação em Ciência da Computação — Universidade Federal de Viçosa, Minas Gerais.

cPanel and WHM. **The Hosting Platform of Choice**. <Disponível em: https://cpanel.com/>. Acesso em 05 de junho de 2016.

Digium. **Asterisk.org**. <Disponível em: http://www.asterisk.org/>. Acesso em 05 de junho de 2016.

Edgewall Software. **The Trac Project**. <Disponível em: https://trac.edgewall.org/>. Acesso em 05 de junho de 2016.

FreeRADIUS. **FreeRADIUS**: the world's most popular radius server. <Disponível em: http://freeradius.org/>. Acesso em 05 de junho de 2016.

GONçALVES, E. M. **Implementação de Alta disponibilidade em máquinas virtuais utilizando Software Livre**. 2009. Trabalho de Conclusão (Curso de Engenharia da Computação) — Faculdade de Tecnologia e Ciências Sociais Aplicadas, Brasilia.

IBM. **System/370 Model 145**. <Disponível em: https://www-03.ibm.com/ibm/history/exhibits/mainframe/mainframe_PP3145.html>. Acesso em 22 de maio de 2016.

Icewarp. **IceWarp Server para Windows e Linux**. <Disponível em: https://www.icewarp.com.br/>. Acesso em 05 de junho de 2016.

ISC. **Internet Systems Consortium**. <Disponível em: https://www.isc.org/downloads/>. Acesso em 05 de junho de 2016.

JUNIOR, E. P. F.; FREITAS, R. B. de. **Construindo Supercomputadores com Linux - Cluster Beowulf**. 2005. Trabalho de Conclusão (Curso de Redes de Comunicação) — Centro Federal de Educação Tecnológica de Goiás, Goiânia - Goiás.

LAUREANO, M. A. P.; MAZIERO, C. A. Virtualização: conceitos e aplicações em segurança. In: **Simpósio Brasileiro em Segurança da Informação e de Sistemas Computacionais**. Gramado - Rio Grande do Sul: VIII Simpósio Brasileiro em Segurança da Informação e de Sistemas Computacionais, 2008.

Linux-HA. **Linux-HA**. <Disponível em: http://www.linux-ha.org/wiki/Main_Page>. Acesso em 08 de junho de 2016.

Linux-HA. **Heartbeat - Linux-HA**. <Disponível em: http://www.linux-ha.org/wiki/Heartbeat>. Acesso em 08 de junho de 2016.

M. Tim Jones. **High availability with the Distributed Replicated Block Device**. <Disponível em: http://www.ibm.com/developerworks/library/l-drbd/>. Acesso em 12 de junho de 2016.

MARINESCU, D. **Cloud Computing**: theory and practice. [S.l.]: Elsevier Science, 2013.

MAZIERO, C. A. **Sistemas Operacionais**: conceitos e mecanismos. 2013. Dissertação (Mestrado em Ciência da Computação) — DAInf UTFPR, Paraná.

Microsoft. **ASP.NET**. <Disponível em: http://www.asp.net/>. Acesso em 05 de junho de 2016.

Microsoft. **Home**: the official microsoft iis site. <Disponível em: http://www.iis.net/>. Acesso em 05 de junho de 2016.

MOREIRA, D. **Virtualização**: rode vários sistemas operacionais na mesma máquina. <Disponível em: http://idgnow.com.br/ti-corporativa/2006/08/01/idgnoticia.2006-07-31.7918579158/#&panel1-3>. Acesso em 5 de abril de 2016.

Munin. **Munin**. <Disponível em: http://munin-monitoring.org/>. Acesso em 05 de junho de 2016.

Nagios. **Nagios - The Industry Standard In IT Infrastructure Monitoring**. <Disponível em: https://www.nagios.org/>. Acesso em 05 de junho de 2016.

NøRVåG, K. **An Introduction to Fault-Tolerant Systems**. 2000. IDI Technical Report 6/99 — Norwegian University of Science and Technology, Trondheim, Noruega.

Ookla. **Speedtest.net by Ookla - The Global Broadband Speed Test**. <Disponível em: https://www.speedtest.net/>. Acesso em 28 de maio de 2016.

Oracle. **Oracle VM VirtualBox**. <Disponível em: https://www.virtualbox.org/>. Acesso em 22 de maio de 2016.

Oracle. **MySQL**. <Disponível em: https://www.mysql.com/>. Acesso em 05 de junho de 2016.

PANKAJ, J. **Fault tolerance in distributed system**. Nova Jérsei, Estados Unidos: P T R Prentice Hall, 1994.

PEREIRA FILHO, N. A. **Serviço de pertinência para clusters de alta disponibilidade**. 2004. Dissertação para Mestrado em Ciência da Computação — Universidade de São Paulo, São Paulo.

PERKOV, L.; PAVKOVIć, N.; PETROVIć, J. High-Availability Using Open Source Software. In: MIPRO, 2011 PROCEEDINGS OF THE 34TH INTERNATIONAL CONVENTION. **Anais...** [S.l.: s.n.], 2011. p.167–170.

PHP Group. **PHP**: hypertext preprocessor. <Disponível em: http://php.net/>. Acesso em 05 de junho de 2016.

Postfix. **The Postfix Home Page**. <Disponível em: http://www.postfix.org/>. Acesso em 05 de junho de 2016.

PostgreSQL Group. **PostgreSQL**: the world's most advanced open source database. <Disponível em: https://www.postgresql.org/>. Acesso em 05 de junho de 2016.

ProcessOne. **ejabberd — robust, massively scalable and extensible XMPP server**. <Disponível em: https://www.ejabberd.im/>. Acesso em 05 de junho de 2016.

QEmu. **QEMU open source processor emulator**. <Disponível em: http://wiki.qemu.org/Main_Page>. Acesso em 22 de maio de 2016.

Red Hat Inc. **Plataforma corporativa Linux**. <Disponível em: https://www.redhat.com/pt-br/technologies/linux-platforms>. Acesso em 05 de junho de 2016.

REIS, W. S. dos. **Virtualização de serviços baseado em contêineres**: uma proposta para alta disponibilidade de serviços em redes linux de pequeno porte. 2009. Monografia Pós-Graduação (Administração em Redes Linux) — Apresentada ao Departamento de Ciência da Computação, Minas Gerais.

ROUSE, M. **Hot spare**. <Disponível em: http://searchstorage.techtarget.com/definition/hot-spare>. Acesso em 12 de abril de 2016.

Samba Team. **Samba - Opening Windows to a Wider World**. <Disponível em: https://www.samba.org/>. Acesso em 05 de junho de 2016.

SANTOS MACEDO, A. dos; SANTOS, C. C. G. **Hypervisor**: segurança em ambientes virtualizados. <Disponível em: http://www.devmedia.com.br/hypervisor-seguranca-em-ambientes-virtualizados/30993>. Acesso em 05 de junho de 2016.

SILVA VIANA, A. L. da. **MySQL**: replicação de dados. <Disponível em: http://www.devmedia.com.br/mysql-replicacao-de-dados/22923>. Acesso em 21 de abril de 2016.

SILVA, Y. F. da. **Uma Avaliação sobre a Viabilidade do Uso de Técnicas de Virtualização em Ambientes de Alto Desempenho**. 2009. Trabalho de Conclusão (Curso de Ciência da Computação) — Universidade de Caxias do Sul, Caxias do Sul - Rio Grande do Sul.

SMITH, J. E.; NAIR, R. The architecture of virtual machines. **IEEE Computer**, [S.l.], v.38, p.32–38, 2005.

SMITH, R. **Gerenciamento de Nível de Serviço**. <Disponível em: http://blogs.technet.com/b/ronaldosjr/archive/2010/05/25/gerenciamento-de-n-237-vel-de-servi-231-o.aspx/>. Acesso em 25 de março de 2016.

SPANIOL, B. **Quatro estratégias de proteção para seu ambiente virtual**. <Disponível em: http://www.aliancatecnologia.com/conteudo/2015/05/quatro-estrategias-de-protecao-para-seu-ambiente-virtual/>. Acesso em 02 de junho de 2016.

VMware. **VMware ESXi**. <Disponível em: http://www.vmware.com/products/esxi-and-esx/overview.html>. Acesso em 22 de maio de 2016.

VMware. **VMware Workstation Player (formerly known as Player Pro)**. <Disponível em: https://www.vmware.com/products/player>. Acesso em 22 de maio de 2016.

VMware. **VDI Virtual Desktop Infrastructure with Horizon**. <Disponível em: https://www.vmware.com/products/horizon-view>. Acesso em 22 de maio de 2016.

WEBER, T. S. **Um roteiro para exploração dos conceitos básicos de tolerância a falhas**. 2002. Curso de Especialização em Redes e Sistemas Distribuídos — UFRGS, Rio Grande do Sul.

WineHQ. **WineHQ - Execute aplicativos Windows no Linux, BSD, Solaris e Mac OS X**. <Disponível em: https://www.winehq.org/>. Acesso em 22 de maio de 2016.

WMware. **VMware Workstation Pro**. <Disponível em: http://www.vmware.com/br/products/workstation>. Acesso em 17 de maio de 2016.

Xiph.Org Foundation. **Icecast**. <Disponível em: http://icecast.org/>. Acesso em 05 de junho de 2016.

ZAMINHANI, D. **Cluster de alta disponibilidade através de espelhamento de dados em máquinas remotas**. 2008. Pós-Graduação apresentada ao Departamento de Ciência da Computação — Universidade Federal de Lavras, Minas Gerais.

ZoneMinder. **ZoneMinder**. <Disponível em: https://zoneminder.com/>. Acesso em 05 de junho de 2016.