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
Novembro 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
1 de julho de 2016

Daniel Luís Notari
Coordenador

AGRADECIMENTOS

Primeiramente agradeço aos meus pais, Lourdes e Irineu, pela educação e ensino que me transmitiram, sendo que muitos desses ensinos levarei para toda minha vida e além disso me tornaram a pessoa que sou hoje. Agradeço ao grande coração de minha mãe e pelo grande conhecimento de vida de meu pai.

Um agradecimento especial para minha namorada Franciele Damin, que me ajudou e apoiou durante este trabalho. Mulher a qual compartilharei sonhos e realizações durante toda minha vida.

Agradeço meu orientador André Luis Martinotto que me auxiliou neste trabalho de conclusão. E também a todos os professores da UCS pelos conhecimentos que compartilharam comigo.

Agradeço aos amigos que conheci ao longo de minha vida acadêmica e profissional pela troca de conhecimento.

SUMÁRIO

LISTA DE SIGLAS 5

LISTA DE FIGURAS 7

LISTA DE TABELAS 8

RESUMO 9

ABSTRACT 10

1 INTRODUÇÃO 11
1.1 Objetivos 12
1.2 Estrutura do trabalho 12

2 ALTA DISPONIBILIDADE 13
2.1 Tolerância a falhas 13
2.2 Redundância 15
2.3 Cálculo da alta disponibilidade 16
2.4 Considerações finais 17

3 VIRTUALIZAÇÃO 18
3.1 Máquinas virtuais de aplicação 20
3.2 Máquinas virtuais de sistema 21
3.2.1 Arquiteturas de máquinas virtuais de sistema 22
3.2.2 Implementações de máquinas virtuais de sistema 23
3.3 Vantagens das máquinas virtuais 24
3.3.1 Virtualização de Desktop 24
3.3.2 Virtualização de servidores 25
3.4 Considerações finais 26

4 INFRAESTRUTURA DA EMPRESA 27
4.1 Instalação física 28
4.2 Servidores sem virtualização 30
4.3 Servidores com virtualização 31
4.3.1 Servidor Brina 31
4.3.2 Servidor Fulmine 32
4.3.3 Servidor Piova 33
4.3.4 Servidor Raggio 35
4.3.5 Servidor Tempesta 35

4.3.6 Servidor Tuono . . . 36
4.3.7 Servidor Venti . . . 37
4.4 Considerações finais . . . 38

5 SERVIÇOS CRíTICOS . . . 39
5.1 DNS recursivo primário . . . 39
5.2 Autenticação Radius . . . 40
5.3 Sistemas da empresa e do provedor . . . 41
5.4 Telefonia interna sobre IP . . . 42
5.5 Resumo e os serviços críticos . . . 43
5.6 Considerações finais . . . 43

6 SOFTWARES PARA IMPLEMENTAÇÃO . . . 44
6.1 Softwares para a replicação de dados . . . 45
6.1.1 DRBD (*Distributed Replicated Block Device*) . . . 45
6.1.2 GlusterFS . . . 46
6.1.3 Rsync . . . 47
6.1.4 Resumo e software de replicação adotado . . . 48
6.2 Softwares para o gerenciamento do cluster . . . 48
6.2.1 Ganeti . . . 48
6.2.2 Heartbeat . . . 49
6.2.3 Pacemaker . . . 49
6.2.4 Resumo e software de gerenciamento adotado . . . 51
6.3 Considerações finais . . . 51

7 IMPLEMENTAÇÃO E TESTES . . . 53
7.1 Implementação . . . 53
7.2 Testes . . . 55
7.2.1 Teste 1 - Desligamento físico . . . 55
7.2.2 Teste 2 - Desligamento por software . . . 55
7.2.3 Teste 3 - Manutenção agendada . . . 56
7.3 Comparação final da disponibilidade . . . 58
7.4 Considerações finais . . . 58

8 CONCLUSÃO . . . 59
8.1 Trabalho futuros . . . 60

REFERÊNCIAS . . . 61

APÊNDICE A . . . 67
A.1 Configuração do *Operating System* (OS) . . . 67
A.2 Configuração de rede . . . 67
A.3 Configuração de disco . . . 68
A.4 Configuração do ambiente virtualizado . . . 70
A.5 Configuração do cluster Pacemaker . . . 70

APÊNDICE B . . . 73
B.1 Script indisponibilidade . . . 73
B.2 Script manutenção Pacemaker . . . 73

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*
GFS	*Global File System*
HTTP	*Hypertext 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*
LVM	*Logical Volume Manager*
MAC	*Media Access Control*
MTBF	*Mean Time Between Failures*
MTTR	*Mean Time To Repair*
NAT	*Network Address Translation*
OCFS2	*Oracle Cluster File System 2*
OS	*Operating System*
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*
UTP	*Unshielded Twisted Pair*
VM	*Virtual Machine*
VMM	*Virtual Machine Monitor*
WHM	*WebHost Manager*
XMPP	*EXtensible Messaging and Presence Protocol*

LISTA DE FIGURAS

Figura 3.1: Interfaces de sistemas de computação. 19
Figura 3.2: Máquinas virtuais de aplicação e de sistema. 20
Figura 3.3: Componentes da virtualização. 21
Figura 3.4: Arquiteturas de máquinas virtuais de sistema. 22
Figura 3.5: Implementações de máquinas virtuais de sistema. 23

Figura 4.1: Diagrama de instalação elétrica. 27
Figura 4.2: Modelo de estrutura física. 29
Figura 4.3: Imagem do *rack* e dos servidores. 29
Figura 4.4: Servidor de virtualização *Brina*. 32
Figura 4.5: Servidor de virtualização *Fulmine*. 32
Figura 4.6: Servidor de virtualização *Piova*. 34
Figura 4.7: Servidor de virtualização *Raggio*. 35
Figura 4.8: Servidor de virtualização *Tempesta*. 36
Figura 4.9: Servidor de virtualização *Tuono*. 37
Figura 4.10: Servidor de virtualização *Venti*. 38

Figura 5.1: Gráfico de requisições DNS por segundo do período de uma semana (a) e comparação de requisições UDP simultâneas máximas entre os principais servidores (b). 40
Figura 5.2: Gráfico de comparação de elementos (a) e de conexões TCP simultâneas máximas (b) entre os principais servidores. 41
Figura 5.3: Gráfico de requisições por segundo do sistema do provedor durante o período de uma semana. 42
Figura 5.4: Gráfico da quantidade de canais ativos simultâneamente no servidor de telefonia durante o período de uma semana. 42

Figura 6.1: Exemplo do modelo *master-slave* do DRBD. 45
Figura 6.2: Modelo do *GlusterFS*. 46
Figura 6.3: Transferência de arquivos através do *Rsync*. 47
Figura 6.4: Arquitetura do *Ganeti*. 49
Figura 6.5: Arquitetura do *Heartbeat*. 50
Figura 6.6: Exemplo da arquitetura do *Pacemaker*. 50

Figura 7.1: Estrutura física. 54
Figura 7.2: Estrutura do *cluster*. 54
Figura 7.3: Latência da máquina virtual durante o *live migration*. 57

LISTA DE TABELAS

Tabela 2.1: Níveis de alta disponibilidade e exemplos de sistemas. 16

Tabela 4.1: Configuração dos servidores físicos. 28

Tabela 5.1: Serviços críticos do ano de 2015. 43

Tabela 6.1: Comparação ferramentas de replicação de dados. 48
Tabela 6.2: Comparação entre ferramentas de gerenciamento de *cluster*. . . . 51

Tabela 7.1: Resultados do teste de desligamento físico, contendo o tempo de indisponibilidade em segundos e o desvio padrão. 56
Tabela 7.2: Resultados do teste de desligamento por *software*, com tempo de indisponibilidade em segundos e o desvio padrão. 56
Tabela 7.3: Resultados do teste de manutenção agendada, com o tempo de indisponibilidade em segundos e o desvio padrão. 57
Tabela 7.4: Disponibilidade e tempo de indisponibilidade do período de 13 dias. 57

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 este tipo de serviço. De fato, o aumento da disponibilidade é um grande diferencial para essas empresas.

Dentro deste contexto, neste trabalho é feito um estudo para a implementação de um ambiente de alta disponibilidade em uma empresa prestadora de serviços para Internet, utilizando ferramentas de código aberto. Para isso, foi feita uma análise dos serviços oferecidos pela empresa, bem como um levantamento do ambiente de servidores da mesma.

Criou-se um projeto de implementação para a solução de alta disponibilidade no ambiente da empresa, esse ambiente utiliza virtualização para uma grande parte dos serviços. Para tal, pesquisou-se algumas ferramentas de gerenciamento de *cluster* e de replicação de dados. A partir destas ferramentas é feita uma análise das mesmas e adotada a mais adequada para o ambiente da empresa. Deste modo, o *cluster* é composto pelo *software Pacemaker*, que faz o gerenciamento do *cluster* e pelo *software DRBD*, que é responsável pela replicação dos dados.

Por fim validou-se o ambiente de alta disponibilidade através de testes que simulam falhas de *hardware*, de energia elétrica e de *software*.

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

ABSTRACT

Keywords: high availability, virtualization, fault tolerance, open source.

1 INTRODUÇÃO

O crescente avanço tecnológico e o desenvolvimento da Internet, provocou um aumento significativo no número de aplicações ou serviços que dependem da infra-estrutura de Tecnologia da Informação (TI). Além disso, percebe-se um aumento no número de operações *on-line* que são realizadas, 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, *websites*, 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 foi realizado um estudo sobre a implementação de um sistema de alta disponibilidade em uma empresa prestadora de serviços para Internet. Essa empresa oferece serviços, 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.

Anteriormente, a empresa possuía somente redundância nas conexões de acesso à Internet, refrigeração e energia, com *nobreaks* e geradores. Porém, essa empresa não possuía nenhuma redundância nos serviços que estavam sendo executados nos servidores. Desta forma, caso ocorresse uma falha de *software* ou de *hardware*, os serviços ficariam indisponíveis. Neste trabalho foi realizada uma análise dos serviços oferecidos pela empresa, sendo que mecanismos de alta disponibilidade foram desenvolvidos para os serviços mais críticos. Para a redução dos custos foram utilizadas ferramentas gratuitas e de código aberto.

1.1 Objetivos

A empresa estudada não apresentava nenhuma solução de alta disponibilidade para seus serviços críticos. Desta forma, neste trabalho foi desenvolvida uma solução de alta disponibilidade para estes serviços, sendo que essa solução foi baseada no uso de ferramentas de código aberto e de baixo custo. Para que o objetivo geral fosse atendido os seguintes objetivos específicos foram realizados:

- Identificação dos serviços críticos a serem integrados ao ambiente de alta disponibilidade;
- Definição das ferramentas a serem utilizadas para implementar tolerância a falhas;
- Realização de testes para a validação do sistema de alta disponibilidade que foi desenvolvido.

1.2 Estrutura do trabalho

O trabalho foi estruturado em seis 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 as estratégias utilizadas para a implementação das mesmas;
- Capítulo 4: descreve o ambiente inicial da empresa e os serviços que são fornecidos por esta;
- Capítulo 5: neste capítulo são definidos os critérios para selecionar os serviços críticos da empresa;
- Capítulo 6: neste capítulo é apresentado as ferramentas que compõem o ambiente de alta disponibilidade;
- Capítulo 7: este capítulo apresenta a solução de alta disponibilidade, sendo que esta foi desenvolvida baseada na utilização de virtualização. Além disso, são descritos os testes realizados e feita a análise dos resultados proveniente dos testes no ambiente de alta disponibilidade;
- Capítulo 8: apresenta as conclusões do trabalho e as sugestões de trabalhos futuros.

2 ALTA DISPONIBILIDADE

Alta disponibilidade é uma técnica conhecida 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 à 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 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 utilizam-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 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 probabilidade 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 falha sem afetar o funcionamento do sistema (PANKAJ, 1994).

A tolerância a falhas frequentemente é implementada utilizando redundância, que será apresentada na Seção 2.2. Pode-se utilizar virtualização para implementar uma redundância e, consequentemente, tornar um sistema tolerante a falhas. 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 a detecção e a 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 está 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 nos 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 à 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).

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. 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 *sla-*

ves, 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 essas versões 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 essa ocorra. Essa técnica necessita tempo adicional, e é utilizada em sistemas onde o tempo não é crítico. Como exemplo pode-se citar 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 ainda o percentual de disponibilidade do serviço. Além disso, um SLA deverá conter uma 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 por exemplo, os tipos de tolerância a falhas, os tipos de redundância e a forma de calcular a disponibilidade de um serviço. Pode-se concluir que a alta disponibilidade é alcançada através da tolerância a falhas, que por sua vez utiliza-se de redundância para mascarar as falhas e reduzir os SPOF.

Como mencionado anteriormente, um dos principais recursos utilizados atualmente para a obtenção de alta disponibilidade é a virtualização, uma vez que essas são utilizadas para implementar a redundância a nível de *software*. Desta forma, no próximo capítulo será feita uma breve definição de virtualização, com as vantagens, as estratégias de implementação e os grupos de máquinas virtuais. Neste trabalho, será implementado redundância através de dois servidores, sendo que neste ambiente será utilizado virtualização de sistema utilizando o hipervisor *Kernel-based Virtual Machine* (KVM).

3 VIRTUALIZAÇÃO

O conceito de virtualização surgiu na década de 60, onde o usuário muitas vezes necessitava de um ambiente individual, com as suas próprias aplicações e totalmente isolado dos demais usuários. Esse foi um das principais motivações 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 à 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, a JVM 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 ser executada em uma plataforma Y (LAUREANO; MAZIERO, 2008).

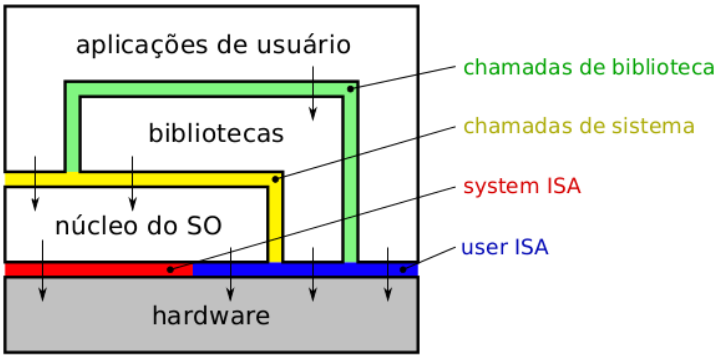
Como mencionado, a virtualização permite a comunicação entre diferentes interfaces. Destaca-se que existem diferentes tipos de interfaces em sistemas de computação (MAZIERO, 2013), sendo que entre essas pode-se citar:

- 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 que estão disponíveis para as aplicações de usuários. Essas instruções executam em modo usuário, sendo que neste modo existem restrições que procuram garantir um controle e segurança no acesso aos recursos de *hardware*. As 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 funções oferecidas pelo núcleo do sistema operacional para as aplicações dos usuários. Essas funções permitem um acesso controlado aos dispositivos, à 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 ou não 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 feita 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)

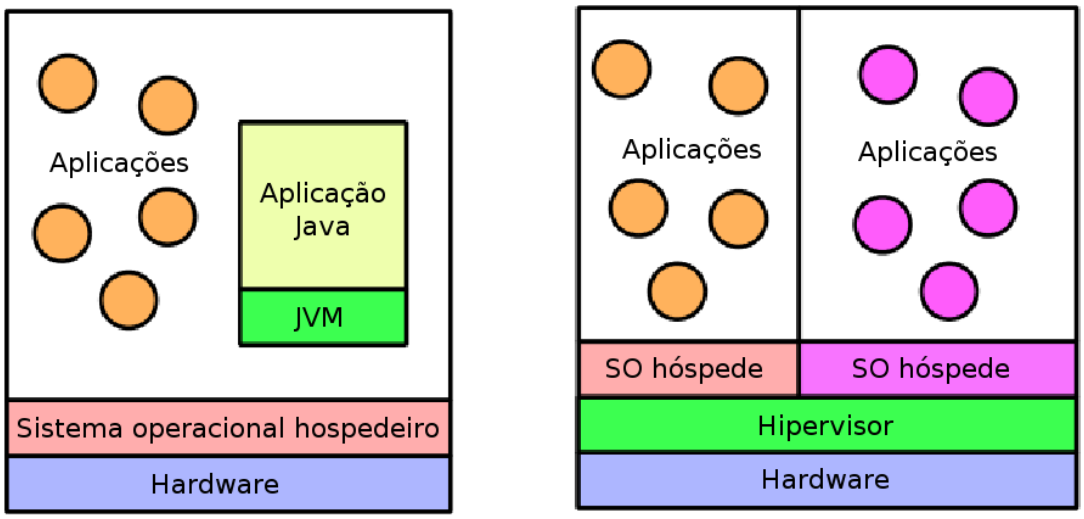
Figura 3.1: Interfaces de sistemas de computação.

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, 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

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

sobre ele. Uma máquina virtual executando sobre o hipervisor KVM (OVA, 2016) é um exemplo de uma máquina virtual de sistema (LAUREANO; MAZIERO, 2008).

Na Figura 3.2 (a) tem-se o modelo de máquina virtual de aplicação, onde uma JVM, juntamente com aplicações, está executando sobre um sistema operacional hospedeiro. A Figura 3.2 (b) apresenta uma 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 de 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 JVM e a *Microsoft Common Language Infrastructure*, que é a base da plataforma *.NET* (CARISSIMI, 2008);

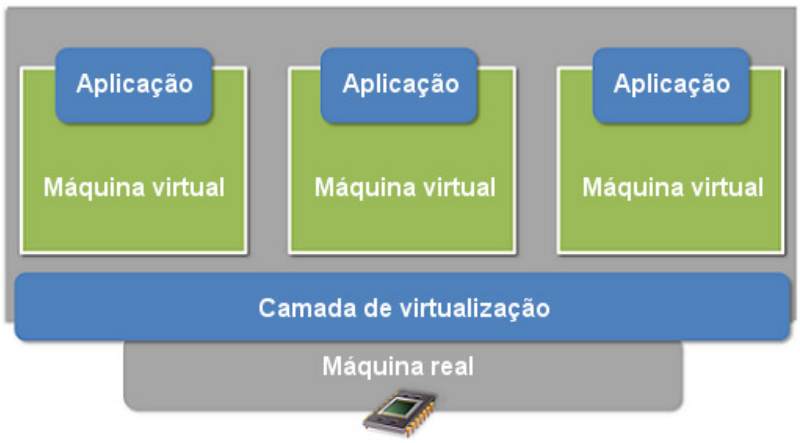
- Emulação do sistema operacional: nesse caso é feito um mapeamento entre as chamadas de sistema que são utilizadas pela aplicação e as chamadas de sistema operacional hospedeiro. A virtualização de aplicação pode ser encontrada em ferramentas que emulam uma aplicação que foi desenvolvida para uma plataforma em uma outra plataforma. Como exemplo, pode-se citar o *Wine* (WineHQ, 2016), que permite a execução de 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 à 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 discos 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.



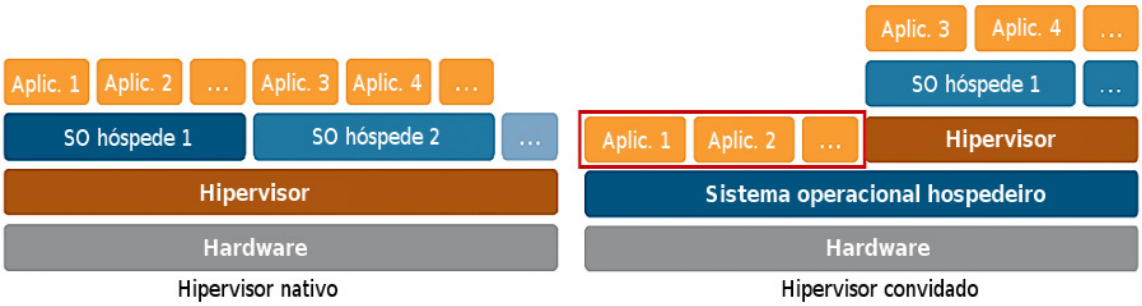
Fonte: ANDRADE (2011)

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 disponibiliza 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 arquitetura 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).



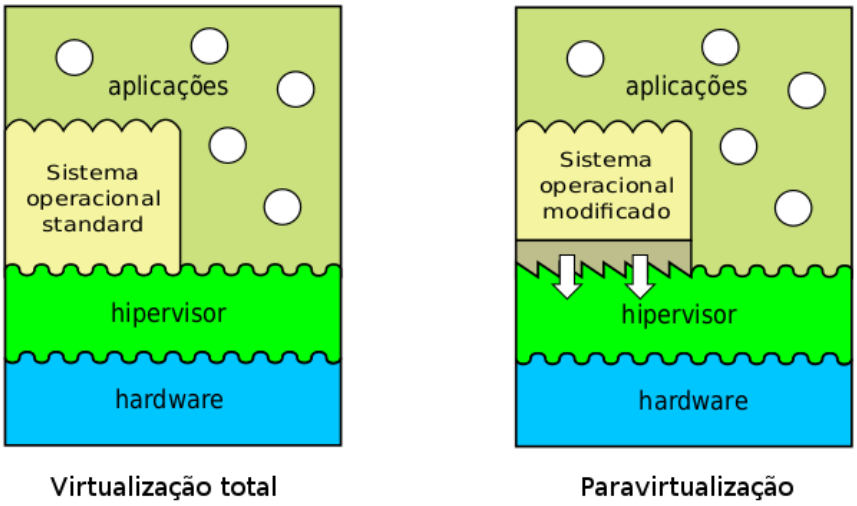
Fonte: SANTOS MACEDO; SANTOS (2016)

Figura 3.4: Arquiteturas de máquinas virtuais de sistema.

Os hipervisores convidados são mais flexíveis que os nativos, pois podem ser facilmente instalados ou removidos de um sistema operacional já instalado. Por outro lado, os hipervisores nativos possuem melhor desempenho pois acessam o *hardware* de forma direta.

3.2.2 Implementações de máquinas virtuais de sistema

As máquinas virtuais de sistema podem ser implementadas utilizando-se de 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-se de uma 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: essa estratégia utiliza uma arquitetura de hipervisor nativo, sendo que a interface entre o hipervisor e o sistema operacional convidado é modificada para se obter uma maior eficiência. 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 converte 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 responsável somente por impedir que o sistema convidado execute operações indevidas. Como exemplo pode-se citar o controle de acesso à memória que é 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, não sendo necessário nenhuma conversão de endereços (MAZIERO, 2013).

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. Desta forma, qualquer sistema operacional pode ser instalado em um ambiente de virtualização total. Além disso, essa técnica permite virtualizar um sistema operacional já instalado 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 possibilita a execução de múltiplas plataformas de desenvolvimento sem comprometer o sistema operacional original (CARISSIMI, 2008). Um exemplo é o *VMware Workstation*, que possibilita a virtualização em PCs para fins de desenvolvimento de *software* (WMware, 2016).

Em empresas pode-se ainda utilizar a 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, os *desktops* 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 esses serão armazenados em um local seguro, como por exemplo, um *data center*. 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 encontra-se disponível

para a plataforma utilizada. Deste modo, a virtualização possibilita executar diferentes sistemas operacionais no computador do usuário. Por exemplo, para um usuário de sistema operacional *MacOS* é comum a necessidade de executar aplicações que não existem para a sua plataforma, sendo assim esse pode utilizar uma máquina virtual para executar essas aplicações.

Pode-se também encontrar virtualização de *desktops* em laboratórios de ensino, devido à 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

É muito comum as empresas utilizarem serviços distribuídos entre diferentes servidores físicos, como por exemplo, servidores de *e-mail*, hospedagens de sites e banco de dados. Essa estrutura faz com que alguns recursos fiquem ociosos, pois em muitos casos esses serviços necessitam de uma quantidade de recursos inferior ao que o servidor físico oferece. Por exemplo, um serviço de transmissão de *streaming* de áudio utiliza pouco acesso ao disco rígido, porém utiliza um poder de processamento e de memória RAM (*Random Access Memory*) 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 nos custos de administração e manutenção dos servidores físicos.

Em um ambiente heterogêneo pode-se também utilizar a 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 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 na equipe responsável pela manutenção. 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 do espaço físico necessário para esses.

Por fim, a virtualização possibilita o uso de 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*.

3.4 Considerações finais

Neste capítulo foi apresentado um breve histórico da virtualização e os dois principais grupos de máquinas virtuais existentes, que são: máquinas virtuais de aplicação e máquinas virtuais de sistema. Também foram apresentadas as vantagens e as estratégias de implementação de máquinas virtuais, dando ênfase para as máquinas virtuais de sistema, uma vez que essas foram utilizadas no desenvolvimento deste trabalho para fazer a redundância de *software*. Para tal desenvolvimento, foi utilizado o hipervisor KVM que é caracterizado pela virtualização total.

No próximo capítulo será feito o levantamento e análise dos serviços fornecidos pela empresa que está sendo estudada neste trabalho. Também será detalhado a instalação física, do ambiente e dos servidores, bem como as configurações de todas as máquinas virtuais e seus respectivos *softwares* e serviços.

4 INFRAESTRUTURA DA EMPRESA

Este capítulo apresentará a infraestrutura de TI da empresa, descrevendo o ambiente, a configuração dos servidores e dos serviços. 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, esta empresa possui aproximadamente 9000 clientes. A sede da empresa está localizada na cidade de Garibaldi. Além disso, a empresa possui quatro filiais no estado, atendendo aproximadamente 30 cidades.

A empresa oferece serviços pela Internet, 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) e rádio *online*. 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.

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 arescondicionados, que foram denominados de *Ar 1 (Inverter)* e *Ar 2* na Figura 4.1.

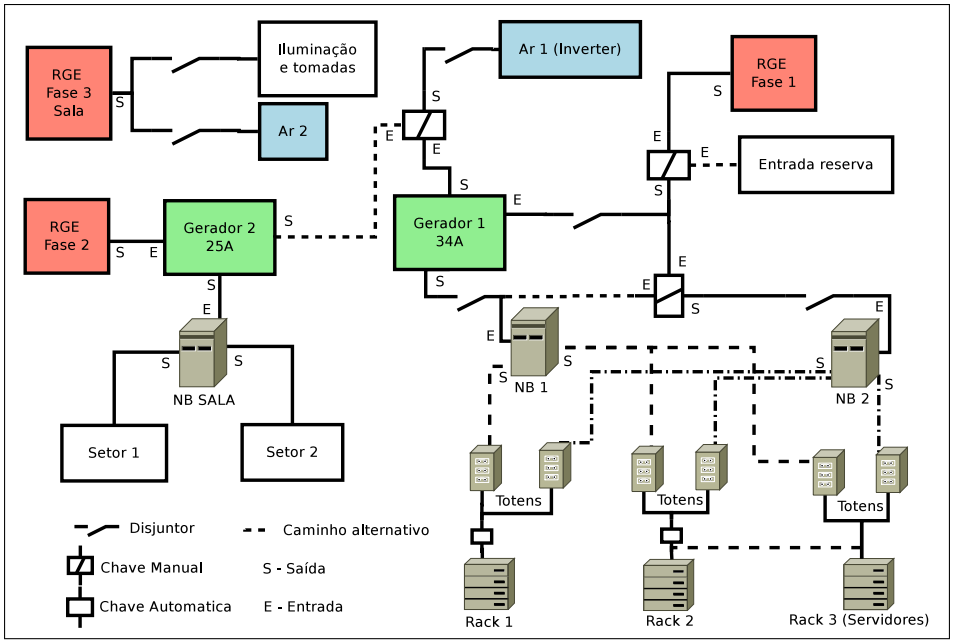


Figura 4.1: Diagrama de instalação elétrica.

[1]É importante salientar que esse provedor utiliza a maior parte dos serviços da empresa.

A redundância de energia é feita através de três *nobreaks*, sendo que dois deles (identificados como *NB 1* e *NB 2* na Figura 4.1) são utilizados para a 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* na Figura 4.1) é utilizado para alimentar os computadores dos funcionários de dois setores da empresa. Conectados aos *nobreaks* estão seis *totens*[1] de energia, sendo que nestes *totens* são ligados os equipamentos e os servidores que estão localizados 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 breve descrição da estrutura da empresa. Na Seção 4.1 serão descritos os servidores físicos, com suas estruturas e configurações. Na Seção 4.2 serão descritos os servidores que não utilizam virtualização, ou seja, os servidores que possuem os serviços instalados diretamente sobre o sistema operacional nativo. Na Seção 4.3 será descrita a estrutura dos servidores que utilizam de virtualização e todos os serviços fornecidos por esses servidores.

4.1 Instalação física

A estrutura da empresa é composta por quatorze servidores físicos. A configuração desses servidores pode ser observada na Tabela 4.1, onde tem-se o nome do servidor, o modelo, a configuração dos processadores, quantidade de memória RAM (*Random Access Memory*), 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 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

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

de tráfego de dados e obter redundância de interfaces. O diagrama da Figura 4.2 demonstra uma visão geral da estrutura dos servidores da empresa.

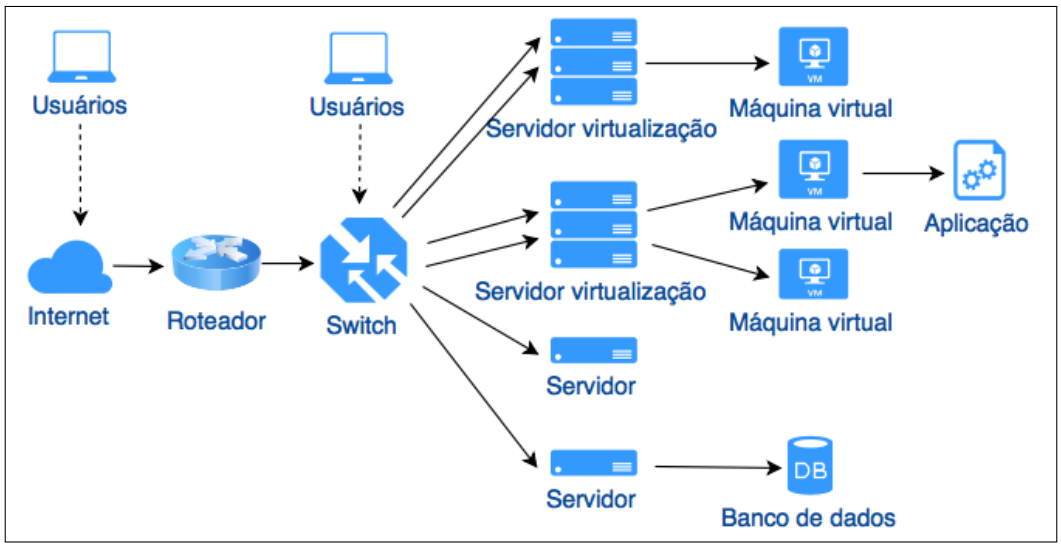


Figura 4.2: Modelo de estrutura física.

A Figura 4.3 apresenta uma foto, com o *rack* e todos os servidores, inclusive o *switch*.

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

4.2 Servidores sem virtualização

Tem-se sete servidores que possuem os serviços executando sobre o sistema operacional nativo, ou seja, sem virtualização. Esses 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* da empresa. Para isso ele possui instalada a ferramenta *Bacula Storage 5.2.6* (Bacula, 2016). Destaca-se que esse servidor armazena apenas os dados do *backup*, ou seja, esse servidor não é responsável pela execução do *backup*. A ferramenta que faz a execução e a gerência do *backup* é o *Bacula Director*, que encontra-se instalado no servidor *Quebei* o qual 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 SNMP (*Simple Network Management Protocol*) (KUROSE; ROSS, 2006). Essa ferramenta monitora a maior parte da rede *core*[1] e da rede *backbone*[2], 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), *Icewarp Server* (Icewarp, 2016) e *Ejabberd* (ProcessOne, 2016). Essas aplicações estão executando nos servidores *Vigilante* (servidor de câmeras), *Merak* (servidor de *e-mail*) e *Parla* (servidor de mensagens instantâneas), respectivamente. Esses servidores 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* é responsável por monitorar o *hardware* e os serviços que estão executando em cada servidor. Já o *Munin* é responsável por gerar gráficos de monitoramento. A partir do *Munin* pode-se criar, por exemplo, gráficos com a utilização do processador, de memória RAM, do disco rígido, da temperatura e da velocidade dos *fans*[3];
- *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,

[1] A rede *core* é a rede de transporte do provedor, por onde passam os principais *links* de acesso à Internet.

[2] A rede *backbone* é a rede de transporte entre os pontos de atendimento a clientes.

[3] Os *fans* são exaustores, ou seja, ventiladores que empurram o ar quente para fora de um recipiente ou ambiente (ATS, 2012). Nos servidores, os *fans* removem o ar quente gerado pelos componentes de *hardware*.

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 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 de dados dos clientes do provedor, 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 a aplicação responsável pela captura e pelo armazenamento das imagens das câmeras.

4.3 Servidores com virtualização

Os servidores de virtualização possuem suas respectivas VMs, sendo que, para a criação dessas VMs utiliza-se o hipervisor KVM (OVA, 2016) 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 hospedeiro. 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 RAID (TANENBAUM; WOODHULL, 2009). 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 redundância no cabeamento de rede, como visto anteriormente.

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

4.3.1 Servidor Brina

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

- *Masterauth*: sua configuração é de 1 *core* de *2.33 GHz*, 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 virtual fornece um serviço de autenticação PPPoE (*Point-to-Point Protocol over Ethernet*) (TECHNOLOGIES, 2005) para uma parte dos clientes do provedor. Para a autenticação é utilizado o *software Freeradius 2.1.12* (FreeRADIUS, 2016);
- *Monete*: sua configuração é de 1 *core* de *2.33 GHz*, 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.

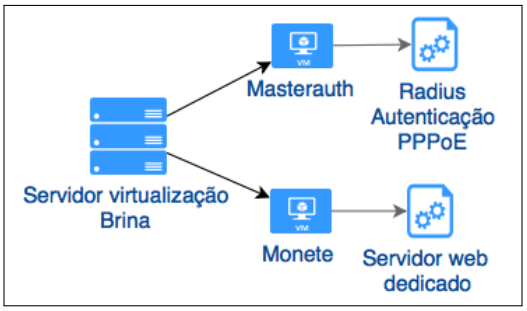


Figura 4.4: Servidor de virtualização *Brina*.

Para isso ele utiliza os *softwares Apache 2.4.7* (Apache Software Foundation, 2016a), *PHP 5.5.9* (PHP Group, 2016) 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 nas VMs são:

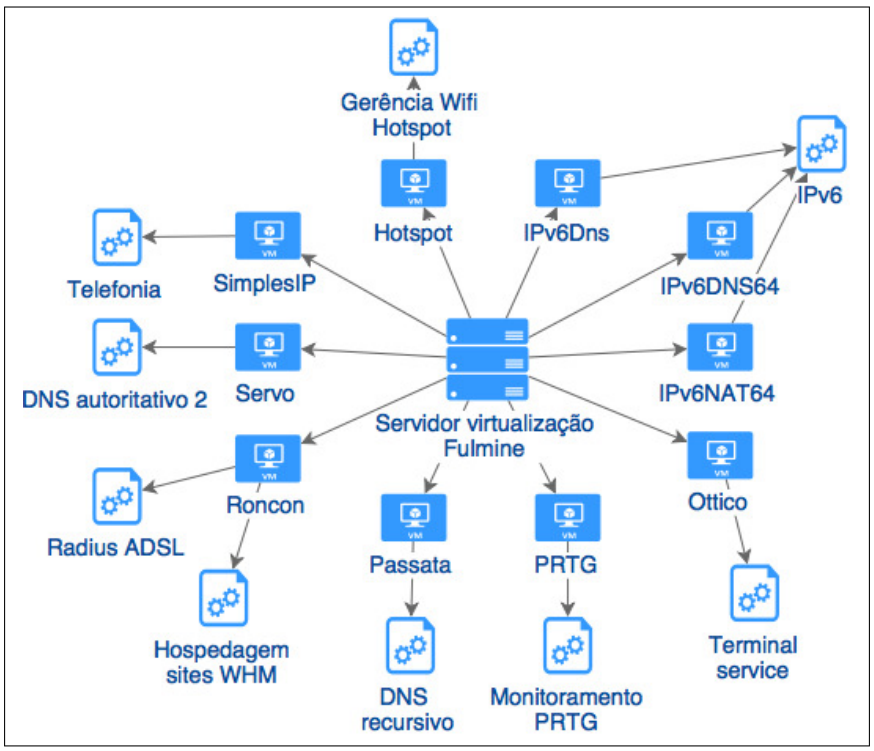


Figura 4.5: Servidor de virtualização *Fulmine*.

- *Hotspot*: sua configuração é de 1 *core* de *2.0 GHz*, 1,5 GB de memória RAM e 8 GB de disco. Esse servidor virtual possui o sistema operacional *Ubuntu 14.04 LTS* (Canonical, 2016), sendo que esse servidor faz a gerência de equipamentos *AP Unifi* fabricados pela empresa *Ubiquiti* que fazem *hotspot*[1]. Esses equipamentos disponibilizam a tecnologia *Wi-fi* para prover acesso à Internet em ambientes públicos e são utilizados pelo provedor;

[1] *Hotspot* é um tipo de rede sem fio para computadores móveis e normalmente estão presentes em locais como hotéis, aeroportos, entre outros (TANENBAUM; WETHERALL, 2011).

- *IPv6Dns*, *IPv6Dns64* e *IPv6Nat64*: suas configurações são de 1 *core* de *2.0 GHz*, 1 GB de memória RAM e 8 GB de disco. O sistema operacional é o *Ubuntu 14.04 LTS* (Canonical, 2016), sendo que essas VMs fornecem o serviço de DNS (*Domain Name System*) e NAT (*Network Address Translation*) para navegação IPv6 (*Internet Protocol version 6*) (NIC.br, 2016) do provedor;
- *Ottico*: esse servidor possui 2 *cores* de *2.0 GHz*, 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*, 4 GB de memória RAM e 100 GB de disco. O servidor possui o sistema operacional *Windows 2008 Server R2* e sua função é monitorar o tráfego de rede dos equipamentos da rede *core* do provedor;
- *Passata*: esse servidor possui 2 *cores* de *2.0 GHz*, 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 a navegação dos clientes do provedor;
- *Roncon*: esse servidor possui 4 *cores* de *2.0 GHz*, 6 GB de memória RAM e 400 GB de disco. Ele possui o sistema operacional *Red Hat 5.11* (Red Hat, 2016a) 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 de banco de dados. Além disso, encontra-se disponível a ferramenta *cPanel*, que faz parte do WHM e fornece acesso aos desenvolvedores de sites. Para fornecer a hospedagem desses sites 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 (*Asymmetric Digital Subscriber Line*) de terceiros utilizando o *software Freeradius 1.1.3* (FreeRADIUS, 2016);
- *Servo*: sua configuração é de 1 *core* de *2.0 GHz*, 2 GB de memória RAM e 30 GB de disco. Esse servidor possui o sistema operacional *CentOS 6.8* (CentOS, 2016), sendo que este servidor virtual fornece, através do *software Bind 9.8.2* (ISC, 2016), o serviço de DNS autoritativo. Esse é o servidor de DNS secundário dos domínios hospedados pela empresa;
- *SimplesIP*: esse servidor possui 2 *cores* de *2.0 GHz*, 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 IP (*Internet Protocol*) do provedor. Esse utiliza como base o *software 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 nas VMs são:

- *ASP*: esse servidor possui 1 *core* de *2.40 GHz*, 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*

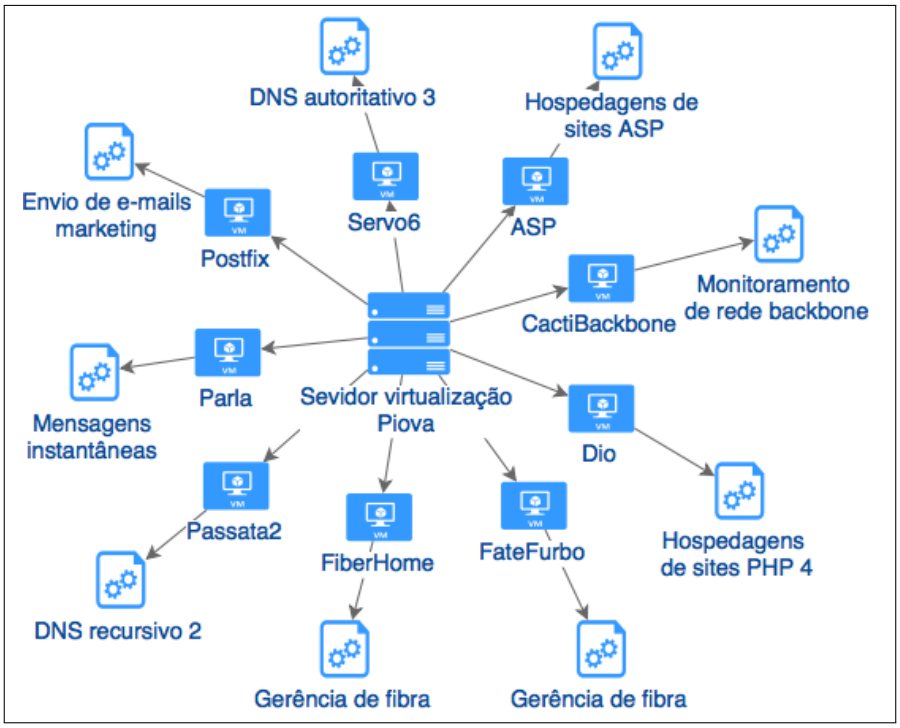


Figura 4.6: Servidor de virtualização *Piova*.

(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. De fato, esse servidor armazena somente 10 sites;

- *CactiBackbone*: esse servidor possui 1 *core* de *2.40 GHz*, 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 uma parte da rede *backbone* do provedor;
- *Dio*: esse servidor possui 1 *core* de *2.40 GHz*, 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 do PHP. Esse servidor armazena somente 10 sites;
- *FateFurbo*: sua configuração é de 2 *cores* de *2.40 GHz*, 4 GB de memória RAM e 80 GB de disco. O sistema operacional é o *Ubuntu 14.04 LTS* (Canonical, 2016). Esse servidor 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 de 2 *cores* de *2.40 GHz*, 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 para a gerência da fibra óptica do provedor;
- *Parla*: sua configuração é de 1 *core* de *2.40 GHz*, 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 XMPP (*EXtensible Messaging and Presence Protocol*) (XSF, 2016). Esse serviço é utilizado para comunicação entre funcionários da empresa e do

provedor, e também entre os 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*, 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 de 1 *core* de *2.40 GHz*, 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. Ou seja, esse servidor faz o envio de *e-mails* em massa para divulgação de informações ou produtos;
- *Servo6*: sua configuração é de 1 *core* de *2.40 GHz*, 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. Esse 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. O acesso a essas máquinas virtuais é realizado através de um serviço de acesso remoto, como por exemplo, o de SSH (*Secure Shell*) (BARRETT; SILVERMAN; BYRNES, 2005).

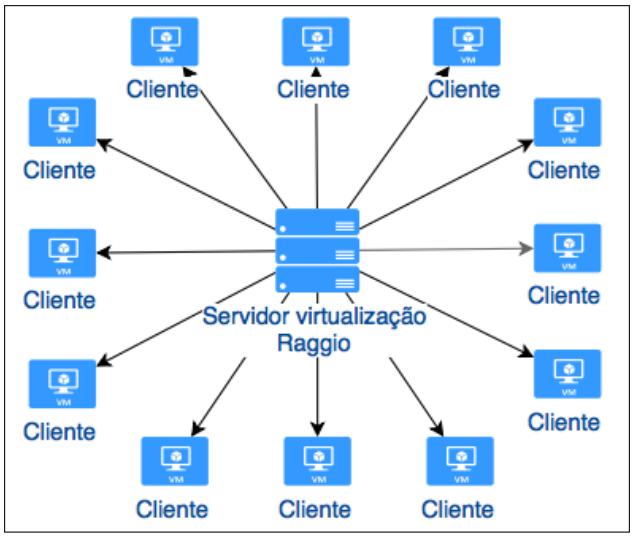


Figura 4.7: Servidor de virtualização *Raggio*.

4.3.5 Servidor Tempesta

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

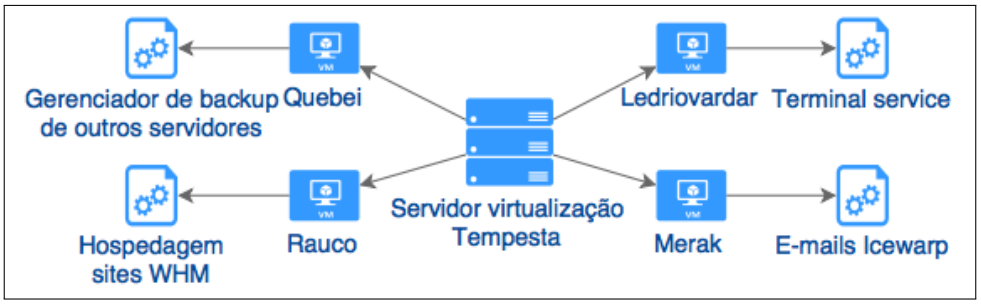


Figura 4.8: Servidor de virtualização *Tempesta*.

- *Ledriovardar*: esse servidor possui 2 *cores* de *2.40 GHz*, 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*. Ele possui uma configuração de 6 *cores* de *2.00 GHz*, 10 GB de memória RAM e 1 TB de disco. O servidor possui o 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 SMTP (*Simple Mail Transfer Protocol*); recebimentos de *e-mails* através dos protocolos POP (*Post Office Protocol*) e IMAP (*Internet Message Access Protocol*); e 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 o serviço de Internet fornecida pelo provedor;
- *Quebei*: sua configuração é de 1 *core* de *2.00 GHz*, 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 os dados de *backup* são armazenados no servidor físico *Bello*, que foi detalhado na Seção 4.2. Além disso, esse servidor possui o sistema gerenciador de banco de dados *MySQL 5.5.49* (Oracle, 2016b), que está configurado no modelo *master-slave*, sendo que esse servidor é o *slave* e o servidor *Dati* é o *master*;
- *Rauco*: esse servidor possui 2 *cores* de *2.00 GHz*, 6 GB de memória RAM e 600 GB de disco. Ele possui o sistema operacional *CentOS 6.8* (CentOS, 2016), e fornece o mesmo serviço do servidor *Roncon*, que foi descrito na Seção 4.3.2. Ou seja, esse servidor fornece acesso a sites *Web* desenvolvidos com a linguagem PHP. Foram criados dois servidores para a hospedagem de forma a suportar um número maior de sites.

4.3.6 Servidor Tuono

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

- *Mondoperso*: sua configuração é de 1 *core* de *2.53 GHz*, 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 uma *Web* rádio. Esse serviço é feito através do *software* livre *Icecast 2.3.3* (Xiph.Org

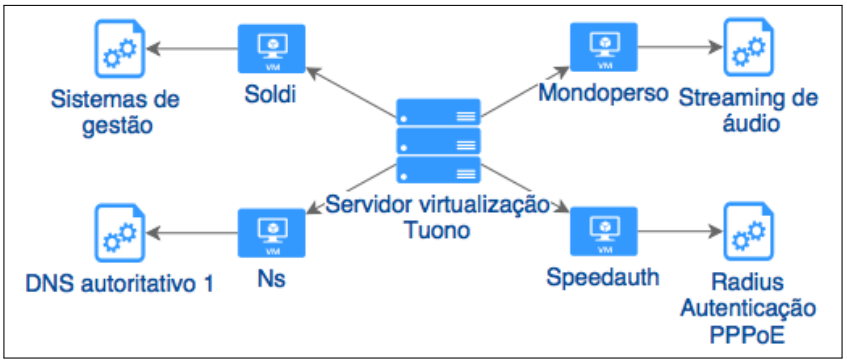


Figura 4.9: Servidor de virtualização *Tuono*.

Foundation, 2016);

- *Ns*: esse servidor possui 1 *core* de *2.53 GHz*, 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;
- *Soldi*: sua configuração é 4 *cores* de *2.53 GHz*, 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 os *softwares* de gestão que são desenvolvidos pela empresa. Os seguintes *softwares* encontram-se instalados neste servidor: *Apache 2.4.7* (Apache Software Foundation, 2016a), *PHP 5.5.9* (PHP Group, 2016) e *MySQL 5.5.49* (Oracle, 2016b);
- *Speedauth*: sua configuração é de 2 *cores* de *2.53 GHz*, 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 é responsável pelas autenticações da maior parte dos usuários do provedor.

4.3.7 Servidor Venti

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

- *Backup*: sua configuração é de 1 *core* de *3.10 GHz*, 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 servidor utiliza *scripts* que foram desenvolvidos internamente e que efetuam a cópia de dados através do protocolo FTP (*File Transfer Protocol*);
- *Esibire*: sua configuração é de 1 *core* de *3.10 GHz*, 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 utilizando o protocolo FTP. Além disso, ele faz a reprodução de *streaming* utilizando um servidor *Web Apache 2.4.7*;
- *Miatanto*: sua configuração é de 1 *core* de *3.10 GHz*, 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

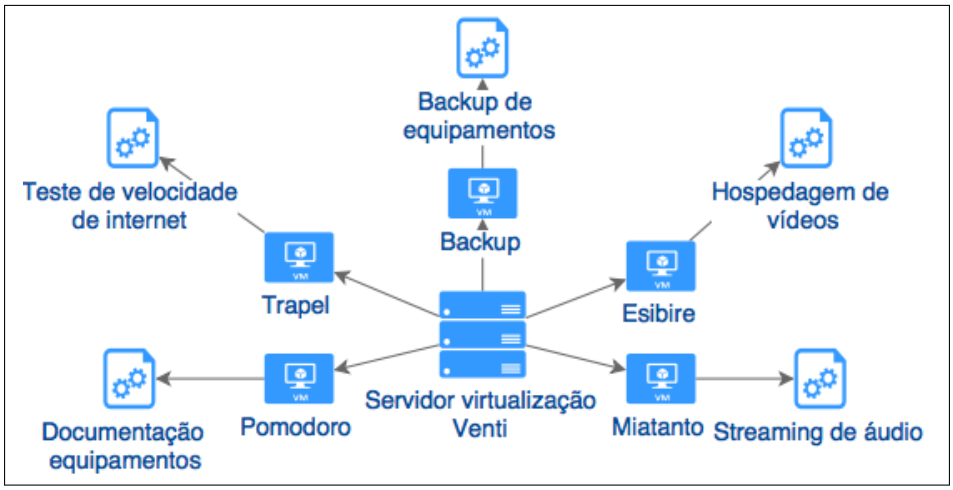


Figura 4.10: Servidor de virtualização *Venti*.

serviço é feito através do *software* livre *Icecast 2.3.3* (Xiph.Org Foundation, 2016);

- *Pomodoro*: sua configuração é de 1 *core* de *3.10 GHz*, 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 de 1 *core* de *3.10 GHz*, 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 para conexões de 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 apresentada a infraestrutura de TI da empresa estudada, com isso, pôde-se conhecer o *hardware* que é utilizado e o *hardware* que está disponível. Além disso, pôde-se ter uma visão geral dos serviços fornecidos, dos *softwares* utilizados, bem como do *hardware* que vem sendo utilizado.

Com base nos serviços apresentados neste capítulo, o próximo capítulo apresentará os serviços que são considerados críticos, para tanto, será definidos alguns critérios. Posteriormente, serão definidas algumas ferramentas para a criação do ambiente de alta disponibilidade, que farão parte do projeto de implementação.

5 SERVIÇOS CRÍTICOS

No capítulo anterior foram detalhados os serviços que estão disponíveis na empresa. Neste capítulo serão apresentados os serviços que foram considerados críticos para a empresa, sendo que para a definição desses foram adotados alguns critérios. Esses critérios foram criados através de uma análise dos serviços, levando em consideração a importância para a empresa, para o seu ambiente e a opnião da direção da empresa (AREND, 2014). Mais especificamente, os critérios definidos foram:

- A quantidade de clientes que utilizam o serviço: esse é o critério 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 ao tempo que ficou sem acesso;
- O número de requisições: esse número é importante, uma vez que, indica a quantidade de usuários que dependem do serviço e a frequência de utilização do serviço. Esse critério engloba, por exemplo, o número de conexões TCP (*Transmission Control Protocol*), o número de requisições UDP (*User Datagram Protocol*), a quantidade de acessos em um servidor de hospedagens de sites e a quantidade de requisições DNS (*Domain Name System*) em um servidor recursivo;
- O volume de elementos do serviço: essa medida demonstra a abrangência do serviço, ou seja, quantos clientes são dependentes deste. Como exemplo de elementos pode-se citar a quantidade de contas de *e-mail* ativas em um servidor de *e-mail* ou a quantidade de equipamentos monitorados por um servidor.

Nas próximas seções serão descritos os serviços que foram considerados críticos, com base nos critérios apresentados.

5.1 DNS recursivo primário

Esse serviço foi classificado como o serviço mais importante pois possui um impacto direto nos clientes do provedor. Além disso, esse é o único serviço que todos os clientes e funcionários utilizam, totalizando aproximadamente 9000 pessoas. O objetivo de um provedor é fornecer uma navegação de qualidade aos seus clientes, sendo assim, o DNS é fundamental para essa navegação. A importância deste serviço está ilustrada na Figura 5.1 (a), onde pode ser observado que esse serviço possui picos de aproximadamente 1150 requisições por segundo. Já na Figura 5.1 (b) pode ser observado que o servidor *Passata* é o servidor que apresenta o maior número de

requisições UDP[1]. Essa figura compara os principais servidores da empresa através de requisições UDP e pode ser usada como um indicador da quantidade de clientes que utilizam determinados serviços.

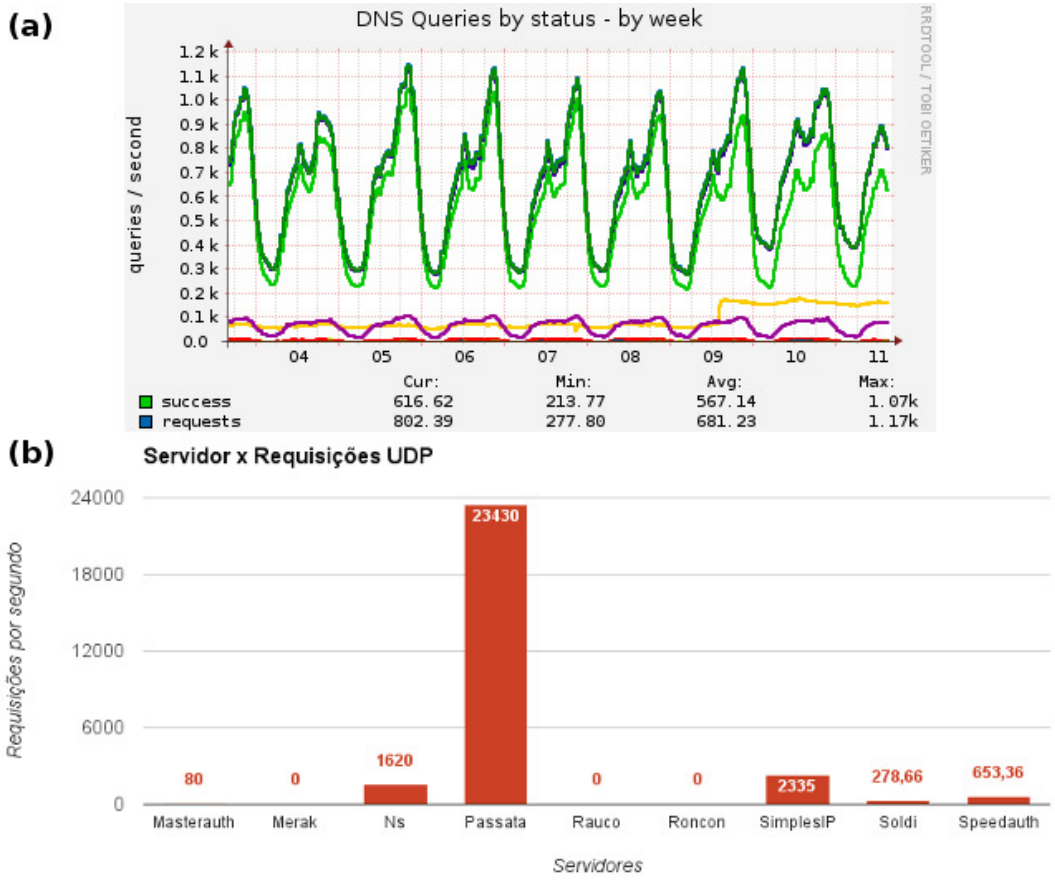


Figura 5.1: Gráfico de requisições DNS por segundo do período de uma semana (a) e comparação de requisições UDP simultâneas máximas entre os principais servidores (b).

5.2 Autenticação Radius

Esse serviço é importante pois é o responsável pela autenticação de todos os clientes do provedor. Caso esse serviço fique indisponível, os clientes não conseguirão estabelecer conexão e, consequentemente, não conseguirão utilizar o serviço de Internet. Os servidores *Masterauth* e *Speedauth* fornecem o serviço de *Radius*, sendo que esses recebem em média, 1,6 requisições de autenticação por segundo. Além disso, esses servidores armazenam dados relacionados à conexão dos clientes, como por exemplo, o endereço de IP que é utilizado por um cliente em um determinado período de tempo, o tráfego de dados da conexão, o tempo de conexão, o endereço MAC (*Media Access Control*) dos equipamentos dos clientes, entre outros. Essas operações resultam em média 23 requisições por segundo. Outro critério que é relevante para esses servidores é o volume de elementos do serviço, que neste caso,

[1]Esse número de requisições UDP é elevado devido ao fato do serviço DNS utilizar esse protocolo de transporte.

representa a quantidade de clientes que utilizam esses servidores para autenticação. De fato, como pode-se observar na Figura 5.2 (a) que os servidores *Masterauth* e *Speedauth* estão entre os que apresentam o maior número de contas. Além disso, nesses servidores existe um grande número de conexões TCP[1] simultâneas, como pode ser observado na Figura 5.2 (b).

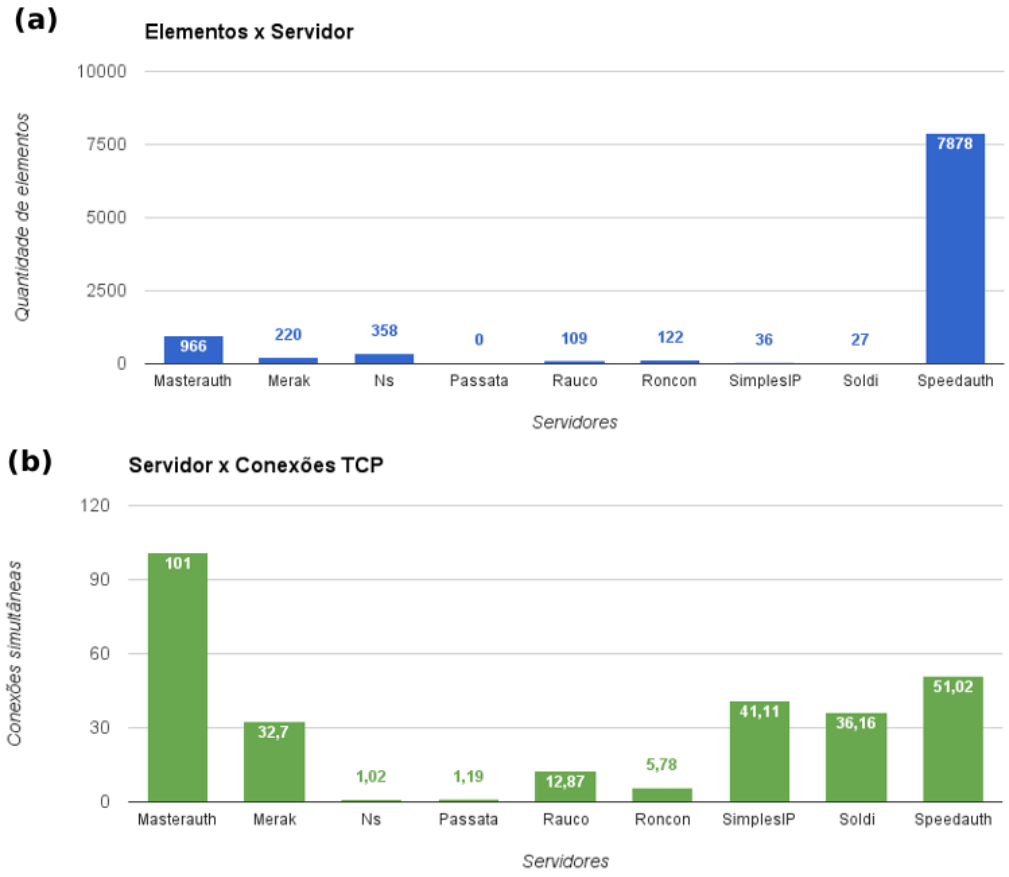


Figura 5.2: Gráfico de comparação de elementos (a) e de conexões TCP simultâneas máximas (b) entre os principais servidores.

5.3 Sistemas da empresa e do provedor

O sistema do provedor é responsável pela maior parte das operações gerenciais do provedor. Este sistema é responsável pela emissão de boletos, 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 o funcionamento 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 um total de 65 funcionários.

O sistema do provedor é executado no servidor *Soldi* que recebe aproximadamente 3 requisições HTTP (*Hypertext Transfer Protocol*) por segundo (Figura 5.3). Além disso, a empresa mantém 28 sistemas de outros clientes nesse servidor. Como

[1]Esse número de conexões TCP deve-se ao fato do *software Freeradius* utiliza esse protocolo para a comunicação com o seu banco de dados.

pode ser observado na Figura 5.2 (b), esse servidor encontra-se entre os que apresentam um maior número de conexões TCP[1].

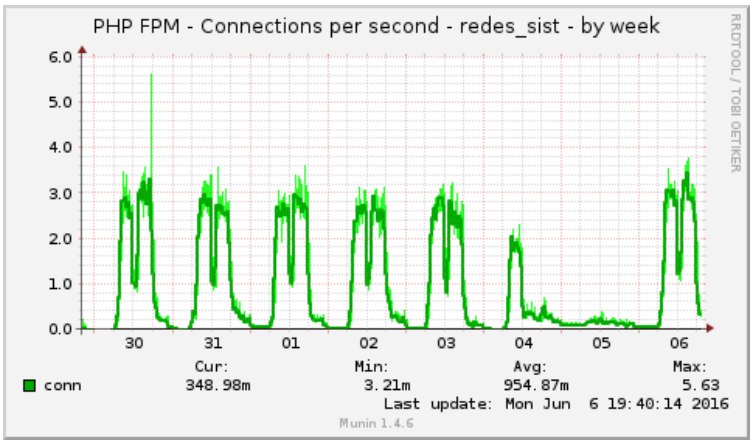


Figura 5.3: Gráfico de requisições por segundo do sistema do provedor durante o período de uma semana.

5.4 Telefonia interna sobre IP

Esse serviço tem relevância para a empresa e para o provedor, pois permite a comunicação entre os clientes e os funcionários. De fato, o servidor *SimplesIP* é responsável por garantir o atendimento dos 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. Além disso, no mesmo mês foram efetuadas 674 ligações entre funcionários.

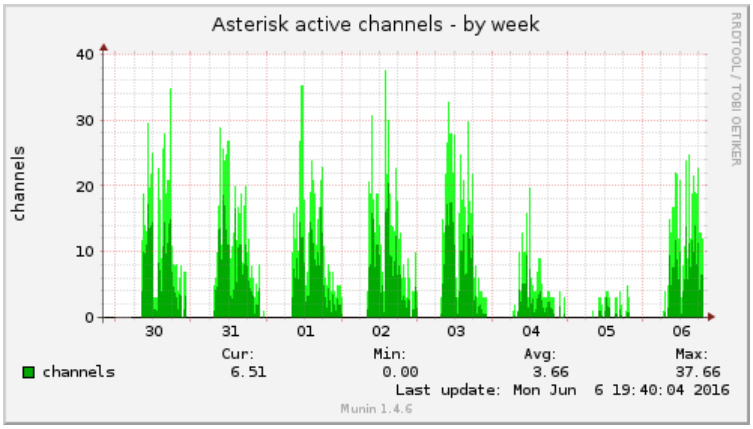


Figura 5.4: Gráfico da quantidade de canais ativos simultâneamente no servidor de telefonia durante o período de uma semana.

O gráfico da Figura 5.4 mostra a quantidade de canais ativos no servidor de telefonia. Observa-se que ocorrem de 20 a 30 ligações simultâneas durante o horário comercial, que é das 08:00 às 12:00 e das 13:00 às 18:00. Também pode-se observar,

[1]O número de conexões TCP é considerado devido ao fato do protocolo HTTP utilizar esse protocolo de transporte.

na Figura 5.1 (b), que esse serviço possui um elevado número de requisições UDP[2], quando comparado aos demais servidores.

5.5 Resumo e os serviços críticos

A partir da análise feita decidiu-se que os servidores com maior importância para a empresa são:

- *Passata*: servidor de DNS recursivo utilizado tanto pelo provedor quanto pela empresa;
- *Speedauth*: servidor *Radius* para autenticação dos clientes do provedor;
- *Masterauth*: servidor *Radius* para autenticação dos clientes do provedor;
- *Soldi*: servidor dos sistemas gerenciais da empresa e do provedor;
- *SimplesIP*: servidor de telefonia sobre IP para atendimento dos clientes e comunicação interna do provedor e da empresa;

Na Tabela 5.1, tem-se esses servidores, seus respectivos serviços, o percentual de *Uptime* e o tempo de *Downtime* por ano. Destaca-se que o serviço de telefonia do servidor SimplesIP foi implantado em 06/2015, sendo assim foi apresentada a medição apenas dos últimos 6 meses de 2015. A partir da implementação da solução de alta disponibilidade deste trabalho pretende-se atingir um *Uptime* superior a 99,99% em todos esses serviços.

Servidor	**Serviço**	**Uptime**	**Downtime por ano**
Passata	DNS recursivo	99,913%	7 horas 37 minutos 30 segundos
Speedauth	Radius	99,755%	21 horas 25 minutos 50 segundos
Masterauth	Radius	99,475%	33 horas 47 minutos
Soldi	Sistemas	99,989%	59 minutos 30 segundos
SimplesIP	Telefonia	99,997%	15 minutos 10 segundos

Tabela 5.1: Serviços críticos do ano de 2015.

5.6 Considerações finais

Neste capítulo foram apresentados os serviços que foram considerados críticos, sendo que os critérios utilizados para a escolha destes foram criados de acordo com a análise feita nos serviços, levando em consideração a importância para a empresa, para o seu ambiente e a opnião da direção da empresa. Pôde-se observar que a maioria dos serviços críticos estão diretamente relacionados com o provedor, o qual possui a maior parte dos clientes. Os serviços críticos apresentados foram o DNS recursivo primário e a autenticação *Radius*, as quais possuem um impacto direto na navegação dos clientes do provedor. E os sistemas da empresa e do provedor juntamente com a telefonia interna, que são responsáveis pela gestão e pela gerência e operação da empresa e do provedor. No próximo capítulo serão apresentadas as ferramentas escolhidas para implementação do ambiente de alta disponibilidade.

[2]Esse número de requisições UDP deve-se ao fato da telefonia utilizar o protocolo UDP para a transmissão de voz.

6 SOFTWARES PARA IMPLEMENTAÇÃO

Neste capítulo serão apresetadas as ferramentas que irão compor o ambiente de alta disponibilidade. A solução será baseada na utilização de virtualização e de *softwares* de código aberto. Essa estrutura foi baseada nos trabalhos de GONçALVES (2009), REIS (2009) e ZAMINHANI (2008). Os dois primeiros trabalhos implementam alta disponibilidade através de máquinas virtuais, com os mesmos objetivos deste trabalho. O terceiro autor utiliza uma técnica semelhante, porém implementada para apenas um serviço. Nestes trabalhos a estrutura é baseada em *cluster*[1], onde é utilizado um *software* responsável pela replicação de dados e um outro para o monitoramento e a gerência do *cluster*.

A escolha do *software* de replicação de dados baseou-se em critérios que fossem compatíveis com as estruturas apresentadas nesses trabalhos e que fossem adequadas ao ambiente da empresa. Os critérios utilizados para a escolha do *software* de replicação de dados foram:

- Integração com virtualização: possibilidade de replicação dos dados das máquinas virtuais;
- *Dual-primary*: suporte para a leitura e escrita de dados simultaneamente em ambos os nós;
- Replicação em tempo real: replicação dos dados de forma instantânea e automática;
- Nível de replicação: possibilidade de replicação a nível de bloco ou de arquivo.

Já os critérios utilizados para a escolha do *software* de monitoramento e gerência do *cluster* foram:

- Suporte nativo à virtualização: possibilidade de monitorar e manipular máquinas virtuais;
- Migração de VMs em tempo real: suporte para a utilização de *live migration* das máquinas virtuais, ou seja, possibilita mover uma VM de um nó para outro sem reiniciá-lo;
- *Failover* e *failback* automáticos: suporte para a movimentação automática dos serviços de um nó que falhou para um outro nó disponível.

[1]Pode-se definir *cluster* como um grupo de computadores interligados por rede com o objetivo de aumentar o desempenho ou disponibilidade de um serviço (JUNIOR; FREITAS, 2005)

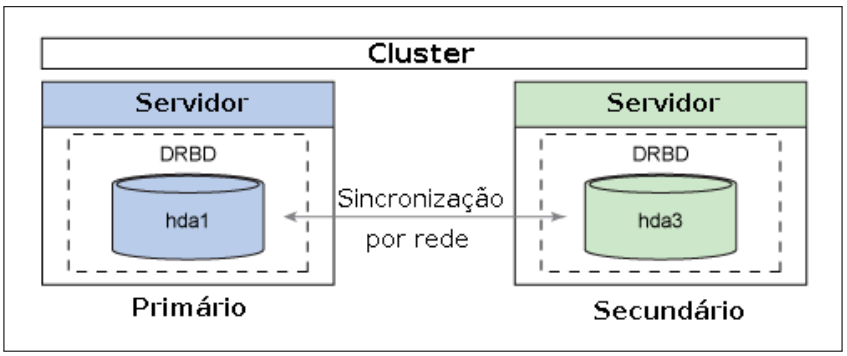
6.1 Softwares para a replicação de dados

A replicação de dados pode ser realizada de diferentes formas, esta pode ser a nível de aplicação ou até mesmo a nível de *hardware*. Dependendo do objetivo pode-se utilizar uma replicação através de um RAID. Essa solução é eficaz para garantir que o sistema não fique indisponível em caso de falha de discos[1] rígidos, porém não garante a disponibilidade quando um *software* ou algum outro componente de *hardware* falhar (ZAMINHANI, 2008).

A solução de replicação a ser adotada neste trabalho consiste em um espelhamento de dados através da rede. Essa solução permite a sincronização dos dados de um servidor para outro servidor em tempo real. Nas próximas seções tem-se uma descrição dos *softwares* de replicação de dados que foram estudados, bem como as funcionalidades suportadas pelos mesmos.

6.1.1 DRBD (*Distributed Replicated Block Device*)

O DRBD é um projeto de código aberto desenvolvido pela *LINBIT* (LINBIT, 2016). Esse *software* é uma solução de replicação de dispositivos de armazenamento, ou seja, esse permite a duplicação de um dispositivo de bloco (geralmente um disco rígido) em um servidor remoto. O DRBD é implementado através de um módulo do *kernel Linux*. Na Figura 6.1 tem-se dois servidores com seus respectivos discos rígidos, *hda1* e *hda3*, formando um *cluster*, sendo que esses discos estão sincronizados através da rede. Ou seja, todas as operações de escrita realizadas no disco rígido do nó primário são replicadas para o disco do nó secundário (ZAMINHANI, 2008).



Fonte: JONES (2010)

Figura 6.1: Exemplo do modelo *master-slave* do DRBD.

O DRBD pode ser configurado dos seguintes modos (LINBIT, 2016):

- *Single-primary* ou *master-slave*: neste modo apenas um nó do *cluster* pode ser o nó primário, sendo que somente o nó primário terá permissão para acessar o dispositivo, ou seja, somente ele poderá fazer operações de leitura e escrita. Já o nó secundário terá apenas uma réplica dos dados;
- *Dual-primary* ou *dual-master*: neste modo existem dois nós primários, nos quais podem ser realizadas operações de leitura e escrita de forma simultânea. Porém, este modo necessita de um sistema de arquivos compartilhados, sendo

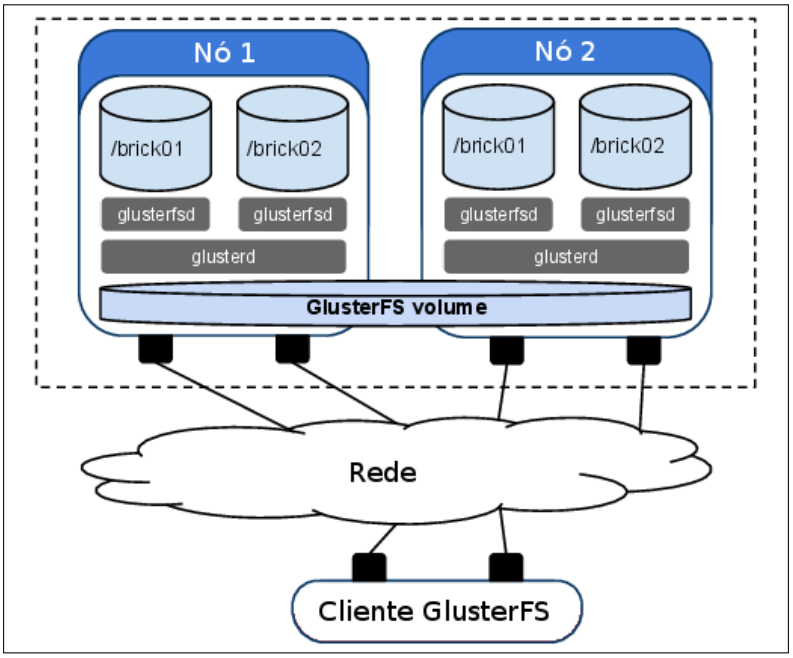
[1]Lembrando que essa solução é utilizada no ambiente atual para aumentar a disponibilidade dos servidores.

que neste caso podem ser utilizados os sistemas *Global File System* (GFS) (Red Hat, 2016b) e *Oracle Cluster File System 2* (OCFS2) (Oracle, 2016c).

6.1.2 GlusterFS

O *GlusterFS* (Red Hat, 2016c) é um sistema de arquivos distribuídos mantido pela *Gluster community*. Este utiliza estrutura de *cluster* e seu principal objetivo é a escalabilidade, ou seja, este possui funcionalidades que facilitam ampliar a capacidade do *cluster* a partir da inclusão de novos nós.

Na Figura 6.2 tem-se um exemplo de dois nós, onde cada nó possui dois discos rígidos, que são denominados *bricks*. A partir dos *bricks*, o *GlusterFS* constrói um volume lógico que é disponibilizado através da rede para os clientes *GlusterFS*. A organização destes *bricks* vai depender do objetivo da aplicação, sendo que uma das formas é a replicação. Os diferentes tipos de configurações serão detalhados a seguir.



Fonte: DAVIES; ORSARIA (2013)

Figura 6.2: Modelo do *GlusterFS*.

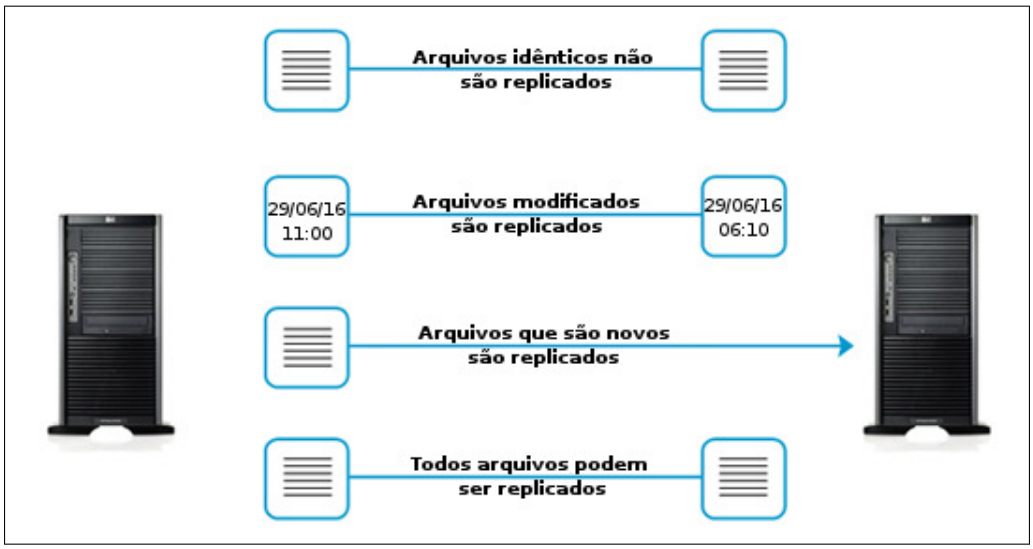
O *GlusterFS* suporta diferentes tipos de configurações de volumes, com as opções de distribuição e replicação de dados (Red Hat, 2016c):

- Volume distribuído: neste tipo os arquivos são distribuídos entre os diferentes *bricks* dos nós. O objetivo deste tipo de configuração é ampliar a capacidade de armazenamento. Neste modo, não se tem preocupação com a redundância de dados, sendo que no caso de uma falha em um nó haverá uma perda de todos dados;
- Volume replicado: neste tipo os arquivos são replicados entre os *bricks*, deste modo, tem-se uma redundância, ou seja, caso ocorra uma falha em um *brick* não haverá perda de dados;
- Volume distribuído e replicado: este é uma combinação dos dois tipos de volumes anteriores. Neste caso, é feita a distribuição e a replicação dos arquivos entre os nós;

- Volume listrado: neste tipo de configuração ocorre a distribuição de um mesmo arquivo entre os *bricks*, ou seja, um arquivo é dividido entre os *bricks*. Esse tipo é normalmente utilizado para o armazenamento de arquivos muito grandes e para garantir um melhor balanceamento de carga em sistemas com muito acesso à disco. Neste tipo de volume não se tem nenhuma forma de replicação de dados;
- Volume distribuído e listrado: este tipo de volume é uma combinação do distribuído e do listrado. Neste modo é feita a divisão do arquivo entre *bricks* distintos, sendo que estes são replicados.

6.1.3 Rsync

O *Rsync* (DAVISON, 2016) é um *software* desenvolvido e mantido por *Wayne Davison*. Esse *software* provê uma rápida transferência de arquivos, ou seja, ele faz o sincronização de arquivos transferindo-os de um servidor de origem para um de destino. A Figura 6.3 demonstra o funcionamento do *Rsync*. Observa-se que no *Rsync* tem-se uma replicação somente dos arquivos que ainda não foram atualizados ou que não existem no servidor de destino. Além disso, o *Rsync* permite uma replicação completa, sendo que neste caso os arquivos já existentes são sobrescritos.



Fonte: LOPEZ (2011)

Figura 6.3: Transferência de arquivos através do *Rsync*.

O *Rsync* pode ser configurado como servidor, o que permite que vários clientes possam sincronizar seus arquivos com ele. Além disso, a sincronização pode ser de cliente para cliente, utilizando como transporte o protocolo SSH ou através de *sockets*. Destaca-se que o *Rsync* não faz uma sincronização em tempo real, ou seja, ele faz a sincronização a partir de uma ação de um operador, por exemplo, a execução de um comando.

6.1.4 Resumo e software de replicação adotado

Na Tabela 6.1 tem-se uma comparação entre as ferramentas citadas anteriormente. A ferramenta adotada para replicação de dados foi o DRBD, pois permite a configuração *dual-primary* e também *master-slave*, além de suportar a replicação de discos das máquinas virtuais e fazer replicação a nível de bloco. Além disso, essa ferramenta permite a ressincronização dos dados de forma automática em caso de uma falha (LINBIT, 2016).

Critério	**DRBD**	**GlusterFS**	**Rsync**
Integração com virtualização	Sim	Sim	Não
Dual-primary	Sim	Sim	Não
Replicação em tempo real	Sim	Sim	Não
Nível de replicação	Bloco	Arquivo	Arquivo
Distribuições Linux	Suse, Debian, CentOS, Red Hat, Ubuntu	Debian, CentOS, Red Hat, Fedora, Ubuntu	Todas

Tabela 6.1: Comparação ferramentas de replicação de dados.

O *GlusterFS* poderia ser utilizado, porém este faz uma replicação a nível de arquivo, não sendo adequado para uma solução de alta disponibilidade que utiliza-se de virtualização. E, por fim, o *Rsync* não pode ser utilizado pois não executa uma replicação em tempo real, além de não ter sido desenvolvido para ser utilizado em conjunto com a virtualização.

6.2 Softwares para o gerenciamento do cluster

Para ser possível implementar uma solução de alta disponibilidade é necessário organizar os servidores em uma estrutura de *cluster*, sendo assim, é interessante a utilização de *softwares* que facilitam o gerenciamento deste *cluster*. Esses *softwares* permitem detectar falhas em um nó, sendo elas de *hardware* ou de serviços. Essas ferramentas são conhecidas como *Cluster Resource Management* (CRM).

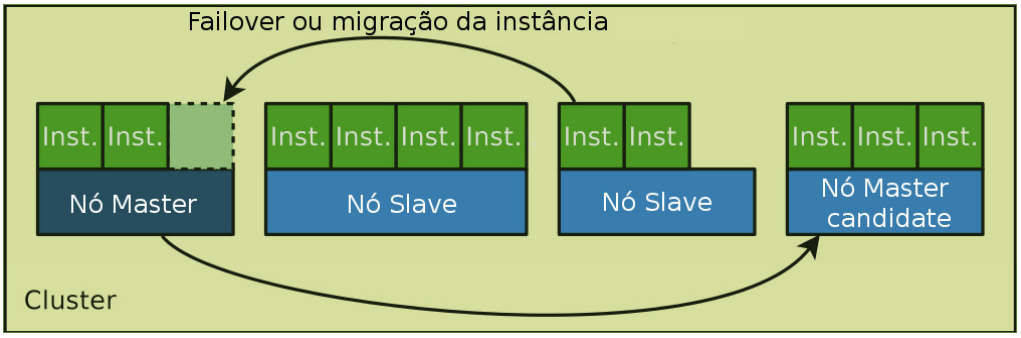
Após a detecção de uma falha, os *softwares* de gerenciamento de *cluster* executam operações de *failover* e *failback*. O *failover* é um processo no qual um outro servidor recebe os serviços que estavam executando no servidor que falhou. Já no *failback* tem-se o retorno dos serviços para o servidor de origem quando este estiver disponível. Esse processo ocorre após o *failover*, sendo que ele é opcional (BASSAN, 2008). Nas próximas seções tem-se uma breve descrição dos *softwares* de gerenciamento de *cluster* que foram estudados.

6.2.1 Ganeti

O *Ganeti* (Google, 2016) é um *software* desenvolvido pelo *Google*. Esse *software* é um gerenciador de *cluster* de virtualização. Ele foi desenvolvido especificamente para ambientes de virtualização e suporta os hipervisores KVM (OVA, 2016) e *Xen* (Citrix, 2016a).

Na Figura 6.4 tem-se um exemplo de um *cluster* com a arquitetura do *Ganeti*, sendo que este *cluster* é composto por um nó *master*, que armazena as configurações e gerencia o *cluster*, e um nó *master candidate*, para o caso do nó *master* falhar. Além disso, a arquitetura é formada por vários nós *slaves*. Destaca-se que no *Ganeti* todos os nós são responsáveis por prover o ambiente de virtualização e armazenar

os dados das VMs, sendo que cada nó pode possuir uma ou mais instâncias de VMs. Em cada instância das VMs configura-se dois nós, que são: o nó primário, que a instância da VM executará inicialmente; e o nó secundário, que será utilizado no caso de uma falha no nó primário. Além disso, as instâncias das VMs também podem ser migradas de um nó para outro de forma manual.



Fonte: CARVALHO; SOBRAL (2011)

Figura 6.4: Arquitetura do *Ganeti*.

De forma resumida, as principais funcionalidades do *Ganeti* são (Google, 2016):

- Criação de instâncias de VMs;
- Gerenciamento do armazenamento das instâncias;
- Iniciar e finalizar instâncias, além de efetuar a migração entre os nós.

6.2.2 Heartbeat

O *Heartbeat* é um subprojeto do *Linux-HA* (Linux-HA, 2016a), que desenvolve soluções de alta disponibilidade. Esse subprojeto é uma aplicação que envia pacotes *keepalive*[1] *UDP*, através da rede, para outras aplicações *Heartbeat*. Esses pacotes possuem como objetivo verificar se uma aplicação está ativa. Destaca-se que esse *software* pode ser utilizado para alta disponibilidade em ambientes de virtualização (REIS, 2009). Na Figura 6.5 tem-se o *Heartbeat* executando em dois servidores sobre as interfaces de rede dedicadas (identificadas na figura como *ethX*). Neste caso, se o nó secundário deixar de receber os sinais do nó primário, este irá tornar-se o nó primário, e iniciará o processo de *failover*.

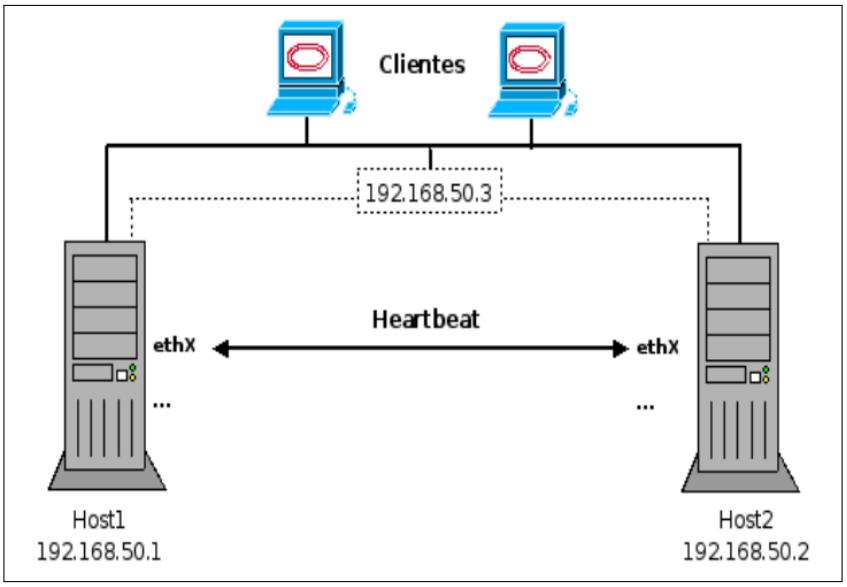
De forma resumida, as principais funcionalidades do *Heartbeat* são (ClusterLabs, 2016a):

- Enviar mensagens entre os nós para a detecção de falhas;
- Efetuar os processos de *failover* e *failback*;
- Iniciar e finalizar serviços nos nós;

6.2.3 Pacemaker

O *Pacemaker* (ClusterLabs, 2016b) é um projeto de código aberto mantido pela *ClusterLabs*, e teve origem com a necessidade do aperfeiçoamento do *Heartbeat* (Linux-HA, 2016b). O *Pacemaker* pode ser definido como um *software* de recuperação de falhas a nível de serviço (PERKOV; PAVKOVIć; PETROVIć, 2011).

[1] *Keepalive* significa mantenha vivo, são sinais enviados em uma determinada frequência para verificar se a comunicação esta ativa.



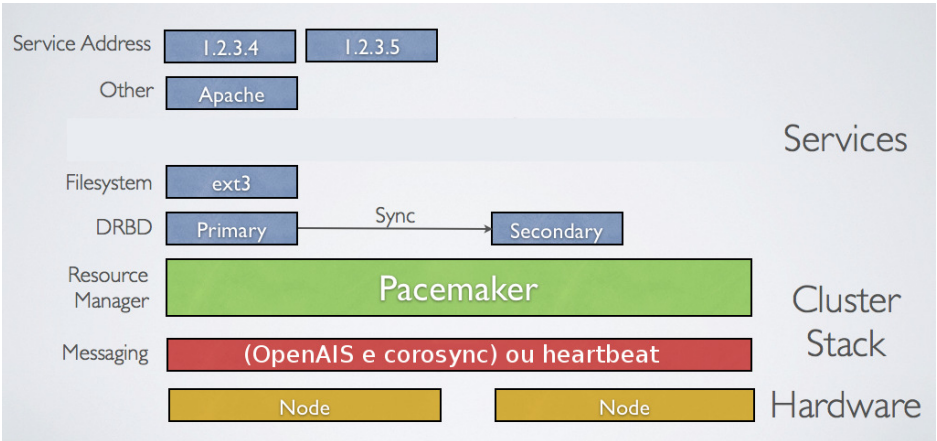
Fonte: ZAMINHANI (2008)

Figura 6.5: Arquitetura do *Heartbeat*.

Esse *software* é utilizado juntamente com outros *softwares* que fazem os registros dos nós e troca de mensagens entre os nós do *cluster*, sendo que os *softwares* que podem ser integradas com o *Pacemaker* são (ClusterLabs, 2016b):

- *Corosync* (Corosync, 2016): derivou do projeto *OpenAIS* e é responsável pelo processo de registro dos nós e pelo processo de *failover*;
- *Heartbeat*: responsável pelo envio de mensagens entre os nós do *cluster*, além de iniciar e finalizar serviços.

Na Figura 6.6 tem-se a arquitetura do *Pacemaker*. Como pode ser observado na camada inferior tem-se os nós do *cluster*. Na camada superior tem-se o *software* de envio de mensagens e acima deste tem-se o *Pacemaker*. Por fim, tem-se os serviços, que serão executados no *cluster*.



Fonte: ClusterLabs (2016b)

Figura 6.6: Exemplo da arquitetura do *Pacemaker*.

Entre as principais funcionalidades do *Pacemaker* destaca-se:

- Iniciar e finalizar os serviços dos nós do *cluster*: esses serviços podem ser desde um servidor *web*, uma interface de rede, ou até uma máquina virtual;
- Replicação de configuração do *cluster*: a configuração alterada em um nó é replicada para todos os nós de forma transparente;
- Eleição de um nó como primário: no caso de uma falha neste nó, um outro será eleito primário de forma automática.

No *Pacemaker* os serviços são denominados recursos (*resources*), esses recursos são monitorados, inicializados e parados. Pode-se criar dependências e uma ordem de inicialização entre esses recursos, para que esses sejam iniciados em uma determinada sequência. O *Pacemaker* também pode ser configurado para fazer o *failover*, desta forma caso ocorra uma falha em um nó, esse fará a inicialização dos serviços em um nó secundário. Além disso, este também faz a recuperação de um recurso que parou de executar por alguma falha, por exemplo, se ocorrer algum erro interno em um *software* e este for interrompido, o *Pacemaker* tentará iniciá-lo novamente.

6.2.4 Resumo e software de gerenciamento adotado

Na Tabela 6.2 tem-se uma comparação dos *softwares* de gerenciamento de *cluster*. O *software* escolhido para gerenciar o ambiente foi o *Pacemaker*, pois este possui todos os requisitos para a criação de um *cluster* de alta disponibilidade utilizando virtualização. A principal característica disponível neste é o *failover* automático dos nós, que não encontra-se disponível nas demais ferramentas. Além disso, essa ferramenta possibilita a migração em tempo real. Ela também faz o sincronismo de configuração entre todos os nós, ou seja, a configuração do *cluster* pode ser feita a partir de qualquer nó.

Critério	**Ganeti**	**Heartbeat**	**Pacemaker**
Suporte nativo à virtualização	Sim	Não	Sim
Migração de VMs em tempo real	Sim	Não	Sim
Failover e failback automáticos	Não	Não	Sim
Distribuições Linux	Todas	Red Hat, CentOS, Fedora, Suse, Debian, Ubuntu	Red Hat, CentOS, Fedora, Suse, Debian, Ubuntu

Tabela 6.2: Comparação entre ferramentas de gerenciamento de *cluster*.

O *software Ganeti* não se mostrou adequado pois não possui suporte de *failover* de forma automática. Seria possível também implementar uma solução de alta disponibilidade com a ferramenta *Heartbeat*. Porém, neste caso seria necessário criar um conjunto de *scripts* para a migração das máquinas virtuais.

6.3 Considerações finais

Neste capítulo foi feito um levantamento de *softwares* para possibilitar a construção de um ambiente de alta disponibilidade utilizando virtualização. Para construção desse ambiente dois *softwares* foram necessários, um para a replicação de dados e outro para gerência e monitoramento do *cluster*. Foram utilizados alguns critérios para auxiliar na escolha dos *softwares*, sendo assim, os *softwares* escolhidos foram o DRBD para fazer a replicação dos dados, e o *Pacemaker* que é responsável pela gerência e monitoramento. No próximo capítulo será apresentado a descrição

da implementação efetuada e posteriormente serão descritos os testes criados e apresentado os resultados obtidos.

7 IMPLEMENTAÇÃO E TESTES

Neste capítulo será descrita a implementação do ambiente de alta disponibilidade na empresa. Posteriormente será apresentada a metodologia de testes utilizada para validar desse ambiente criado, bem como os resultados obtidos dos testes efetuados.

7.1 Implementação

O ambiente foi projetado na forma de um *cluster*, o qual é composto por dois servidores com requisitos de configuração de 12 *cores* de processamento, 14 GB de memória RAM e 180 GB de disco rígido. Essa configuração foi calculada a partir da soma dos recursos consumidos pelas máquinas virtuais que atualmente executam os serviços que foram considerados críticos. Observa-se que tais recursos de *hardware* já encontravam-se disponíveis, sendo necessário somente efetuar uma reorganização das máquinas virtuais. Destaca-se que a configuração também inclui 2 GB de memória RAM e 24 GB de disco para cada sistema operacional hóspede.

Além disso, optou-se por utilizar o mesmo sistema operacional e o mesmo hipervisor que são adotados atualmente na empresa, que são, respectivamente, o sistema *Ubuntu 14.04 LTS* e o KVM (OVA, 2016). O processo de instalação e de configuração encontra-se no Apêndice A.1 e A.4.

A estrutura física adotada encontra-se na Figura 7.1 (a). Como pode ser observado, os dois servidores encontram-se conectados a um *switch* através de dois cabos UTP (*Unshielded Twisted Pair*), de forma a implementar uma redundância de cabeamento. Além disso, manteve-se o *link aggregation* e utilizou-se uma *bridge*[1] para incluir as máquinas virtuais à rede. Os detalhes da configuração de rede estão localizados no Apêndice A.2. Na Figura 7.1 (b) tem-se a imagem dos servidores, o primeiro é o Nó 1 (*Brina*), que é um *Dell PowerEdge 2950*, e o segundo servidor é o Nó 2 (*Piova*), que é um *Dell PowerEdge R410*.

A estrutura lógica dos servidores juntamente com as máquinas virtuais e seus respectivos serviços são apresentados na Figura 7.2. Para a replicação de dados foi utilizado o *software* DRBD, que foi configurado no modo *dual-master* onde os dois nós são configurados como primários. Para tal configuração é necessário utilizar um sistema de arquivos distribuídos para um acesso compartilhado dos dados, sendo que o *software* adotado foi o OCFS2 (Oracle, 2016c). Os detalhes da instalação e da configuração do DRBD e do sistema de arquivos OCFS2 estão detalhadas no Apêndice A.3. Já para o gerenciamento do *cluster* e das VMs utilizou-se o *software*

[1] *Bridges*, também conhecidas como pontes, são meios que fazem a conexão entre *LANs*, e operam na camada de enlace de dados.

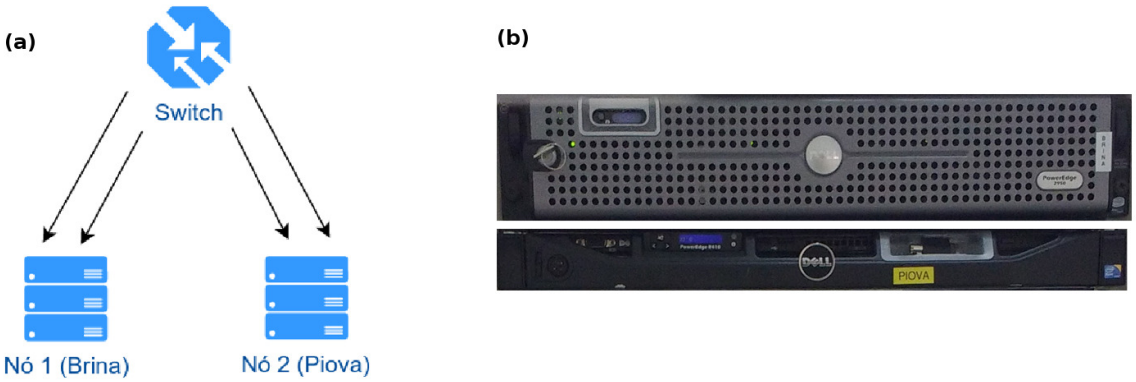


Figura 7.1: Estrutura física.

Pacemaker. Mais especificamente esse é o *software* responsável pelo monitoramento e pela migração das VMs entre os nós.

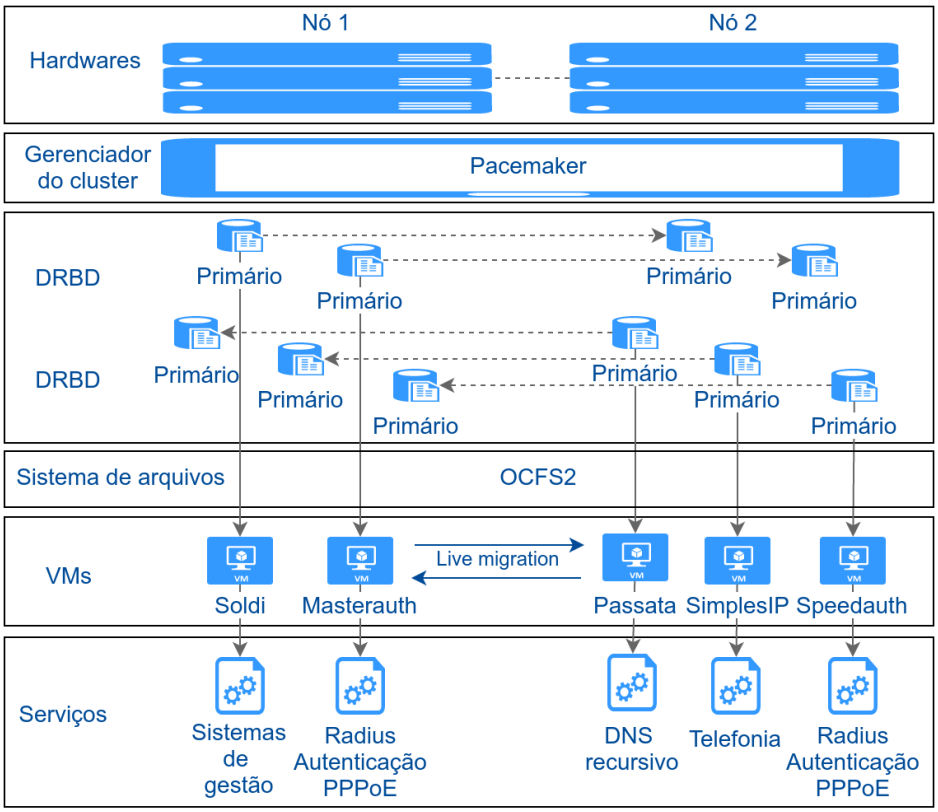


Figura 7.2: Estrutura do *cluster*.

Os serviços foram distribuídos entre os nós dos *cluster*, sendo que no Nó 1 foram instalados os serviços de sistemas de gestão, DNS recursivo e autenticação PPPoE. Já no Nó 2 foram instalados os serviços de telefonia e autenticação PPPoE. No caso de falha em um nó, os serviços serão inicializados no outro nó disponível. O detalhamento da configuração do *Pacemaker* estão disponíveis no Apêndice A.5.

7.2 Testes

A metodologia de testes deste trabalho foi baseada nos trabalhos de REIS (2009) e GONçALVES (2009). No primeiro trabalho, o autor simulou falhas de três formas, que foram através do desligamento brusco do servidor, desligamento por *software* e falha de rede. Já no segundo trabalho, foram realizados três testes, que foram a reinicialização do servidor físico, a migração de uma máquina virtual de forma manual, e por fim um teste com migração em tempo real (*live migration*).

Neste trabalho optou-se por utilizar o teste de desligamento brusco e o desligamento por *software* utilizados por REIS (2009). Já do autor GONçALVES (2009), utilizou-se o teste de migração em tempo real, porém adaptado para simular um agendamento de manutenção de *hardware* ou de *software*, ou seja, migrar as VMs de um nó para outro utilizando a migração em tempo real, com objetivo de fazer uma manutenção do sistema operacional deste nó.

7.2.1 Teste 1 - Desligamento físico

Neste teste são simuladas falhas de *hardware* ou falhas elétricas nos nós do *cluster*. Para isso foi feito o desligamento forçado do servidor, ou seja, foi pressionado o botão *power* até o servidor desligar. Com esse teste pode-se validar ainda o processo de *failover* dos serviços (máquinas virtuais) que estavam executando no nó que falhou, bem como medir o tempo de indisponibilidade dos serviços. Esse teste de desligamento foi executado 4 vezes no *cluster*. O tempo de indisponibilidade foi medido utilizando o comando *ping* (para facilitar a medição foi utilizado um *script* que está localizado no Apêndice B.1), ou seja, durante a execução de cada teste foi executado um *ping* para os dois nós e para a máquina virtual, sendo que cada execução possui duração de 5 minutos.

Com os resultados desses testes efetuados criou-se a Tabela 7.1, onde pode-se observar que o tempo de indisponibilidade da máquina virtual é de apenas 81 segundos. Isso se deve ao fato da VM ser iniciada em um outro nó após o desligamento do primeiro. Caso um servidor de virtualização que não possua esta solução de alta disponibilidade for reiniciado por causa de uma falha elétrica por exemplo, o tempo para a recuperação do serviço seria igual a soma do tempo de inicialização do servidor físico mais o tempo de inicialização da máquina virtual, que totalizaria aproximadamente 250 segundos. Na pior das hipóteses, caso haja uma falha definitiva do servidor físico, sendo necessário reconfigurar a máquina virtual, reinstalar as aplicações, configurá-las e restaurar o *backup*, o MTTR seria significativamente maior. Dependendo do servidor e da aplicação, a indisponibilidade poderia ser maior que 24 horas.

Pode-se observar na tabela uma diferença entre os tempos de indisponibilidade do Nó 1 e Nó 2, isso ocorre devido a utilização de *hardwares* diferentes na implementação. Além disso, o desvio padrão demonstra que existe pouca variação entre os tempos de indisponibilidade.

7.2.2 Teste 2 - Desligamento por software

Esse teste simula falhas de *software* nos nós do *cluster*, desta forma pode-se citar como exemplo uma falha de um *software* de virtualização, com isso o nó não conseguiria iniciar a máquina virtual. Esse tipo de situação também pode ocorrer em uma manutenção emergencial, onde é preciso desligar um servidor físico de forma

	Tempo de indisponibilidade médio	**Desvio padrão da indisponibilidade**
Nó 1	124	4,24
Nó 2	170,5	6,36
Máquina virtual	81	7,07

Tabela 7.1: Resultados do teste de desligamento físico, contendo o tempo de indisponibilidade em segundos e o desvio padrão.

rápida. Desta forma, pode-se verificar a capacidade do outro nó manter os serviços em funcionamento. Para simular esta situação, foram acessados os nós via SSH e executado o comando *reboot*. Esse teste foi executado 4 vezes nos nós do *cluster*. O tempo de indisponibilidade foi medido da mesma forma que o teste da seção anterior, utilizando o comando *ping*. A medição desse tempo foi feita nos dois nós e também na máquina virtual para possibilitar a análise e comparação dos tempos. A Tabela 7.2 demonstra o tempo de indisponibilidade e o desvio padrão dos nós e da VM. Pode-se observar que o tempo de indisponibilidade da VM é consideravelmente menor que o tempo do servidor físico. De fato, ele representa apenas 3,27% do tempo de indisponibilidade dos servidores físicos que são 138,5 e 167 segundos.

A alguns meses ocorreu um problema semelhante a este teste, um servidor de virtualização, que é atualizado de forma automática, foi reiniciado, porém ocorreu um erro na atualização e o servidor não pôde iniciar corretamente. Os serviços executados nele ficaram aproximadamente 6 horas indisponíveis.

	Tempo de indisponibilidade média	**Desvio padrão da indisponibilidade**
Nó 1	138,5	2,12
Nó 2	167	1,41
Máquina virtual	10	0

Tabela 7.2: Resultados do teste de desligamento por *software*, com tempo de indisponibilidade em segundos e o desvio padrão.

7.2.3 Teste 3 - Manutenção agendada

Reinicializações são necessárias para manutenções de *hardware*, atualização de *software* e até mesmo para rejuvenescimento[1] de *software* (MELO, 2014). Desta forma, criou-se esse teste para simular manutenções previamente agendadas.

Esse teste consiste no agendamento de 4 manutenções efetuadas durante o período de 13 dias, para tanto, criou-se um *script* que é responsável por migrar as VMs para liberar o nó e reiniciá-lo, simulando assim uma atualização de *kernel* (este *script* está disponível no Apêndice B.2). Para a execução do *script* foi utilizada a ferramenta *crontab* do *Linux*. Os resultados são apresentados na Tabela 7.3, pode-se observar que não houve *downtime* na máquina virtual, e consequentemente, não houve indisponibilidade nos serviços, devido a migração em tempo real das VMs. Já nos nós

[1]O rejuvenescimento de *software* consiste na aplicações de métodos para remover problemas gerados pelo envelhecimento. Um exemplo de um método é o *reboot* de um sistema operacional, uma vez que esse torna-se suscetível a gerar erros e falhas.

do *cluster* tem-se um tempo de indisponibilidade de 145,5 e 173 segundos devido a reinicialização dos nós.

	Tempo de indisponibilidade média	**Desvio padrão da indisponibilidade**
Nó 1	145,5	0,70
Nó 2	173	1,41
Máquina virtual	0	0

Tabela 7.3: Resultados do teste de manutenção agendada, com o tempo de indisponibilidade em segundos e o desvio padrão.

A Tabela 7.4 demonstra o percentual de disponibilidade da VM e dos nós do *cluster* durante o período de 13 dias do teste. Destaca-se que a máquina virtual não apresentou nenhuma indisponibilidade. Esses dados foram produzidos pela ferramenta *Nagios* (Nagios, 2016), responsável pelo monitoramento da empresa. COLOCAR ESSA TABELA ??

	Disponibilidade	**Tempo indisponibilidade**
Nó 1	99,976%	4 minutos e 30 segundos
Nó 2	99,975%	4 minutos e 40 segundos
Máquina virtual	100%	0 minutos

Tabela 7.4: Disponibilidade e tempo de indisponibilidade do período de 13 dias.

Durante o processo de *live migration* de um nó para outro a latência da máquina virtual aumentou, isso pode ser observado na Figura 7.3 através do pico ocorrido entre o tempo de 50 e 100 segundos.

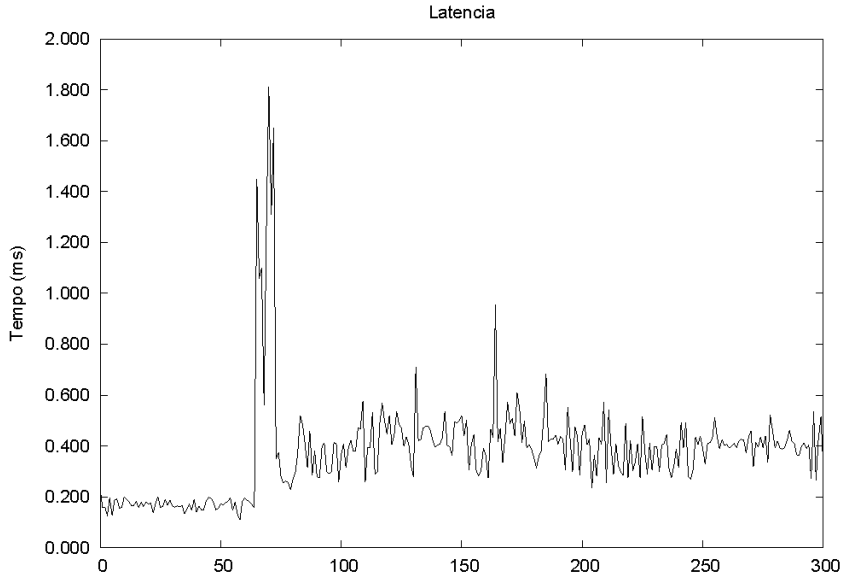


Figura 7.3: Latência da máquina virtual durante o *live migration*.

7.3 Comparação final da disponibilidade

Nesta seção será feita a medição e a análise do período de um mês no ambiente de alta disponibilidade, sendo que neste ambiente estarão executando os serviços críticos definidos na Seção 5.5. Durante o período foi feito uma manutenção ...

continuar ... ??

adicionar gráficos de disponibilidade final dos serviços críticos:

7.4 Considerações finais

Neste capítulo foi apresentada a arquitetura de alta disponibilidade implementada com base em virtualização, sendo composta por dois servidores físicos, onde foram configuradas máquinas virtuais contendo os serviços críticos da empresa.

Foi feita uma análise do ambiente de alta disponibilidade implementado na empresa, através de testes e da coleta de dados provenientes desses. Pode-se perceber que resultados obtidos foram positivos, e que os objetivos deste trabalho foram alcançados. ...

8 CONCLUSÃO

Neste trabalho foi feito um estudo sobre uma empresa prestadora de serviços para Internet, analisando sua estrutura física e os serviços oferecidos por esta. Durante este estudo foram definidos os serviços críticos. Para tanto considerou-se o impacto dos mesmos para a empresa, medido através da quantidade de clientes e funcionários que utilizam o serviço. Destaca-se que foi necessário selecionar os serviços mais críticos, por não haver recursos necessários para implementar a alta disponibilidade para todos os serviços.

Deste modo, os serviços críticos definidos foram: o serviço de DNS recursivo, pois é utilizado para navegação de Internet de todos os clientes e funcionários do provedor; o serviço de autenticação *Radius*, por influenciar diretamente na navegação dos clientes e armazenar dados importantes para o provedor; sistemas da empresa, uma vez que todos os funcionários utilizam-nos e também por ter um impacto indireto nos clientes; e serviço de telefonia interna, por ser responsável pela comunicação tanto entre funcionários, como entre clientes e funcionários.

Após criou-se uma proposta de alta disponibilidade para esses serviços. Essa proposta é composta por um *cluster* o qual é constituído por dois servidores físicos. Para o gerenciamento do *cluster* foi adotado o *software Pacemaker*, que é responsável pelo gerenciamento, monitoramento e a transferência das máquinas virtuais, as quais executam os serviços. Para a replicação de dados do *cluster* foi adotado o *software* DRBD. Esse é responsável pela replicação dos dados dos dois servidores.

O ambiente de alta disponibilidade é composto por máquinas virtuais que executam os serviços críticos. Para garantir a disponibilidade foi utilizada a opção de migração em tempo real, fornecida pelo hipervisor KVM, juntamente com o *Pacemaker*. Desta forma, caso um dos servidores falhe as máquinas virtuais serão migradas para o outro servidor.

Ao final do trabalho, criou-se uma metodologia de testes para validar a solução de alta disponibilidade. Esses testes simulam falhas de *hardware*, de energia elétrica, de *software* e manutenções previamente agendadas foram implementados e aplicados, com isso foi comprovado que a disponibilidade dos serviços críticos foi aumentada. Desta forma, é possível concluir que é possível aumentar a disponibilidade de serviços em máquinas virtuais utilizando uma solução de código aberto e de baixo custo.

8.1 Trabalho futuros

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.

AREND, G. M. **MONITORAMENTO VIA SNMP PARA REDES DE FIBRA ÓPTICA DO TIPO FTTH**. 2014. Trabalho de Conclusão (Bacharel em Sistemas de Informação) — Universidade de Caxias do Sul, Caxias do Sul - Rio Grande do Sul.

ATS, G. **Entenda o que são coolers e fans**. <Disponível em: http://www.techtudo.com.br/artigos/noticia/2012/01/entenda-o-que-sao-coolers-e-fans.html>. Acesso em 19 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.

BARRETT, D.; SILVERMAN, R.; BYRNES, R. **SSH, The Secure Shell**: the definitive guide. [S.l.]: O'Reilly Media, 2005.

BASSAN, R. **Avaliação de cluster de alta disponibilidade baseado em software livre**. 2008. Trabalho de Conclusão (Curso de Ciência da Computação) — Faculdade de Jaguariúna, Jaguariúna - São Paulo.

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.

CARVALHO, G.; SOBRAL, D. **Assuma o controle do seu parque virtualizado**. <Disponível em: http://gutocarvalho.net/wordpress/wp-content/uploads/2011/04/Palestra-Ganeti-Puppet.pdf>. Acesso em 26 de junho de 2016.

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

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

Corosync. **Corosync by corosync**. <Disponível em: http://corosync.github.io/corosync/>. Acesso em 25 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.

DAVIES, A.; ORSARIA, A. Scale out with GlusterFS. **Linux J.**, Houston, TX, v.2013, n.235, nov 2013.

DAVISON, W. **rsync**. <Disponível em: https://rsync.samba.org/>. Acesso em 20 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, Brasília.

Google. **Ganeti**. <Disponível em: `http://docs.ganeti.org/`>. Acesso em 25 de junho de 2016.

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.

JONES, M. T. **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.

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.

KUROSE, J. F.; ROSS, K. W. **Redes de computadores e a Internet**. 3.ed. São Paulo: Pearson Addison Wesley, 2006.

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.

LINBIT. **DRBD brings you High Availability and Disaster Recovery**. <Disponível em: `http://www.drbd.org/`>. Acesso em 21 de junho de 2016.

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.

LOPEZ, P. **Managed Load-balancing / Server Mirroring Solutions**. <Disponível em: `https://www.gidforums.com/t-27040.html`>. Acesso em 29 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.

MELO, M. D. T. de. **MODELOS DE DISPONIBILIDADE PARA NUVENS PRIVADAS**: rejuvenescimento de software habilitado por agendamento de migraÇÃo de vms. 2014. Pós-Graduação em Ciência da Computação — Universidade Federal de Pernambuco, Recife.

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.

NIC.br. **IPv6.br**. <Disponível em: `http://ipv6.br/`>. Acesso em 21 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.

Oracle. **Project**: ocfs2. <Disponível em: `https://oss.oracle.com/projects/ocfs2/`>. Acesso em 25 de junho de 2016.

OVA. **KVM**. <Disponível em: `http://www.linux-kvm.org/page/Main_Page`>. Acesso em 25 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. **Plataforma corporativa Linux**. <Disponível em: https://www.redhat.com/pt-br/technologies/linux-platforms>. Acesso em 05 de junho de 2016.

Red Hat. **Red Hat GFS**. <Disponível em: https://access.redhat.com/documentation/pt-BR/Red_Hat_Enterprise_Linux/5/html/Cluster_Suite_Overview/s1-rhgfs-overview-CSO.html>. Acesso em 25 de junho de 2016.

Red Hat. **Storage for your Cloud - Gluster**. <Disponível em: https://www.gluster.org/>. Acesso em 21 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.

TANENBAUM, A. S.; WETHERALL, D. **Redes de computadores**. 5.ed. São Paulo: Pearson Prentice Hall, 2011.

TANENBAUM, A.; WOODHULL, A. **Sistemas Operacionais**: projetjos e implementação. [S.l.]: Bookman, 2009.

TECHNOLOGIES, J. **Network Protocols Handbook**. [S.l.]: Javvin Technologies, 2005.

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.

XSF. **XMPP**. <Disponível em: https://xmpp.org/>. Acesso em 21 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.

ApêndiceA

A.1 Configuração do OS

Foi feita a instalação do sistema operacional *Ubuntu 14.04 LTS* nos dois servidores. A configuração feita foi a básica do sistema, com nome do servidor, configuração de rede, localização e instalação do servidor SSH. Além disso, é feito as configurações padrões adotadas pela empresa, como por exemplo, ferramentas de monitoramento, atualização automática e *firewall*.

A.2 Configuração de rede

Uma forma de incluir as máquinas virtuais a uma rede é através da criação de uma *bridge*. A instalação é feita através do comando:

```
$ apt-get install bridge-utils
```

Além disso, é necessário instalar e configurar o *link aggregation*, o *Ubuntu* usa o padrão *IEEE 802.3ad*. A configuração é feita no servidor e no *switch*. Abaixo tem-se os comandos para instalação do *link aggregation*, o primeiro comando faz a instalação e o segundo comando carrega o modulo *bonding* no *kernel*:

```
$ apt-get install ifenslave-2.6
$ sh -c 'grep -q bonding /etc/modules \
    || echo bonding >> /etc/modules'
```

Para a configuração do *link aggregation* deve-se criar uma interface *bond* e nela inclui-se as duas interfaces físicas. Abaixo tem-se a configuração de rede, que inclui o *link aggregation* e a *bridge*, esta configuração deve ser colocada no arquivo */etc/network/interfaces* em cada nó:

```
auto eth0
iface eth0 inet manual
bond-master bond0

auto eth1
iface eth1 inet manual
bond-master bond0

auto bond0
iface bond0 inet manual
       bond-mode 4
       bond-slaves eth0 eth1
       bond-lacp-rate 1
       bond-miimon 100
       bond-xmit_hash_policy layer3+4

auto br0
iface br0 inet static
       address x.x.x.x
```

```
        netmask 255.255.255.0
        network x.x.x.0
        broadcast x.x.x.255
        gateway x.x.x.x
        dns-nameservers 8.8.8.8 8.8.4.4

        bridge_ports bond0
        bridge_stp off
        bridge_maxwait 0
```

Também é necessário configurar algumas *vlans* para algumas máquinas virtuais. Os comandos a seguir fazem a instalação e carregam o modulo do *kernel* necessário para gerenciar *vlans*:

```
$ apt-get install vlan
$ sudo  sh -c 'grep -q 8021q /etc/modules \
   || echo 8021q >> /etc/modules'
$ sudo modprobe 8021q
```

A.3 Configuração de disco

Os discos que são replicados podem ser utilizados diretamente no DRBD (por exemplo */dev/sda2*) ou configurados com *Logical Volume Manager* (LVM). Adotou-se a configuração utilizando LVM pois torna-se mais fácil a manipulação dos discos. O comando abaixo faz a criação de um volume lógico chamado *lvdrbd* com tamanho de 500 GB, este volume pertence ao grupo de volumes *vg0*.

```
$ lvcreate -n lvdrbd vg0 -L 500G
```

Para a instalação do *software* de replicação de dados DRBD é necessário instalar os dois pacotes, que estão listados abaixo:

```
$ apt-get install drbd8-utils drbdlinks
```

É necessário alterar a configuração global do DRBD que está localizada em */etc/drbd.d/global_common.conf*:

```
global {
        usage-count yes;
        minor-count 16;
}
```

Posteriormente, deve-se criar um recurso que definirá os dispositivos de disco, os endereços IP e portas dos servidores. Deve-se criar o arquivo */etc/drbd.d/vms.res*, o qual armazena essa configuração:

```
resource vms {
   meta-disk internal;
   device /dev/drbd0;
   protocol C;
   disk {
       fencing resource-only;
       resync-rate 50M;
   }
   handlers {
       fence-peer "/usr/lib/drbd/crm-fence-peer.sh";
       after-resync-target "/usr/lib/drbd/crm-unfence-peer.sh";
   }
   net {
       allow-two-primaries;
   }
   startup {
       become-primary-on both;
   }
   on brina {
       address x.x.x.x:7791;
       disk /dev/vg0/lvdrbd;
```

```
    }
    on piova {
        address y.y.y.y:7791;
        disk /dev/vg0/lvdrbd;
    }
}
```

Para o funcionamento correto dessa ferramenta, o nome dos servidores (localizado em */etc/hostname*) deve ser exatamente igual a opção *on* da configuração do recurso.

Após ter sido feita a configuração do DRBD, deve-se reiniciar o serviço para aplicá-la:

```
$ service drbd restart
```

Os comandos abaixo, inicializam os discos do DRBD, conectam ao outro nó e elegem um nó como primário, respectivamente, para iniciar o sincronismo dos dados:

```
$ drbdadm create-md vms
$ drbdadm up vms
$ drbdadm -- --overwrite-data-of-peer primary vms
```

E por fim pode-se verificar o estado dos recursos do DRBD através do comando:

```
$ service drbd status
```

O OCFS2 é um sistema de arquivos para *cluster*, ele faz o gerencimento do acesso aos dados (*locks*) que é necessário para o funcionamento do *Pacemaker*. Para seu funcionamento é necessário a instalação de um pacote:

```
$ apt-get install ocfs2-tools
```

É preciso fazer duas configurações, a primeira para criar o *cluster*, localizada no arquivo */etc/ocfs2/cluster.conf*:

```
node:
        ip_port = 7777
        ip_address = x.x.x.x
        number = 0
        name = brina
        cluster = clusterocfs
node:
        ip_port = 7777
        ip_address = y.y.y.y
        number = 1
        name = piova
        cluster = clusterocfs
cluster:
        node_count = 2
        name = clusterocfs
```

A segunda configuração é feita no arquivo */etc/default/o2cb*, ela faz o carregamento do *driver* no *boot* e a definição do nome do cluster para inicialização:

```
O2CB_ENABLED=true
O2CB_BOOTCLUSTER=clusterocfs
```

Para aplicar as configurações deve-se reiniciar o serviço:

```
$ service ocfs2 restart
```

E por fim, deve-se construir o sistema de arquivos no dispositivo criado pelo DRBD, através do comando:

```
$ mkfs.ocfs2 /dev/drbd/by-res/vms
```

Na versão 1.6.4 do pacote *ocfs2-tools* no *Ubuntu 14.04* há um *bug*[1] que impede o *Pacemaker* de montar o sistema de arquivo OCFS2. Para corrigi-lo deve aplicar o *patch* abaixo no arquivo */usr/lib/ocf/resource.d/heartbeat/Filesystem*:

[1]Detalhes do *bug*: https://bugs.launchpad.net/ubuntu/+source/ocfs2-tools/+bug/1412438

```
--- Filesystem 2013-12-16 07:41:25.000000000 +0000
+++ Filesystem.new 2015-01-19 19:01:30.181772112 +0000
@@ -338,7 +338,7 @@ ocfs2_init()
  # not need this:
  OCFS2_SLES10=""
- if [ "X$HA_cluster_type" = "Xcman" ]; then
+ if [ "X$HA_cluster_type" = "Xcorosync" ]; then
      return
   elif [ "X$HA_cluster_type" != "Xopenais" ]; then
    if grep -q "SUSE Linux Enterprise Server 10" /etc/SuSE-release >/dev/null 2>&1 ; then
```

A.4 Configuração do ambiente virtualizado

Como já mencionado anteriormente o hipervisor utilizado pela empresa é o KVM. Para a virtualização das máquinas também é utilizado a API *libvirt*, que faz o gerenciamento da virtualização. O comando abaixo faz a instalação dessas ferramentas:

```
$ apt-get install qemu-kvm libvirt-bin
```

Para incluir as máquinas virtuais é necessário criar um *pool*, onde serão armazenadas suas imagens, os comandos abaixo criam a pasta e o *pool*:

```
$ mkdir /var/lib/libvirt/images/ocfs
$ virsh pool-create-as ocfs --type=dir \
  --target=/var/lib/libvirt/images/ocfs
```

Para o funcionamento do *live migration* é preciso fazer a troca das chaves SSH do usuário root entre os dois nós. No nó *Brina*, o primeiro comando cria uma chave *rsa* e o segundo copia a chave para o outro nó:

```
$ ssh-keygen
$ ssh-copy-id piova
```

E no nó *Piova*, deve-se efetuar o mesmo processo:

```
$ ssh-keygen
$ ssh-copy-id brina
```

A.5 Configuração do cluster Pacemaker

O *Pacemaker* faz monitoramento dos nós do *cluster* e iniciará ou finalizará os serviços (recursos) em caso de falhas de um nó. No *Pacemaker*, os serviços que são configurados para serem monitorados são chamados de recursos (*resources*).

A primeira etapa é a instalação da ferramenta:

```
$ apt-get install pacemaker corosync
```

Após deve-se configurar o *corosync* para fazer o monitoramento do *cluster*, para isso deve editar o arquivo */etc/corosync/corosync.conf*:

```
totem {
        version: 2
        token: 3000
        token_retransmits_before_loss_const: 10
        join: 60
        consensus: 3600
        vsftype: none
        max_messages: 20
        clear_node_high_bit: yes
        secauth: on
        threads: 0
        rrp_mode: none
        interface {
                member {
                        memberaddr: x.x.x.x
```

```
                }
                member {
                        memberaddr: y.y.y.y
                }
                ringnumber: 0
                bindnetaddr: x.x.x.0
        }
        transport: udpu
}
amf {
        mode: disabled
}
quorum {
        # Quorum for the Pacemaker Cluster Resource Manager
        provider: corosync_votequorum
        expected_votes: 2
        two_node: 1
}
service {
        # Load the Pacemaker Cluster Resource Manager
        ver:       0
        name:      pacemaker
}
aisexec {
        user:   root
        group:  root
}
logging {
        fileline: off
        to_stderr: yes
        to_logfile: no
        to_syslog: yes
        syslog_facility: daemon
        debug: off
        timestamp: on
        logger_subsys {
                subsys: AMF
                debug: off
                tags: enter|leave|trace1|trace2|trace3|trace4|trace6
        }
}
```

Ajustar para o *corosync* inicie com o sistema, através da alteração da configuração em */etc/default/corosync*:

```
START=yes
```

Para a comunicação entre os nós deve-se criar uma chave e copiá-la para o outro nó:

```
$ corosync-keygen
$ scp /etc/corosync/authkey piova:/etc/corosync/
```

E por fim deve-se reiniciar os serviços:

```
$ service corosync restart
$ service pacemaker restart
```

Para verificar o estado os *cluster*:

```
$ crm status
```

Após os nós estarem *online* deve-se criar os recursos do *Pacemaker*. Para alterar a configuração pode utilizar os diversos comandos do *crm*, ou pode-se visualizar e alterar toda configuração através do comando:

```
$ crm configure edit
```

Primeiramente deve-se criar o recurso para o DRBD. Esse recurso faz o monitoramento e inicia os dispositivos nos nós do DRBD. A configuração é:

```
primitive DRBD ocf:linbit:drbd \
        params drbd_resource="vms" \
        op monitor interval="20" role="Master" timeout="240" \
        op monitor interval="30" role="Slave" timeout="240" \
        meta is-managed="true" target-role="Started"
ms MS_DRBD DRBD \
        meta master-max="2" clone-max="2" notify="true" \
        interleave="true" allow-migrate="true" is-managed="true"
```

Onde *primitive* é o recurso DRBD configurado anteriormente, e *ms* é a opção que fará o dispositivo se tornar *master* sendo limitado em 2.

Também é necessário criar um recurso para montar um sistema de arquivos OCFS2, configurado da seguinte forma:

```
primitive OCFS_MOUNT ocf:heartbeat:Filesystem \
        params device="/dev/drbd/by-res/vms" \
        directory="/var/lib/libvirt/images/ocfs" fstype="ocfs2"
clone OCFS_MOUNT_CLONE OCFS_MOUNT \
        meta interleave="true" ordered="true" target-role="Started"
colocation COL_DRBD_OCFS inf: OCFS_MOUNT_CLONE MS_DRBD:Master
order ORD_DRBD_OCFS inf: MS_DRBD:promote OCFS_MOUNT_CLONE:start
```

Onde *primitive* define qual é o dispositivo e onde está montado, o recurso *clone* permite montar o sistema nos dois nós ao mesmo tempo. Já o *colocation* faz a dependêcia entre os recursos do *Pacemaker*, por exemplo, é necessário ter o dispositivo DRBD como *master* para depois montar o sistema de arquivos. E o recurso *order* identifica a ordem de inicialização dos recursos, da equerda para direita.

E por fim, deve-se configurar um recurso para cada máquina virtual no *Pacemaker* para possibilitar que as máquinas virtuais sejam iniciadas ou migradas de um nó para outro de forma automática:

```
primitive VM_1 ocf:heartbeat:VirtualDomain \
        params config="/etc/libvirt/qemu/vm1.xml" force_stop="0" \
        hypervisor="qemu:///system" migration_transport="ssh" \
        op monitor timeout="30" interval="10" depth="0" \
        op start timeout="90" interval="0" \
        op stop timeout="90" interval="0" \
        op migrate_from interval="0" timeout="240" \
        op migrate_to interval="0" timeout="240" \
        meta allow-migrate="true" target-role="Started" \
        is-managed="true" migration-threshold="5"
location cli-prefer-VM_1 VM_1 inf: brina
location backup-VM_1 VM_1 100: piova
colocation COL_VM_1 inf: VM_1 OCFS_MOUNT_CLONE
order ORD_VM_1 inf: OCFS_MOUNT_CLONE:start VM_1:start
```

ApêndiceB

B.1 Script indisponibilidade

```
#!/bin/bash
# Copyright (c) 2008 Wanderson S. Reis <wasare@gmail.com>
#
# Adaptado para gerar estatisticas do ping.
# Bruno Emer <emerbruno@gmail.com> 2016
#
# parametros :
#endereco IP e numero de sequencia do teste para controle
# ( opcional )
#
IP=$1

ping $1 > $1-stat-$2.log &
PINGPID=$!

INITMP=`date +%s`
RETORNO=0
while [ $RETORNO -eq 0 ]
do
        ping -c 1 $1 2> /dev/null 1> /dev/null
        RETORNO=$?

        TIMECUR=$(( `date +%s` - $INITMP ))
        if [ $TIMECUR -gt 90 ]; then
                break;
        fi
done
INICIAL=`date +%s`
while [ $RETORNO -ne 0 ]
do
        ping -c 1 $1 2> /dev/null 1> /dev/null
        RETORNO=$?
done
FINAL=`date +%s`
INTERVALO=$(( $FINAL - $INICIAL ))
echo "host $1 : $INTERVALO s" > $1-downtime-$2.log

while :; do
        TIMECUR=$(( `date +%s` - $INITMP ))
        if [ $TIMECUR -gt 300 ]; then
                break;
        fi
done

#pkill -SIGINT ping
kill -SIGINT $PINGPID

exit 0
```

B.2 Script manutenção Pacemaker

```
#!/bin/bash
```

```
# ------------------------------------------------------------------
# Copyright (c) 2016-09-02 Bruno Emer <emerbruno@gmail.com>
#
# Verifica e faz reinicializacao de um node do cluster pacemaker / drbd
# Instalacao:
# Agendar no cron de cada node em dias diferentes
#
# ------------------------------------------------------------------

PROGNAME=`/usr/bin/basename $0`
NODE=`hostname`
ONLINE_CHECK_RES="OCFS_MOUNT_CLONE"
STANDBY_CHECK_RES="MS_DRBD"

logger "Script manutencao cluster $PROGNAME node: $NODE"

# ------------------------------------------------------------------
#recovery - retorna node para online se ja foi reiniciado
if [ -f pacemaker_reboot.tmp ]; then
        #verifica se pacemaker esta iniciado
        service pacemaker status
        PACEMAKER_ST=$?
        if [ $PACEMAKER_ST -ne 0 ]; then
                logger "Inicia pacemaker"
                service pacemaker start
        fi
        service pacemaker status
        PACEMAKER_ST=$?
        if [ $PACEMAKER_ST -eq 0 ]; then
                logger "Online $NODE"
                /usr/sbin/crm node online $NODE
                rm pacemaker_reboot.tmp
        fi
else
# ------------------------------------------------------------------
#destroy - derruba node para reiniciar
        #verificacoes
        #se cluster esta ok
        NAG_CHECK=$(/usr/lib/nagios/plugins/check_crm_v0_7)
        NAG=$?
        logger "Cluster nagios check: $NAG_CHECK"
        if [ $NAG -ne 0 ]; then
                logger "Erro. Cluster nao esta ok"
                exit
        fi

        #se ja reiniciou nao executa
        UPTIME=`awk '{print $1}' /proc/uptime`
        UPINT=${UPTIME%.*}
        if [ $UPINT -lt 86400 ]; then
                logger "Ja reiniciado"
                exit
        fi

        #se outro node esta online
        RES_CHECK=$(/usr/sbin/crm resource show $ONLINE_CHECK_RES)
        NODE_LIST=$(/usr/sbin/crm_node -l |awk '{print $2}' |grep -v $NODE)
        NODES_N=$(/usr/sbin/crm_node -l |awk '{print $2}' |grep -v $NODE |wc -l)
        NODES_ON=0
        for row in $NODE_LIST; do
                RES_ON=$(echo "$RES_CHECK" |grep "$row")
                if [ -n "$RES_ON" ]; then
                        logger "$row online"
                        ((NODES_ON++))
                fi
        done
        if [ $NODES_ON -lt $NODES_N ]; then
                logger "Erro. Algum node nao esta online"
                exit
        fi

        #desativa servicos do node
```

```
        logger "Standby $NODE"
        /usr/sbin/crm node standby "$NODE"

        #aguarda node ficar livre, vms down e drbd down
        while :; do
                RES_CHECK_DRBD=$(/usr/sbin/crm resource show $STANDBY_CHECK_RES)
                RES_DRBD_ON=$(echo "$RES_CHECK_DRBD" |grep "$NODE")
                VMS_NUM=$(virsh list --name |wc -l)
                if [ -z "$RES_DRBD_ON" ] && [ $VMS_NUM -le 1 ]; then
                        logger "Pronto para reiniciar"
                        break
                else
                        logger "Servicos ainda executando"
                fi
                sleep 30
        done

        #escreve arquivo para recuperar node depois do reboot
        echo $(date +"%Y-%m-%d") > pacemaker_reboot.tmp

        #reinicia node
        logger "Rebooting..."
        /sbin/reboot

fi
```