



UCS

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

BRUNO EMER

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

André Luis Martinotto
Orientador

Caxias do Sul
Junho de 2016

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

por

Bruno Emer

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

Projeto de Diplomação

Orientador: André Luis Martinotto

Banca examinadora:

Maria de Fatima Webber do Prado Lima

CCTI/UCS

Ricardo Vargas Dorneles

CCTI/UCS

Projeto de Diplomação apresentado em
1 de julho de 2016

Daniel Luís Notari
Coordenador

SUMÁRIO

| | |
|--|----|
| LISTA DE SIGLAS | 4 |
| LISTA DE FIGURAS | 6 |
| LISTA DE TABELAS | 7 |
| RESUMO | 8 |
| 1 INTRODUÇÃO | 9 |
| 1.1 Objetivos | 10 |
| 1.2 Estrutura do trabalho | 10 |
| 2 ALTA DISPONIBILIDADE | 11 |
| 2.1 Tolerância a falhas | 11 |
| 2.2 Redundância | 13 |
| 2.3 Cálculo da alta disponibilidade | 14 |
| 2.4 Considerações finais | 15 |
| 3 VIRTUALIZAÇÃO | 16 |
| 3.1 Máquinas virtuais de aplicação | 18 |
| 3.2 Máquinas virtuais de sistema | 19 |
| 3.2.1 Arquiteturas de máquinas virtuais de sistema | 20 |
| 3.2.2 Implementações de máquinas virtuais de sistema | 21 |
| 3.3 Vantagens das máquinas virtuais | 22 |
| 3.3.1 Virtualização de Desktop | 22 |
| 3.3.2 Virtualização de servidores | 23 |
| 3.4 Considerações finais | 24 |
| 4 INFRAESTRUTURA ATUAL DA EMPRESA | 25 |
| 4.1 Instalação física | 26 |
| 4.2 Servidores sem virtualização | 28 |
| 4.3 Servidores com virtualização | 29 |
| 4.3.1 Servidor Brina | 29 |
| 4.3.2 Servidor Fulmine | 30 |
| 4.3.3 Servidor Piova | 31 |
| 4.3.4 Servidor Raggio | 33 |
| 4.3.5 Servidor Tempesta | 33 |
| 4.3.6 Servidor Tuono | 34 |
| 4.3.7 Servidor Venti | 35 |

| | | |
|----------|---|-----------|
| 4.4 | Considerações finais | 36 |
| 5 | PROJETO DE IMPLEMENTAÇÃO | 37 |
| 5.1 | Levantamento dos serviços críticos | 37 |
| 5.1.1 | DNS recursivo primário | 37 |
| 5.1.2 | Autenticação Radius | 38 |
| 5.1.3 | Sistemas da empresa e do provedor | 39 |
| 5.1.4 | Telefonia interna sobre IP | 40 |
| 5.1.5 | Situação atual | 41 |
| 5.2 | Proposta de solução | 41 |
| 5.2.1 | Softwares para a replicação de dados | 41 |
| 5.2.1.1 | DRBD | 42 |
| 5.2.1.2 | GlusterFS | 43 |
| 5.2.1.3 | Rsync | 44 |
| 5.2.1.4 | Software de replicação adotado | 44 |
| 5.2.2 | Softwares para o gerenciamento de cluster | 45 |
| 5.2.2.1 | Ganeti | 45 |
| 5.2.2.2 | Heartbeat | 46 |
| 5.2.2.3 | Pacemaker | 47 |
| 5.2.2.4 | Software de gerenciamento adotado | 48 |
| 5.2.3 | Projeto de implementação | 48 |
| 5.3 | Considerações finais | 49 |
| 6 | CONCLUSÃO | 50 |
| 6.1 | Cronograma | 51 |
| | REFERÊNCIAS | 52 |

LISTA DE SIGLAS

| | |
|-------|--|
| ADSL | <i>Asymmetric Digital Subscriber Line</i> |
| API | <i>Application Programming Interface</i> |
| ASP | <i>Active Server Pages</i> |
| CRM | <i>Cluster Resource Management</i> |
| DNS | <i>Domain Name System</i> |
| DRBD | <i>Distributed Replicated Block Device</i> |
| ECC | <i>Error Correction Code</i> |
| FTP | <i>File Transfer Protocol</i> |
| GFS | <i>Global File System</i> |
| HTTP | <i>Hypertext Transfer Protocol</i> |
| IIS | <i>Internet Information Services</i> |
| IMAP | <i>Internet Message Access Protocol</i> |
| IP | <i>Internet Protocol</i> |
| IPv6 | <i>Internet Protocol version 6</i> |
| ISA | <i>Instruction Set Architecture</i> |
| JVM | <i>Java Virtual Machine</i> |
| KVM | <i>Kernel-based Virtual Machine</i> |
| LTS | <i>Long Term Support</i> |
| LVM | <i>Logical Volume Manager</i> |
| MAC | <i>Media Access Control</i> |
| MTBF | <i>Mean Time Between Failures</i> |
| MTTR | <i>Mean Time To Repair</i> |
| NAT | <i>Network Address Translation</i> |
| OCFS2 | <i>Oracle Cluster File System 2</i> |
| PC | <i>Personal Computer</i> |
| PHP | <i>Personal Home Page</i> |
| POP | <i>Post Office Protocol</i> |
| PPPoE | <i>Point-to-Point Protocol over Ethernet</i> |
| PRTG | <i>Paessler Router Traffic Grapher</i> |
| RAID | <i>Redundant Array of Independent Disks</i> |
| RAM | <i>Random Access Memory</i> |
| SLA | <i>Service Level Agreement</i> |
| SMTP | <i>Simple Mail Transfer Protocol</i> |
| SNMP | <i>Simple Network Management Protocol</i> |
| SPOF | <i>Single Point Of Failure</i> |

| | |
|------|---|
| SSH | <i>Secure Shell</i> |
| SVN | <i>Subversion</i> |
| TCP | <i>Transmission Control Protocol</i> |
| TI | Tecnologia da Informação |
| UDP | <i>User Datagram Protocol</i> |
| VM | <i>Virtual Machine</i> |
| VMM | <i>Virtual Machine Monitor</i> |
| WHM | <i>WebHost Manager</i> |
| XMPP | <i>EXtensible Messaging and Presence Protocol</i> |

LISTA DE FIGURAS

| | |
|---|----|
| Figura 3.1: Interfaces de sistemas de computação. | 17 |
| Figura 3.2: Máquinas virtuais de aplicação e de sistema. | 18 |
| Figura 3.3: Componentes da virtualização. | 19 |
| Figura 3.4: Arquiteturas de máquinas virtuais de sistema. | 20 |
| Figura 3.5: Implementações de máquinas virtuais de sistema. | 21 |
| Figura 4.1: Diagrama de instalação elétrica. | 25 |
| Figura 4.2: Modelo de estrutura física. | 27 |
| Figura 4.3: Imagem do <i>rack</i> e dos servidores. | 27 |
| Figura 4.4: Servidor de virtualização <i>Brina</i> | 29 |
| Figura 4.5: Servidor de virtualização <i>Fulmine</i> | 30 |
| Figura 4.6: Servidor de virtualização <i>Piova</i> | 32 |
| Figura 4.7: Servidor de virtualização <i>Raggio</i> | 33 |
| Figura 4.8: Servidor de virtualização <i>Tempesta</i> | 34 |
| Figura 4.9: Servidor de virtualização <i>Tuono</i> | 35 |
| Figura 4.10: Servidor de virtualização <i>Venti</i> | 36 |
| Figura 5.1: Gráfico de requisições DNS (a) e comparação de requisições UDP entre os principais servidores (b). | 38 |
| Figura 5.2: Gráfico de comparação de elementos (a) e de conexões TCP (b) entre os principais servidores. | 39 |
| Figura 5.3: Gráfico de requisições por segundo do maior sistema. | 40 |
| Figura 5.4: Gráfico da quantidade de canais ativos no servidor de telefonia. | 40 |
| Figura 5.5: Exemplo do modelo <i>master-slave</i> do <i>Distributed Replicated Block Device</i> (DRBD). | 42 |
| Figura 5.6: Modelo do <i>GlusterFS</i> | 43 |
| Figura 5.7: Transferência de arquivos através do <i>Rsync</i> | 44 |
| Figura 5.8: Arquitetura do <i>Ganeti</i> | 46 |
| Figura 5.9: Arquitetura do <i>Heartbeat</i> | 46 |
| Figura 5.10: Exemplo da arquitetura do <i>Pacemaker</i> | 47 |
| Figura 5.11: Estrutura do <i>cluster</i> | 49 |

LISTA DE TABELAS

| | |
|---|----|
| Tabela 2.1: Níveis de alta disponibilidade e exemplos de sistemas. | 14 |
| Tabela 4.1: Configuração dos servidores físicos. | 26 |
| Tabela 5.1: Serviços críticos do ano de 2015. | 41 |
| Tabela 5.2: Comparação ferramentas de replicação de dados. | 45 |
| Tabela 5.3: Comparação ferramentas de gerenciamento de <i>cluster</i> | 48 |
| Tabela 6.1: Cronograma TCC I | 51 |
| Tabela 6.2: Cronograma TCC II | 51 |

RESUMO

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

Além do estudo sobre alta disponibilidade apresentado neste trabalho, será feito um estudo sobre virtualização, pois é um dos principais recursos utilizados para se obter alta disponibilidade.

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

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

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

1 INTRODUÇÃO

O crescente avanço tecnológico e o desenvolvimento da Internet, provocou um aumento no número de aplicações ou serviços que dependem da infraestrutura de Tecnologia da Informação (TI). Além disso, percebe-se um aumento significativo no número de operações *on-line* que são 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 será 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.

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

1.1 Objetivos

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

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

1.2 Estrutura do trabalho

O trabalho foi estruturado em cinco capítulos, que são:

- Capítulo 2: apresenta o conceito de alta disponibilidade e conceitos relacionados;
- Capítulo 3: é apresentado um breve histórico da virtualização, bem como o conceito de máquinas virtuais e as estratégias utilizadas para a implementação das mesmas;
- Capítulo 4: descreve o ambiente atual da empresa com os serviços que são fornecidos;
- Capítulo 5: neste capítulo são descritos os serviços críticos e é apresentado a solução de alta disponibilidade proposta. Essa proposta de solução foi desenvolvida baseada na utilização de virtualização;
- Capítulo 6: apresenta as conclusões parciais do trabalho e o cronograma de andamento.

2 ALTA DISPONIBILIDADE

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

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

2.1 Tolerância a falhas

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

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

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

O objetivo da tolerância a falhas é aumentar a disponibilidade de um sistema, ou seja, aumentar o intervalo de tempo em que os serviços fornecidos estão disponíveis aos usuários. Um sistema é dito tolerante a falhas se ele for capaz de mascarar a presença de falhas ou recuperar-se de uma sem afetar o funcionamento do sistema (PANKAJ, 1994).

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

A tolerância a falhas pode ser dividida em dois tipos. O primeiro tipo, o mascaramento, não se manifesta na forma de erro ao sistema, pois as falhas são tratadas na origem. O mascaramento é utilizado principalmente em sistemas críticos e de tempo real. Um exemplo são os códigos de correção de erros, em inglês *Error Correction Code* (ECC), que são utilizados em memórias para 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 em outros processos.
- Recuperação: após a detecção de um erro ocorre a recuperação, onde o estado de erro é alterado para estado livre de erros. A recuperação pode ser feita de duas formas, que são:
 - *forward error recovery* (recuperação por avanço): ocorre uma condução para um estado que ainda não ocorreu. É a forma de recuperação mais eficiente, porém mais complexa de ser implementada.

- *backward error recovery* (recuperação por retorno): ocorre um retorno para um estado anterior e livre de erros. Para retornar ao estado anterior podem ser utilizados pontos de recuperação (*checkpoints*). Assim, quando ocorrer um erro, um *rollback* é executado, ou seja, o sistema retornará a um estado anterior a falha.
- Tratamento: procura prevenir que futuros erros aconteçam. Nesta fase ocorre a localização da falha para descobrir o componente que originou a falha. A substituição do componente danificado pode ser feita de forma manual ou automática. O reparo manual é feito por um operador que é responsável pelo reparo ou substituição de um componente. Como exemplo pode-se citar a troca de um disco rígido de um servidor. Já o reparo automático é utilizado quando existe um componente em espera para a substituição, como por exemplo, um disco configurado como *hot spare*, ou seja, um componente de *backup* que assumirá o lugar do outro imediatamente após o componente principal falhar. Em *storages* ou servidores, o *hot spare* pode ser configurado através de um *Redundant Array of Independent Disks* (RAID) (ROUSE, 2013).

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 *slaves*, possam recuperar e executar essas operações, com isso mantendo os dados

sincronizados. Neste caso, tanto o servidor *master* quanto os *slaves* executam o serviço *MySQL*, caracterizando uma redundância (SILVA VIANA, 2015). A redundância de *software* também pode ser implementada com o objetivo de tolerar falhas e *bugs* em um *software* crítico. Existem algumas técnicas que podem ser utilizadas para isso, como por exemplo, a programação de *n*-versões, que consiste no desenvolvimento de *n* versões de um mesmo *software*. Desta forma, possibilita-se o aumento da disponibilidade, uma vez que 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 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

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

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 extremamente 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 descrição do serviço, requerimentos, horário de funcionamento, *uptime* do serviço, *downtime* máximo do serviço, entre outros (SMITH, 2010).

2.4 Considerações finais

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

3 VIRTUALIZAÇÃO

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

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

A virtualização foi definida nos anos 60 e 70 como uma camada entre o *hardware* e o sistema operacional que possibilitava a divisão e a proteção dos recursos físicos. Porém, atualmente ela abrange outros conceitos, como por exemplo a *Java Virtual Machine* (JVM), que não virtualiza um *hardware*. De fato, 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 executar em uma plataforma *Y* (LAUREANO; MAZIERO, 2008).

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

- Conjunto de instruções ou *Instruction Set Architecture* (ISA): é a interface básica, que fica entre o *software* e o *hardware*, e é composta por instruções de código de máquina. Essa interface é dividida em dois grupos:
 - Instruções de usuário ou *User ISA*: são instruções disponíveis às 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*. 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 (BUYA; 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 sensíveis¹ no *hardware* (BUYA; VECCHIOLA; SELVI, 2013). Como exemplo de instruções de sistema pode-se citar as instruções que alteram o estado dos registradores do processador;
- Chamadas de sistema ou *syscalls*: são operações oferecidas pelo núcleo do sistema operacional para as aplicações dos usuários. Essas operações permitem um acesso controlado aos dispositivos, a memória e ao processador. As instruções privilegiadas não podem ser executadas no modo usuário, por isso, as aplicações de usuários utilizam chamadas de sistemas em seu lugar, e então o sistema operacional determina se essas operações poderão comprometer 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 e suportam apenas uma aplicação, ou seja, elas provêm um ambiente que permite a execução de uma aplicação convidada. Um exemplo de máquina virtual de aplicação é a JVM. Já uma máquina virtual de sistema suporta um sistema operacional convidado, com suas aplicações executando sobre ele. Uma máquina virtual executando

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

sobre o hipervisor *Kernel-based Virtual Machine* (KVM) (OVA, 2016) é um exemplo de máquina virtual de aplicação (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 máquina virtual Java (JVM) e a *Microsoft Common Language Infrastructure*, que é a base da plataforma *.NET* (CARISSIMI, 2008);

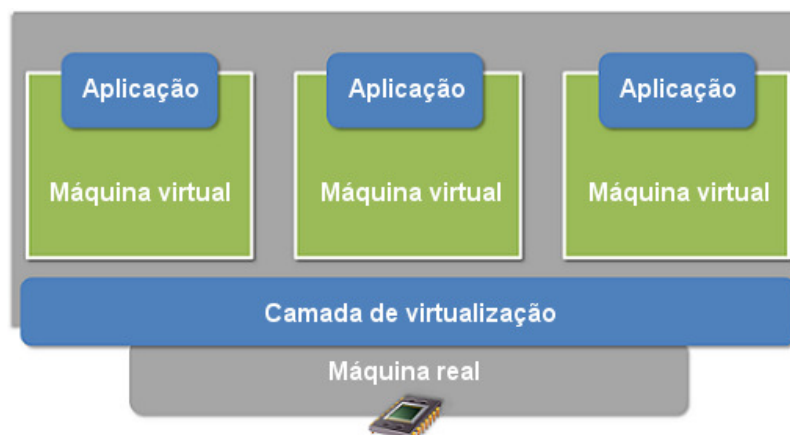
- Emulação do sistema operacional: nesse caso é feito um mapeamento entre as chamadas de sistema que são utilizadas pela aplicação e as chamadas de sistema operacional hospedeiro. A virtualização de aplicação pode ser encontrada em ferramentas que emulam uma aplicação que foi desenvolvida para uma plataforma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

3.2 Máquinas virtuais de sistema

As máquinas virtuais de sistema, também chamadas de hipervisor ou *Virtual Machine Monitor* (VMM), são uma camada de *software* que possibilita que múltiplos sistemas operacionais convidados executem sobre um mesmo computador físico, ou seja, o hipervisor provê uma interface ISA virtual, que pode ou não ser igual a interface real, e virtualiza outros componentes de *hardware*, para que cada máquina virtual convidada possa ter seus próprios recursos. Para tanto, a virtualização de sistema utiliza abstrações em sua arquitetura. Por exemplo, ela transforma um disco rígido físico em dois 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.



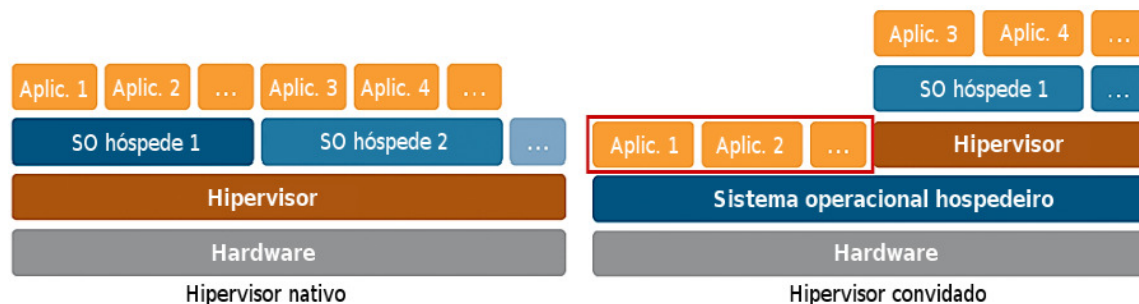
Fonte: ANDRADE (2011)

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 usando diferentes estratégias. Atualmente as estratégias mais utilizadas são a virtualização total e a paravirtualização (Figura 3.5):



Fonte: MAZIERO (2013)

Figura 3.5: Implementações de máquinas virtuais de sistema.

- Virtualização total: nesta estratégia todas as interfaces de acesso ao *hardware* são virtualizadas. Desta forma, possibilita-se que os sistemas operacionais convidados executem como se estivessem diretamente sobre o *hardware*. Na virtualização total o conjunto de instruções do processador é acessível somente ao hipervisor, sendo que essa estratégia utiliza tradução dinâmica¹ para executar as instruções do sistema convidado. A grande vantagem dessa estratégia é a possibilidade de um sistema convidado ser executado sem a necessidade de ser modificado. Porém, essa estratégia possui um desempenho inferior devido ao fato do hipervisor intermediar todas as chamadas de sistemas e operações do sistema convidado. Um exemplo de ferramenta que utiliza a virtualização total é o *QEmu* (QEmu, 2016);
- Paravirtualização: nesta estratégia a interface entre o hipervisor e o sistema operacional convidado é modificado para se obter uma maior eficiência. Destaca-se que essa estratégia de implementação utiliza uma arquitetura de hipervisor nativo. As modificações na interface de sistema (*system ISA*) exigem que o sistema operacional convidado seja adaptado para o hipervisor, para possibilitar a execução sobre a plataforma virtual. Para essa adaptação, o hipervisor disponibiliza uma *Application Programming Interface* (API), para que os sistemas convidados possam acessar o ISA de sistema. Contudo, a interface de usuário é mantida, assim, as aplicações do sistema convidado não precisam ser modificadas (MAZIERO, 2013).

¹A 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 feito pelo hipervisor. Na virtualização total o hipervisor reserva um espaço para cada sistema convidado, que por sua vez, acessa a memória como se fosse uma memória física, ou seja, inicia o seu endereçamento na posição zero. Sendo assim, cada vez que o sistema convidado acessar a memória, o hipervisor precisará converter os endereços do sistema convidado para os endereços reais de memória. Já na paravirtualização, o hipervisor informa ao sistema convidado a área de memória que ele poderá utilizar, assim, não sendo necessário nenhuma conversão de endereços (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 possibilitam 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* (VMware, 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, estes necessitam de um *hardware* de menor valor, uma vez que esses executarão apenas um terminal remoto. Por fim, essa técnica possibilita uma maior segurança dos dados, pois os dados serão armazenados em um local seguro, como por exemplo, um *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 está 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 a necessidade de executar diferentes sistemas operacionais para determinadas disciplinas. Isso é necessário quando pretende-se configurar e executar aplicações para fim de experimentos ou aprendizagem, com isso, essas ações não afetarão o sistema hospedeiro. A grande vantagem da utilização de máquinas virtuais nesse tipo de ambiente é a facilidade na manutenção, pois as máquinas virtuais podem ser restauradas de forma simples.

3.3.2 Virtualização de servidores

Em muitos casos as empresas utilizam serviços distribuídos entre diferentes servidores físicos, como por exemplo, servidores de *e-mail*, hospedagens 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 memória RAM maior. Portanto, uma das grandes vantagens da virtualização é um melhor aproveitamento dos recursos. De fato, alocando vários serviços em um único servidor físico tem-se um melhor aproveitamento do *hardware* (MOREIRA, 2006) e, conseqüentemente, tem-se uma redução nos custos de administração e manutenção dos servidores físicos.

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

Outra motivação para a utilização de virtualização em servidores consiste na redução de custos com energia elétrica e equipe que faz a 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, existe uma técnica chamada *live migration*, ou migração em tempo real. Essa técnica possibilita que uma máquina virtual, que está executando em um servidor físico, seja transferida, através da rede, para outro servidor sem ser reiniciada. Nesse processo a máquina virtual fica no estado suspenso (por um período curto de tempo) até que o servidor de destino receba os dados necessários para continuar a execução da máquina virtual (SILVA, 2009). Essa técnica possibilita a utilização de redundância de *software*.

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 serão utilizadas no desenvolvimento deste trabalho. De fato, essas serão utilizadas para a implementação da redundância de *software*. No próximo capítulo será feito o levantamento e análise dos serviços fornecidos pela empresa que está sendo estudada neste trabalho. Posteriormente, serão apresentados os serviços que são considerados críticos e a proposta de solução de alta disponibilidade.

4 INFRAESTRUTURA ATUAL DA EMPRESA

Este capítulo apresentará a infraestrutura de TI da empresa que será objetivo de estudo neste trabalho. Esta é uma empresa que fornece serviços de hospedagens e também está associada a um provedor de Internet¹. A empresa possui grande parte de seus clientes localizados na serra do Rio Grande do Sul, sendo que, atualmente, 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.

Atualmente, a empresa possui redundância de refrigeração e de energia, como pode ser observado na Figura 4.1. A redundância de refrigeração é composta por dois ares-condicionados (*Ar 1 (Inverter)* e *Ar 2* na Figura 4.1).

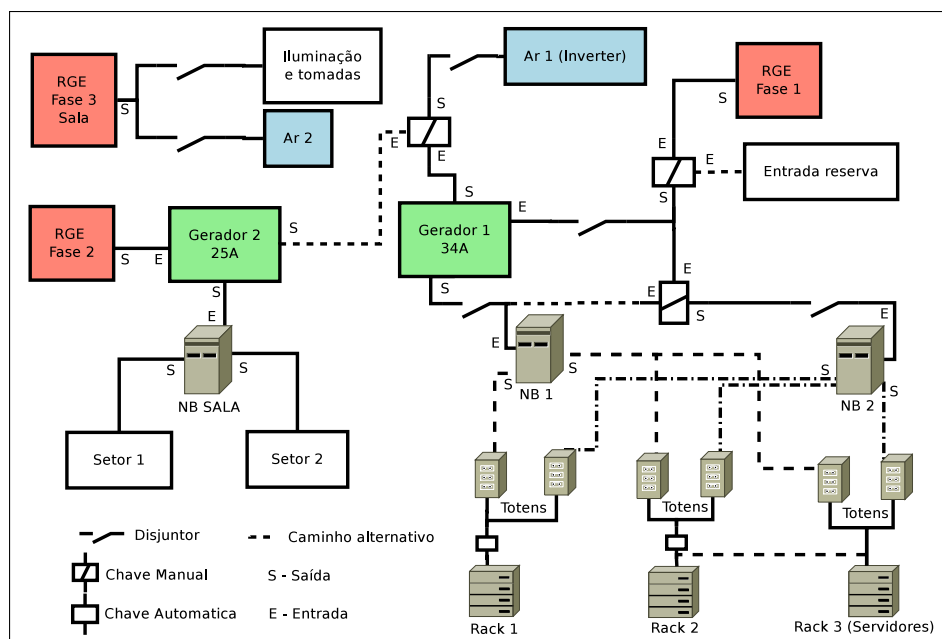


Figura 4.1: Diagrama de instalação elétrica.

¹É 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. Conectado 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 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. Na Seção 4.3 será descrito a estrutura dos servidores que utilizam de virtualização e todos os serviços fornecidos por esses servidores.

4.1 Instalação física

A estrutura atual da empresa é composta por quatorze servidores físicos. A configuração 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 *Random Access Memory* (RAM), número de discos rígidos e a capacidade unitária de cada disco.

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

Tabela 4.1: Configuração dos servidores físicos.

Todos os servidores estão ligados ao *switch*, que provê 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

Atualmente 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 instalado a ferramenta *Bacula Storage 5.2.6* (Bacula, 2016). De fato, esse só armazena 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. Com o *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, 2016); compartilhamento de arquivos *Samba 4.3.9* (Samba Team, 2016); controle de versões de *software Subversion (SVN) 1.8.8* (Apache Software Foun-

¹A rede *core* é a rede de transporte do provedor, onde trafega os principais *link* de acesso à Internet.

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

³Os *fans* são exaustores, ou seja, ventiladores que empurram o ar quente para fora de um recipiente ou ambiente (ATS, 2012).

dation, 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 armazenamento das imagens das câmeras do provedor.

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. Atualmente 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:



Figura 4.4: Servidor de virtualização *Brina*.

- *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 utilizada 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, ou seja, esse servidor armazena o site do provedor. 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 da *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 fim 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 (TANENBAUM; WETHERALL, 2011) e NAT (*Network Address Translation*) (KUROSE; ROSS, 2006) 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 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 (TANENBAUM; WETHERALL, 2011) 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



Figura 4.6: Servidor de virtualização *Piowa*.

provê acesso a sites *Web* desenvolvidos com a linguagem *Active Server Pages* (ASP) (Microsoft, 2016a), através do *software Internet Information Services (IIS) 7.5* (Microsoft, 2016b). Destaca-se que esse servidor hospeda um número baixo de sites. De fato, esse 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 atualmente 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. 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:

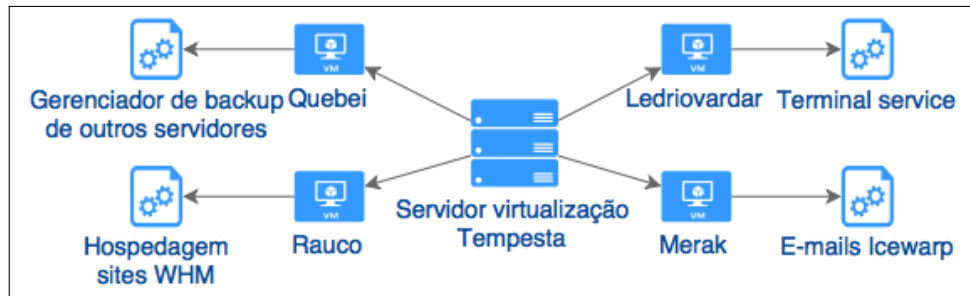


Figura 4.8: Servidor de virtualização *Tempesta*.

- *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* (Icwarp, 2016). Essa aplicação fornece os serviços de envios de: *e-mails* através do protocolo SMTP (*Simple Mail Transfer Protocol*) (KUROSE; ROSS, 2006); recebimentos de *e-mails* através dos protocolos POP (*Post Office Protocol*) (KUROSE; ROSS, 2006) e IMAP (*Internet Message Access Protocol*) (KUROSE; ROSS, 2006); 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 esse *backup* é armazenado 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) instalado, 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). Ou seja, esse servidor virtual fornece o mesmo serviço do servidor *Roncon*, que foi descrito na Seção 4.3.2. Esse servidor fornece acesso a sites *Web* desenvolvidos com a linguagem PHP. 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*

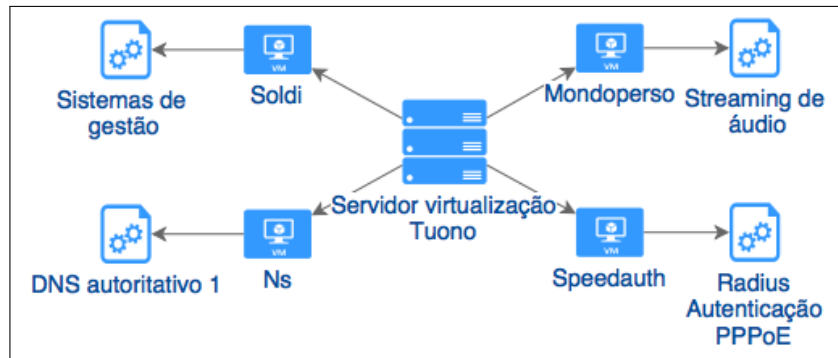


Figura 4.9: Servidor de virtualização *Tuono*.

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

- *Ns*: esse servidor possui 1 *core* de 2.53 GHz, 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* são utilizados: *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*) (KUROSE; ROSS, 2006);
- *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), esse servidor faz a hospedagem de vídeos utilizando o protocolo FTP e 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



Figura 4.10: Servidor de virtualização *Venti*.

8 GB de disco. Esse servidor possui o sistema operacional *Ubuntu 14.04 LTS* (Canonical, 2016) e fornece *streaming* de áudio para uma *Web* rádio. Esse serviço é feito através do *software* livre *Icecast 2.3.3* (Xiph.Org Foundation, 2016);

- *Pomodoro*: sua configuração é 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. No próximo capítulo será feita a identificação dos serviços mais críticos da empresa. Além disso, será apresentada uma proposta de um ambiente de alta disponibilidade para os mesmos. Nesta proposta, serão apresentados os *softwares* que serão utilizados para implementar a alta disponibilidade.

5 PROJETO DE IMPLEMENTAÇÃO

Neste capítulo serão apresentados os serviços considerados críticos para a empresa. Posteriormente, será apresentada uma proposta de uma solução de alta disponibilidade. Essa solução será detalhada na Seção 5.2 e será baseada na utilização de virtualização e de *softwares* de código aberto.

5.1 Levantamento dos serviços críticos

No capítulo anterior foram detalhados todos os serviços que estão disponíveis na empresa. Nesta seção será feito uma análise desses serviços, destacando os serviços considerados críticos para a empresa. Para a seleção dos serviços críticos foram adotados os seguintes critérios:

- A quantidade de clientes ou funcionários que utilizam o serviço: esse é o item mais relevante, pois impacta diretamente no faturamento da empresa. De fato, se um cliente ficar sem acesso à Internet, o cliente terá um desconto proporcional ao tempo que ficou sem acesso;
- O número de requisições: esse número é importante, uma vez que, indicam 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*) (TANENBAUM; WETHERALL, 2011), o número de requisições UDP (*User Datagram Protocol*) (TANENBAUM; WETHERALL, 2011), 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 de monitoramento;

Nas próximas seções serão descritos os serviços críticos, com base nos critérios adotados.

5.1.1 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 ilustrado 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 possui um maior número de requisições UDP¹. 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 os serviços.

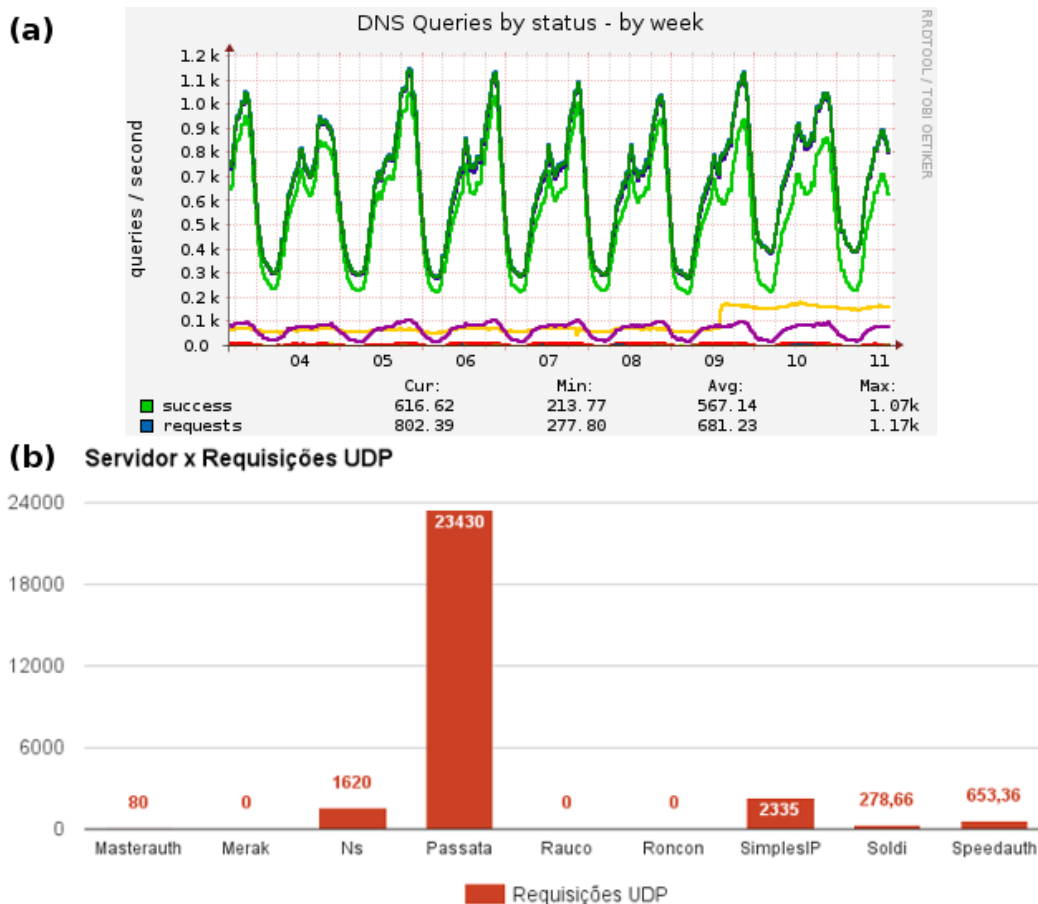


Figura 5.1: Gráfico de requisições DNS (a) e comparação de requisições UDP entre os principais servidores (b).

5.1.2 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, conseqüentemente, não conseguirão utilizar o serviço de Internet. Os servidores *Masterauth* e *Speedauth* fornecem o serviço de *Radius*, sendo que 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 da conexão de cada cliente, o

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

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, representa a quantidade de clientes que utilizam esses servidores para autenticação. Como pode ser observado na Figura 5.2 (a), os servidores *Masterauth* e *Speedauth* estão entre os que apresentam o maior número de elementos. Além disso, existem um grande número de conexões TCP nesses servidores, como pode ser observado na Figura 5.2 (b).



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

5.1.3 Sistemas da empresa e do provedor

O sistema do provedor é responsável pela maior parte das operações gerenciais do provedor. De fato, 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. Esse sistema é executado no servidor *Soldi* que recebe aproximadamente 3 requisições HTTP (*Hypertext Transfer Protocol*) (TANENBAUM; WETHERALL, 2011) por segundo, como pode ser observado na Figura 5.3. Além disso, a empresa mantém 28 sistemas de outros clientes nesse servidor. O nível de utilização deste servidor

é ilustrado na Figura 5.2 (b) onde pode ser observado que esse serviço possui um número considerável de conexões TCP¹.

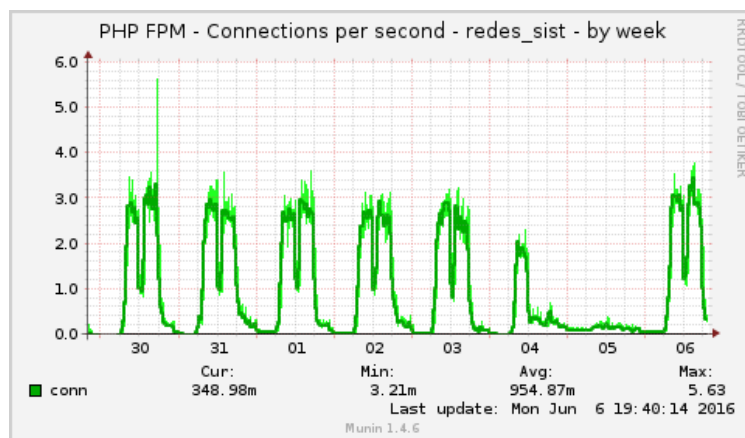


Figura 5.3: Gráfico de requisições por segundo do maior sistema.

5.1.4 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, ou seja, 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. 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, na Figura 5.1 (b), que esse serviço possui um elevado número de requisições UDP², se comparado aos demais servidores.

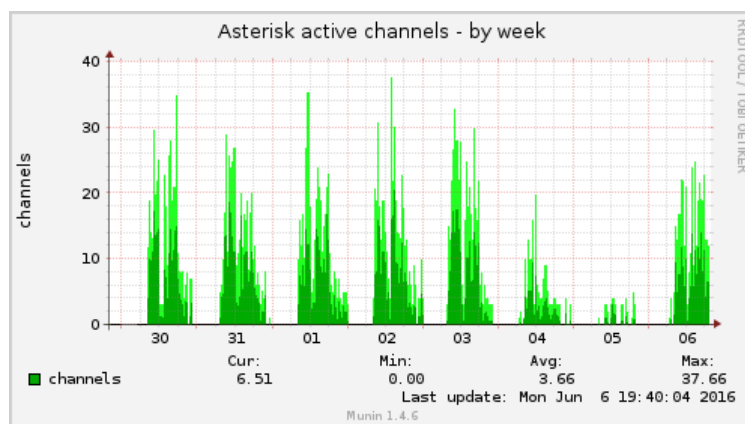


Figura 5.4: Gráfico da quantidade de canais ativos no servidor de telefonia.

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

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

5.1.5 Situação atual

A partir da análise inicial, verificou-se que os servidores a serem incluídos no ambiente de alta disponibilidade 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 da empresa e do provedor;
- *SimplesIP*: servidor de telefonia interna sobre IP para atendimento aos 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 sua medição é dos últimos 6 meses de 2016, por isso possui um *Uptime* elevado. A partir da implementação da solução de alta disponibilidade deste trabalho pretende-se atingir um *Uptime* de 99,99%.

| 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.

5.2 Proposta de solução

Para implementar esta solução será necessário dois servidores físicos, sendo que a configuração de cada servidor deverá ser 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 das máquinas virtuais que atualmente abrangem os serviços críticos. Com essa solução, caso ocorra alguma falha em um servidor, as máquinas virtuais serão transferidas para o outro servidor. Destaca-se que tais recursos de *hardware* já encontram-se disponíveis, sendo necessário somente efetuar uma reorganização das máquinas virtuais.

Já os *softwares* necessários para a implementação da alta disponibilidade podem ser divididas em dois grupos: *softwares* de replicação de dados; e *softwares* para o monitoramento e a transferências das máquinas virtuais entre os servidores.

5.2.1 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 (TANENBAUM; WOODHULL, 2009). Essa solução é eficaz para garantir que o sistema não fique indisponível em

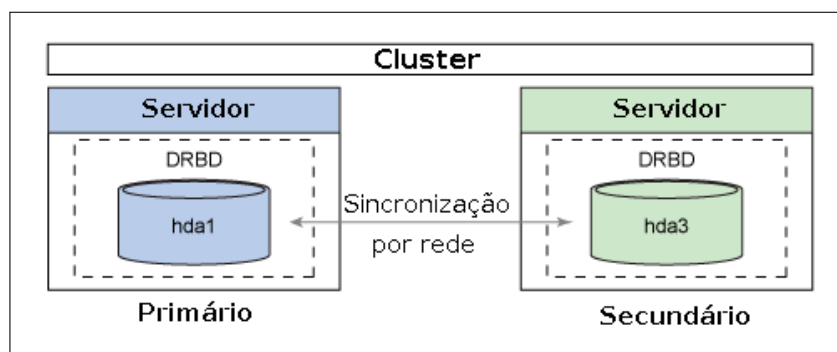
caso de falha de discos¹, porém não garante a disponibilidade quando um *software* ou algum outro componente de *hardware* falhar (ZAMINHANI, 2008).

A solução de replicação a ser adotada consiste em um espelhamento de dados através da rede. Essa solução permite a cópia dos dados de um servidor para um servidor remoto em tempo real. Normalmente essa solução é estruturada na forma de um *cluster*². Nas próximas seções tem-se uma descrição dos *softwares* de replicação de dados que foram estudados.

5.2.1.1 DRBD

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 em um servidor remoto. O DRBD é implementado através de um módulo do *kernel Linux*.

Para ilustrar sua arquitetura, na Figura 5.5 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 escritas realizadas no nó primário serão replicadas para o disco do nó secundário (ZAMINHANI, 2008).



Fonte: JONES (2010)

Figura 5.5: 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 esse nó é o único que terá permissão para acessar o dispositivo, ou seja, somente ele poderá fazer operações de leitura e escrita. O nó secundário terá apenas uma réplica dos dados;
- *Dual-primary*: ou *dual-master*, neste modo existem dois nós primários, nos quais podem ser feitas operações de leitura e escrita simultaneamente. Porém, este necessita de um sistema de arquivos compartilhados, sendo que esse pode utilizar os seguintes sistemas de arquivos: o *Global File System* (GFS) (Red Hat, 2016b); e o *Oracle Cluster File System 2* (OCFS2) (Oracle, 2016c).

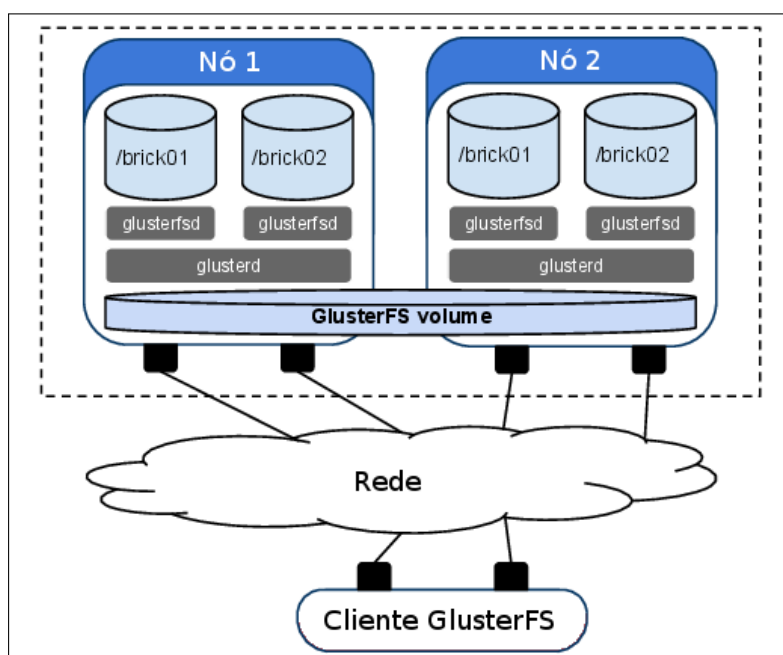
¹Lembrando que essa solução é utilizada no ambiente atual para aumentar a disponibilidade dos servidores.

²Pode-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)

5.2.1.2 GlusterFS

O *GlusterFS* (Red Hat, 2016c) é um sistema de arquivos distribuídos mantido pela *Gluster community*. Esse *software* utiliza estrutura de *cluster* e seu principal objetivo é a escalabilidade, ou seja, existe uma facilidade para aumentar a capacidade do *cluster* a partir da inclusão de novos nós.

Na Figura 5.6 tem-se dois nós, onde cada nó possui dois discos rígidos, esses discos são denominados *bricks*. A partir dos *bricks* o *GlusterFS* constrói um volume lógico que é disponibilizado para clientes *GlusterFS*. Esse volume é acessado pelos clientes através da rede.



Fonte: DAVIES; ORSARIA (2013)

Figura 5.6: Modelo do *GlusterFS*.

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

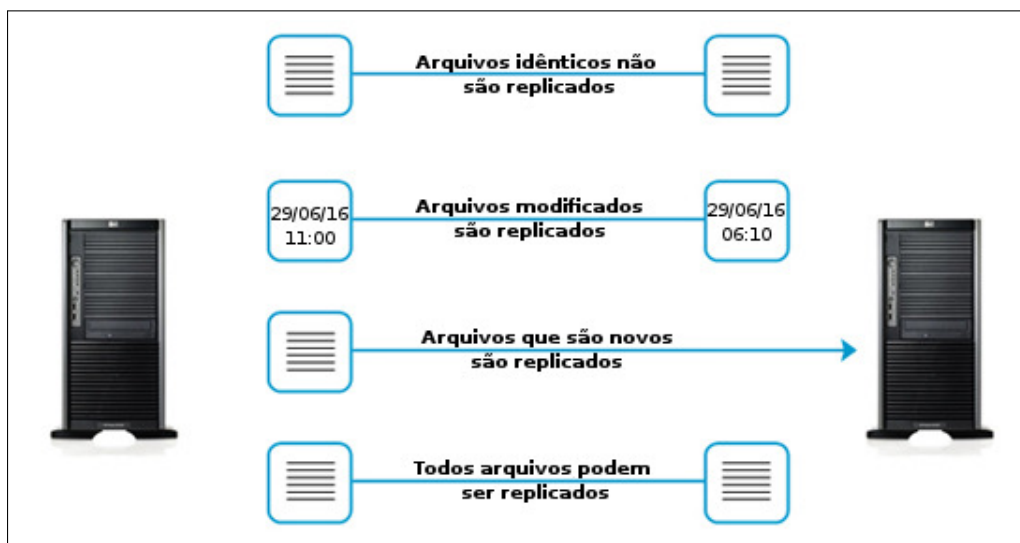
- Volume distribuído: neste tipo os arquivos são distribuídos entre os diferentes *bricks* dos nós. O objetivo deste tipo de configuração é ampliar a capacidade de armazenamento. Desta forma, não tem-se redundância de dados, sendo assim, caso um nó falhe haverá uma perda de todos dados;
- Volume replicado: neste os arquivos são replicados entre os *bricks*, deste modo, tem-se uma redundância, sendo assim, caso houver uma falha em um *brick* não haverá perda de dados;
- Volume distribuído e replicado: este é a combinação dos dois tipos de volumes anteriores, sendo assim, é feito a distribuição e a replicação dos arquivos entre os nós;
- Volume listrado: neste tipo ocorre a distribuição de um mesmo arquivo entre os *bricks*, ou seja, um arquivo é dividido e salvo uma parte em cada *brick*. Esse tipo é normalmente utilizado para armazenamento de arquivos muito grandes;
- Volume distribuído e listrado: este tipo de volume é a combinação do distribuído e do listrado, desta forma, é feito a divisão do arquivo e salvo em

bricks distintos e também é feito a replicação.

5.2.1.3 Rsync

O *Rsync* (DAVISON, 2016) é um *software* desenvolvido e mantido por *Wayne Davison*. Esse *software* prove uma rápida transferência arquivos, ou seja, ele faz o sincronismo de arquivos em servidores transferindo dados de um servidor de origem para um de destino.

A Figura 5.7 demonstra o funcionamento do *Rsync*, ilustrando o sincronismo de dados entre dois servidores. Percebe-se que pode ser feito a replicação dos arquivos que não foram modificados ou que não existem no destino, ou pode ser feito a replicação sobrescrevendo todos arquivos.



Fonte: LOPEZ (2011)

Figura 5.7: Transferência de arquivos através do *Rsync*.

O *Rsync* pode ser configurado como servidor, o que permite que vários clientes possam sincronizar os dados com esse servidor. Além disso, o sincronismo pode ser de cliente para cliente, utilizando como transporte o protocolo SSH ou através de *sockets*.

5.2.1.4 Software de replicação adotado

Na Tabela 5.2 tem-se uma comparação entre as ferramentas citadas anteriormente. A ferramenta adotada para replicação de dados será o DRBD, pois permite a configuração *master-slave* ou *dual-primary*, além de suportar a replicação de 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).

De fato, o *GlusterFS* poderia ser utilizado, porém ele faz a replicação a nível de arquivo, sendo assim, não é adequada 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 executar em conjunto com a virtualização.

| Características | DRBD | GlusterFS | Rsync |
|------------------------------|---------------------------------------|---|-----------|
| Master-slave | Sim | Sim | Não |
| Replicação em tempo real | Sim | Sim | Não |
| Nível de replicação | Bloco | Arquivo | Arquivo |
| Número máximo de nós | 16 | 64 | Ilimitado |
| Plataformas | Suse, Debian, CentOS, Red Hat, Ubuntu | Debian, CentOS, Red Hat, Fedora, Ubuntu | Linux |
| Integração com virtualização | Sim | Sim | Não |

Tabela 5.2: Comparação ferramentas de replicação de dados.

5.2.2 Softwares para o gerenciamento de cluster

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

Após a detecção de uma falha, os *softwares* de gerenciamento de *cluster* pode executar operações de *failover* e *failback*. O *failover* é um processo no qual um servidor recebe os serviços que estavam executando em um 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.

5.2.2.1 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 5.8 tem-se a arquitetura deste *software*. O *cluster* é composto por nós, sendo que é necessário 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, ele permite a criação de vários nós *slaves*. Destacando que todos os nós são responsáveis por prover o ambiente de virtualização e armazenar os dados das VMs. Cada nó possui uma ou mais instâncias. As instâncias são as VMs que executam no *cluster*. Em cada instância configura-se dois nós: o nó primário, que a instância executará inicialmente; e o nó secundário, para a instância executar caso o primário falhe. As instâncias também podem ser migradas de um nó para outro de forma manual.

As principais funcionalidades do *Ganeti* são (Google, 2016):

- Criação de instâncias;
- Gerenciamento do armazenamento de dados das instâncias;
- Iniciar e finalizar instâncias, além de efetuar o *failover* entre os nós.



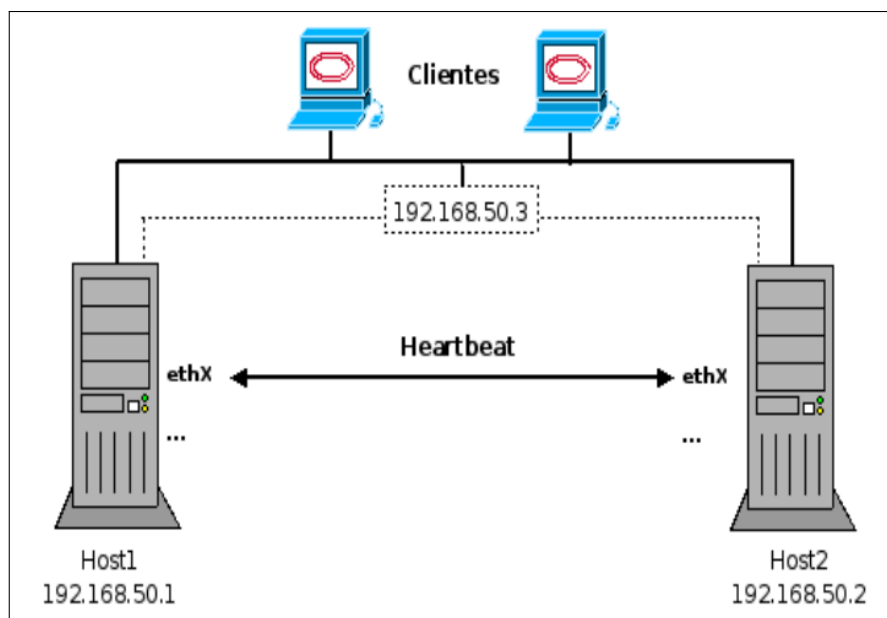
Fonte: CARVALHO; SOBRAL (2011)

Figura 5.8: Arquitetura do *Ganeti*.

5.2.2.2 Heartbeat

O *Heartbeat* é um subprojeto do *Linux-HA* (Linux-HA, 2016a), que desenvolve aplicações para soluções de alta disponibilidade. Esse subprojeto é uma aplicação que envia pacotes *keepalive* UDP, através da rede, para outras aplicações *Heartbeat*, com o objetivo de informar se ele está ativo. Destaca-se que esse *software* pode ser utilizado para alta disponibilidade em ambientes de virtualização (REIS, 2009).

Na Figura 5.9 tem-se o *Heartbeat* executando em dois servidores sobre as interfaces de rede dedicadas (*ethX*), com o objetivo de cada nó monitorar o outro. Caso o nó secundário deixe de receber os sinais do nó primário, este irá tornar-se o nó primário, iniciando o processo de *failover*, com isso, ele receberá o IP virtual (192.168.50.3) e iniciará os serviços previamente configurados.



Fonte: ZAMINHANI (2008)

Figura 5.9: Arquitetura do *Heartbeat*.

As principais funcionalidades do *Heartbeat* são (ClusterLabs, 2016a):

- Envio de mensagens entre os nós para detecção de falhas;
- Efetua os processos de *failover* e *failback*;
- Inicia e finaliza serviços nos nós;

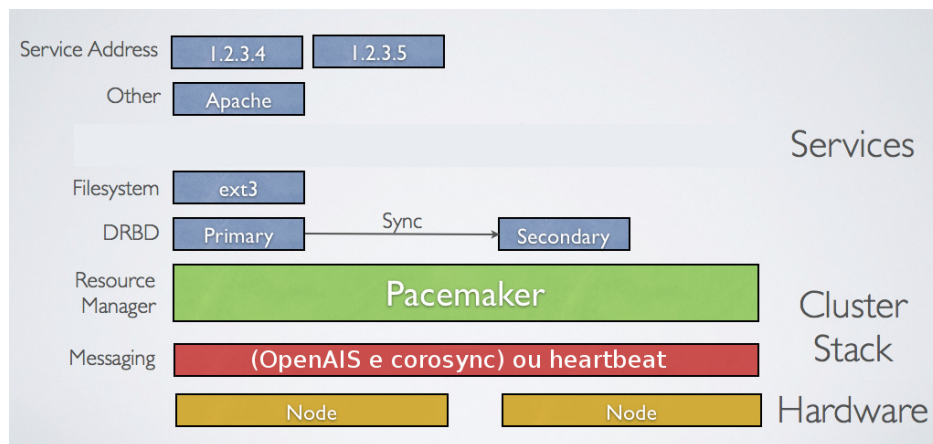
5.2.2.3 Pacemaker

O *Pacemaker* (ClusterLabs, 2016b) é um projeto de código aberto mantido pela *ClusterLabs*, e teve origem com a necessidade de aperfeiçoar o gerenciador de recursos da ferramenta *Heartbeat* (Linux-HA, 2016b). O *Pacemaker* pode ser definido como uma ferramenta de recuperação de falhas a nível de serviço (PERKOV; PAVKOVIĆ; PETROVIĆ, 2011).

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

- *Corosync* (Corosync, 2016): é responsável pelo processo de registro dos nós, de *failover* e de bloqueios distribuídos (utilizados para implementar *Logical Volume Manager* (LVM) (MAUELSHAGEN, 2016), GFS, e OCFS2), entre outros. Essa ferramenta derivou do projeto *OpenAIS*;
- *Heartbeat*: como descrito na Seção 5.2.2.2, essa ferramenta faz envio de mensagens entre os nós do *cluster*, além de iniciar e finalizar serviços.

Na Figura 5.10 tem-se a arquitetura do *Pacemaker*. Como pode ser observado na camada inferior tem-se os nós do *cluster*. Na camada superior tem-se os *softwares* de envio de mensagens e acima deste tem-se o *Pacemaker*. Por fim, tem-se os serviços executando, que são: o *DRBD*; o sistema de arquivos; o IP; e um servidor *Apache*.



Fonte: ClusterLabs (2016b)

Figura 5.10: Exemplo da arquitetura do *Pacemaker*.

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

- Inicia e finaliza serviços dos nós do *cluster*. Esses serviços podem ser desde um servidor *web* até uma interface de rede;
- Replicação de configuração do *cluster* para todos os nós de forma transparente. Desta forma, a configuração do *cluster* pode ser alterada em qualquer nó;
- Elege um nó como primário para centralizar todas as decisões, sendo que, se esse nó falhar outro será eleito primário.

5.2.2.4 Software de gerenciamento adotado

Na Tabela 5.3 tem-se uma comparação dos *softwares* de gerenciamento de *cluster*. O *software* adotado para gerenciar o ambiente será o *Ganeti*, pois este possui todos os requisitos para a criação de um *cluster* de alta disponibilidade, além de possuir a vantagem de ser desenvolvido especificamente para virtualização, o que simplifica a sua configuração. O *Ganeti* possui detecção de falhas tanto a nível de nó quanto a nível de máquina virtual. Destaca-se que foram realizados testes de migração de máquinas virtuais em tempo real utilizando o *Ganeti*, e obteve-se bons resultados.

| Características | Ganeti | Heartbeat | Pacemaker |
|--|----------------------|---|---|
| Sincronismo de configuração entre os nós | Não | Não | Sim |
| Failover e failback | Sim | Sim | Sim |
| Nível de detecção e recuperação | Nó e máquina virtual | Nó | Nó e recurso |
| Plataformas | Linux | Red Hat, CentOS, Fedora, Suse, Debian, Ubuntu | Red Hat, CentOS, Fedora, Suse, Debian, Ubuntu |
| Integração com virtualização | Sim | Não | Sim |

Tabela 5.3: Comparação ferramentas de gerenciamento de *cluster*.

É possível também implementar uma solução de alta disponibilidade com a ferramenta *Heartbeat*. Porém, desta forma seria necessário criar um conjunto de *scripts* para fazer a migração das máquinas virtuais, o que dificultaria a implementação.

Já o *Pacemaker* é o único que possui a característica de sincronismo de configuração entre todos os nós, no entanto, isto não é uma característica essencial para o desenvolvimento deste trabalho.

5.2.3 Projeto de implementação

Como mencionado anteriormente, o ambiente será configurado na forma de um *cluster* o qual será composto por dois servidores, com configuração de 12 *cores* de processamento, 14 GB de memória RAM e 180 GB de disco rígido. Essa configuração inclui 2 GB de memória RAM e 24 GB de disco para cada sistema operacional hospede. Além disso, serão utilizados o mesmo sistema operacional e o mesmo hipervisor que são adotados na empresa, o sistema *Ubuntu 14.04 LTS* e o KVM (OVA, 2016), respectivamente.

A estrutura dos servidores juntamente com as máquinas virtuais e seus respectivos serviços são apresentados na Figura 5.11. Além disso, será utilizado dois *softwares*, o primeiro será o DRBD, que fará a replicação dos dados das máquinas virtuais. Sendo que para cada VM existe um disco DRBD primário, que será replicado em um disco secundário localizado no outro nó do *cluster*. O segundo *software* será o *Ganeti*, que fará o gerenciamento do *cluster*. Esse será responsável pela gerência, monitoramento e migração das VMs. Para essas operações deve-se escolher um nó, que será o nó *master*. Neste nó, serão realizadas as configurações iniciais do *cluster*. Após isso, será adicionado o outro nó para possibilitar a tolerância a falhas.

Deste modo, tem-se o ambiente pronto para a criação das VMs, sendo assim será criado uma instância para cada servidor virtual, e nela será configurado o nó primário e o nó secundário. Como pode ser observado no Figura 5.11, será criado três instâncias no nó *master* e duas instâncias no outro nó, desta forma, além da alta disponibilidade tem-se o balanceamento de carga. Desta forma, quando houver

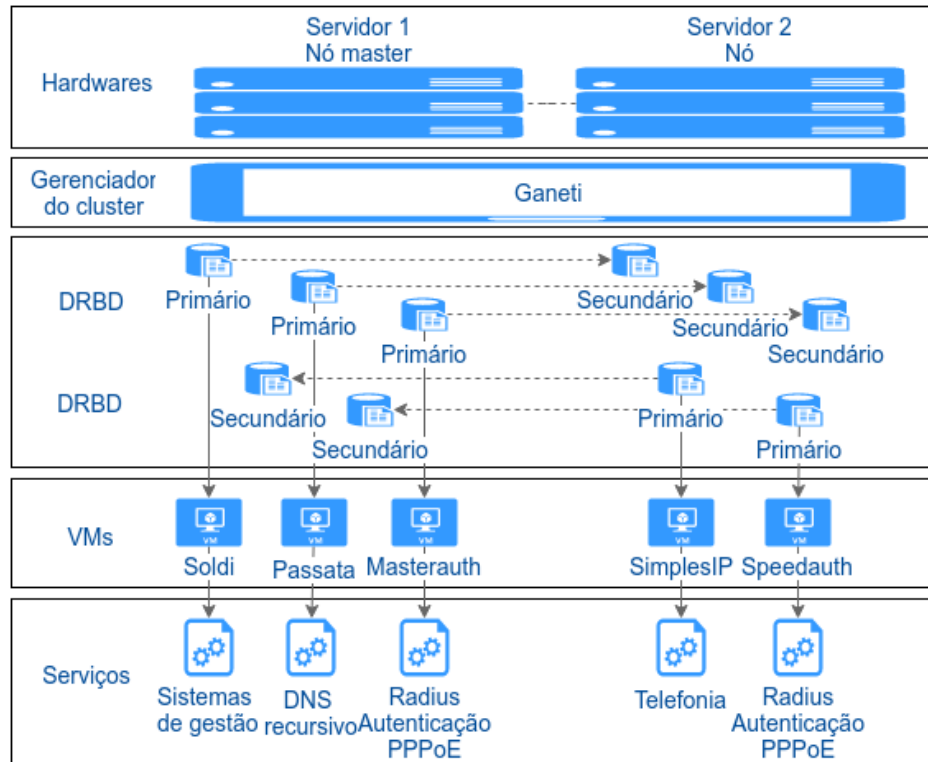


Figura 5.11: Estrutura do *cluster*.

uma falha no nó primário as instâncias serão migradas para o nó secundário, sem que as VMs sejam reiniciadas.

Para essa migração, o *Ganeti* utilizará a opção de migração em tempo real que é fornecida pelo hipervisor KVM (OVA, 2016).

5.3 Considerações finais

Neste capítulo foram apresentados os serviços críticos e foi apresentado uma proposta de solução de alta disponibilidade. No próximo capítulo será feito as conclusões parciais e o cronograma de andamento deste trabalho.

6 CONCLUSÃO

Neste trabalho foi feito um estudo sobre uma empresa prestadora de serviços para Internet, analisando sua estrutura física e os seus servidores. Durante este estudo foram definidos os serviços críticos, para tanto considerou-se o impacto dos mesmos para a empresa.

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 e *softwares* que são responsáveis pelo gerenciamento do *cluster* e pela replicação de dados. O *software* de gerenciamento adotado, que foi o *Ganeti*, fará o gerenciamento, monitoramento e a transferência dos serviços para garantir a alta disponibilidade. O *software* de replicação de dados adotado foi o DRBD, que replicará os dados entre os dois servidores.

O ambiente de alta disponibilidade é composto por máquinas virtuais que contém os serviços críticos, sendo assim, utilizou-se a opção de migração em tempo real, fornecida pelo hipervisor KVM, juntamente com o *Ganeti*. Desta forma, possibilitou-se a alta disponibilidade dos serviços caso um dos servidores falhe. Além disso, nos testes realizados com essas ferramentas pode-se afirmar que obteve-se bons resultados.

Com esse estudo, destaca-se que existe uma variedade de ferramentas que possibilitam criar um ambiente de alta disponibilidade, de forma a atingir o objetivo deste trabalho.

6.1 Cronograma

Na Tabela 6.1 tem-se o cronograma correspondente a primeira parte deste trabalho. Como pode ser observado todas etapas foram concluídas, sendo que houve uma modificação no item 3, definição das ferramentas. Já na Tabela 6.2 tem-se o cronograma da segunda etapa deste trabalho. Nesta etapa será implementada a solução de alta disponibilidade e será realizados os testes.

| | Descrição | Jan | Fev | Mar | Abr | Mai | Jun | Jul |
|---|---------------------------------|------------|------------|------------|------------|------------|------------|------------|
| 1 | Redação TCC I | | | X | X | X | X | X |
| 2 | Estudo bibliográfico | X | X | X | X | X | | |
| 3 | Definição das ferramentas | | | | | X | X | |
| 4 | Análise na empresa | | | | | X | X | |
| 5 | Definição dos serviços críticos | | | | | X | X | |
| 6 | Apresentação TCC I | | | | | | | X |

Tabela 6.1: Cronograma TCC I

| | Descrição | Jul | Ago | Set | Out | Nov | Dez |
|---|--|------------|------------|------------|------------|------------|------------|
| 1 | Redação do TCC II | X | X | X | X | X | X |
| 2 | Elaboração da solução | X | X | X | | | |
| 3 | Reorganização do ambiente de virtualização | X | X | | | | |
| 4 | Realização de testes | | X | X | X | | |
| 5 | Implementação da solução | | | X | X | | |
| 6 | Análise dos resultados e medições | | | | X | X | |
| 7 | Apresentação do TCC II | | | | | | X |

Tabela 6.2: Cronograma TCC II

REFERÊNCIAS

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

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

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

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

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.

BUYA, 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/1-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.

MAUELSHAGEN, H. **LVM2**. <Disponível em: <https://www.sourceware.org/lvm2/>>. Acesso em 26 de junho de 2016.

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

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

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

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

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

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

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-rhafs-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: projetos 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.