Faculdade de Ciências e Tecnologia da
Universidade de Coimbra

Departamento de Engenharia Informática

Solução de Alta Disponibilidade para Sistemas Real-Time

Autor:
Diogo G. C. Espírito Santo

Orientador:
Emanuel Moreira
João P. Vilela

Wavecom - Soluções Rádio, SA
Mestrado em Engenharia Informática

Julho 2013

Resumo

Num mundo cada vez mais informatizado, em que muitas das tarefas do quotidiano se começaram a processar por computador, surge a necessidade de criar sistemas cada vez mais robustos e fiáveis, de forma a garantir que os serviços por estes prestados sejam, tanto quanto possível, infalíveis. Este trabalho de estágio enquadra-se no ramo tecnológico dos Sistemas de Informação e visa essencialmente escolher, de entre várias, a solução de alta disponibilidade que melhores resultados oferece quando aplicada a serviços com necessidades de tempo real, mais precisamente voz, com base em métricas de Qualidade de Serviço pré-definidas. Depois de apresentado o Estado da Arte e de reter um conjunto de ferramentas e soluções de alta disponibilidade, serão definidos um conjunto de objectivos específicos a atingir, de forma a contribuir para a engenharia de sistemas e investigação nesta área. Posteriormente, apresentar-se-á a abordagem de estágio utilizada, os resultados experimentais obtidos e as vantagens e implicações que estes resultados trazem aos Sistemas de Informação.

Palavras-Chave

Alta Disponibilidade, Balanceamento de Carga, *Clustering*, FreeSWITCH, Redundância, Replicação, PostgreSQL, SIP, VoIP.

Índice

Resumo i

Palavras-Chave i

Lista de Figuras v

Lista de Tabelas vii

Acrónimos viii

1 Introdução e Objectivos 1
- 1.1 Introdução . . . 1
- 1.2 Solução IPCentrex . . . 3
- 1.3 Objectivos da Solução . . . 5
- 1.4 Enquadramento na Empresa . . . 6

2 Estado da Arte - Metodologias e Ferramentas 7
- 2.1 Alta Disponibilidade . . . 7
  - 2.1.1 Soluções de *Software* Livre . . . 7
  - 2.1.2 Soluções Comerciais de Alta Disponibilidade . . . 14
- 2.2 SIP e telefonia IP . . . 15
- 2.3 Balanceamento de Carga . . . 16
- 2.4 Replicação de Bases de Dados . . . 18
  - 2.4.1 Comparação de soluções de replicação de base de dados . . . 20
- 2.5 Virtualização . . . 26
  - 2.5.1 Vantagens da Virtualização . . . 27
  - 2.5.2 Tipos de Virtualização . . . 28
  - 2.5.3 Limitações da Virtualização . . . 30
  - 2.5.4 Limitações da Virtualização - Caso Wavecom . . . 30

3 Objectivos da Investigação e Método de Abordagem 32
- 3.1 Critérios de selecção/exclusão de tecnologias . . . 32
- 3.2 Arquitectura . . . 37
  - 3.2.1 Arquitectura Activo/Activo . . . 37
  - 3.2.2 Arquitectura Activo/Passivo . . . 38
- 3.3 Requisitos Funcionais, Não-Funcionais e Métricas . . . 39

4 Planeamento e Realização de Testes **41**
4.1 Descrição dos cenários de teste . . . 41
4.1.1 Cenário Activo/Passivo . . . 45
4.1.2 Cenário Activo/Activo . . . 47
4.2 Conclusões preliminares . . . 50
4.3 Testes Complementares . . . 51
4.4 Online Recovery . . . 55

5 Plano de Trabalho e Implicações **58**
5.1 Planeamento 1º Semestre . . . 58
5.2 Planeamento 2º Semestre . . . 60

6 Conclusões **63**

A Soluções de Telefonia IP **65**
A.1 FreeSWITCH . . . 65
A.2 Asterisk . . . 67

B Soluções de Replicação de Dados **70**
B.1 Bucardo . . . 70
B.2 Pgpool-II . . . 72
B.3 Pgpool-HA . . . 74
B.4 Rubyrep . . . 74
B.5 SkyTools - Londiste . . . 75
B.6 Slony . . . 77
B.7 Mammoth . . . 79
B.8 Postgres-XC . . . 80
B.9 Postgres-R . . . 81
B.10 NAS - *Network Attached Storage* . . . 82
B.11 DRBD . . . 83
B.12 *Log Shipping* . . . 84

C Soluções de Alta Disponibilidade no Mercado **89**
C.1 VMware vSphere . . . 89
C.2 Red Hat . . . 92
C.3 Oracle Solaris Cluster . . . 96
C.4 Suse . . . 98
C.5 Veritas Cluster Server . . . 100
C.6 Microsoft's Cluster Server . . . 101
C.7 HP Serviceguard . . . 102
C.8 IBM HA/CMP . . . 103
C.9 Lifekeeper . . . 104

D Soluções de Base de Dados **106**
D.1 SQLite . . . 106
D.2 PostgreSQL e MySQL . . . 109

E Requisitos Funcionais e Não-Funcionais **112**
E.1 Requisitos Funcionais 112
E.2 Requisitos Não-Funcionais 116

F Máquina Cliente - Configuração das Ferramentas de Monitorização e DNS **117**
F.1 Configuração inicial da máquina 118
F.2 Instalação do Nagios 118
F.3 Instalação do MRTG 121
F.4 Instalação do VoIPMonitor 138
F.5 Configuração do DNS 140

G Mecanismos de detecção de falhas **143**
G.1 *Script* de verificação do estado do serviço Apache 143
G.2 *Script* de verificação do estado do serviço Postgres 145
G.3 *Script* de verificação do estado do serviço FreeSWITCH 146
G.4 *Script* de verificação do estado do serviço Pgpool 147

Referências **149**

Lista de Figuras

1.1 Solução IPCentrex Wavecom 4

2.1 Estruturação do Cluster 8
2.2 Arquitectura LVS 12
2.3 UltraMonkey 17
2.4 Replicação via NAS 21
2.5 Replicação via DRBD 22
2.6 Funcionamento Pgpool-II 24
2.7 Pgpool-II com Streaming Replication 25

3.1 Arquitectura Activo/Activo 37
3.2 Arquitectura Activo/Passivo 38

4.1 Throughput e Jitter 45
4.2 Packet Loss e Delay 46
4.3 CPU e RAM 46
4.4 MOS 47
4.5 Throughput 48
4.6 Jitter e Packet Loss 48
4.7 Delay e MOS 49
4.8 CPU 49
4.9 RAM 50
4.10 Pgpool Online Recovery 56

5.1 Planeamento 1º Semestre 59
5.2 Planeamento 2º Semestre 61

B.1 PostgreSQL Queues 75
B.2 Processos independentes 76
B.3 Funcionamento Slony 78
B.4 Directório do WAL 84
B.5 Gestão de Transacções 85
B.6 Warm Standby 87
B.7 Streaming Replication - Hot Standby 88

C.1 Cluster Manager 94
C.2 Modelo Three Tier 95

F.1 Nagios Email 136
F.2 Nagios Interface 137

F.3 MRTG Interface 138

Lista de Tabelas

2.1 Soluções Livres para a criação de *clusters* de Alta Disponibilidade 13
2.2 Soluções Comerciais de Alta Disponibilidade 14
2.3 Soluções de Balanceamento de Carga 18

3.1 Métodos de Replicação do PostgreSQL 34
3.2 Ferramentas de Replicação do PostgreSQL 36

4.1 Especificações dos Cenários de Teste 44
4.2 Activo/Passivo 1 (parte 1) 51
4.3 Activo/Passivo 2 (parte 2) 52
4.4 Activo/Activo 1 (parte 1) 53
4.5 Activo/Activo 2 (parte 2) 54
4.6 Activo/Activo 3 (parte 3) 54

Acrónimos

ACID	**A**tomicidade **C**onsistência **I**solamento **D**urabilidade
ACK	**Ack**nowledgement
AIS	**A**pplication **I**nterface **S**pecification
API	**A**pplication **P**rogramming **I**nterface
cLVM	**c**lustered **L**ogical **V**olume **M**anager
CRM	**C**luster **R**esource **M**anager
DRBD	**D**istributed **R**eplicated **B**lock **D**evice
GFS	**G**lobal **F**ile **S**ystem
HTTP	**H**yper**T**ext **T**ransfer **P**rotocol
IMAP	**I**nternet **M**essage **A**ccess **P**rotocol
IP	**I**nternet **P**rotocol
IVR	**I**nteractive **V**oice **R**esponse
KVM	**K**ernel-based **V**irtual **M**achine
LVS	**L**inux **V**irtual **S**erver
MTBF	**M**ean **T**ime **B**etween **F**ailures
MTTR	**M**ean **T**ime **T**o **R**epair
OCFS	**O**racle **C**luster **F**ile **S**ystem
ODBC	**O**pen **D**ata**B**ase **C**onectivity
NAS	**N**etwork **A**ttached **S**torage
PBX	**P**rivate **B**ranch **E**xchange
PITR	**P**oint **I**n **T**ime **R**ecovery
PSTN	**P**ublic **S**witched **T**elephony **N**etwork
QoE	**Q**uality **O**f **E**xperience
QoS	**Q**uality **O**f **S**ervice
RDIS	**R**ede **D**igital **I**ntegrada de **Serviços**

RTP	**R**eal **T**ime **P**rotocol
SAN	**S**torage **A**rea **N**etwork
SBC	**S**ession **B**order **C**ontroller
SCSI	**S**mall **C**omputer **S**ystem **I**nterface
SGBD	**S**istema de **G**estão de **B**ase de **D**ados
SIP	**S**ession **I**nitiation **P**rotocol
SPOF	**S**ingle **P**oint **O**f **F**ailure
SRTP	**S**ecure **R**eal **T**ime **P**rotocol
TCP	**T**ransport **C**ommunication **P**rotocol
TI	**T**ecnologias da **I**nformação
UPS	**U**ninterruptible **P**ower **S**upply
VoIP	**V**oice **o**ver **IP**
VRRP	**V**irtual **R**outer **R**edundancy **P**rotocol
WAL	**W**rite **A**head **L**og

Capítulo 1

Introdução e Objectivos

Este capítulo pretende, de forma genérica, abordar o conceito de alta disponibilidade. Na introdução é clarificado o conceito e de que forma ele se insere no nosso quotidiano. Posteriormente, é apresentada a solução de telefonia IP da empresa, os objectivos que a solução a implementar deverá comportar e o meu enquadramento na empresa, face àquilo a que são as minhas responsabilidades e me compete realizar.

1.1 Introdução

Vivemos num mundo que cada vez mais depende da tecnologia e dos recursos que ela nos oferece. Cada vez mais as pessoas se tornam sedentárias tratando das suas tarefas recorrendo a serviços informatizados. O facto de, com a disponibilização de serviços como pagamento de facturas, carregamento de cartões de telemóvel, transferências bancárias, pagamento de serviços, compras em lojas, entre outros, podermos poupar tempo e dinheiro nas nossas deslocações, leva-nos a crer que estamos cada vez mais a caminhar num sentido sem retorno, em que a tecnologia impera e à qual cada vez mais, se exige.

Desta forma, surge o conceito de alta disponibilidade. Quer isto dizer que, face às exigências dos consumidores e também ao interesse económico das empresas, cada vez mais se evolua no sentido de procurar soluções tecnológicas capazes de responder a falhas no acesso aos seus serviços, bem como no tempo de inactividade destes, minimizando

assim o prejuízo que estes períodos de tempo acarretam para as empresas, já que tempo perdido é dinheiro perdido, são produtos não vendidos e, portanto, clientes perdidos.

Um sistema de alta disponibilidade é um sistema robusto e resiliente a falhas quer de *software*, *hardware* ou energia, que tem como objectivo principal a disponibilização dos seus serviços o máximo de tempo possível [1].

Muitos dos componentes de um sistema de informação consistem em *hardware*. Como tal, a fiabilidade destes sistemas revela-se insuficiente quando se trata da disponibilização de um serviço crítico. Desta forma, torna-se importante dispor de *hardware* redundante, um segundo sistema capaz de entrar em funcionamento sempre que for detectada uma falha no primeiro. Percebe-se que, quanto maior for o nível de redundância, menor será o número de pontos únicos de falha e, portanto, menor será a probabilidade de interrupções no(s) serviço(s) [2].

É neste contexto que surge a necessidade de também os operadores de telecomunicações se interessarem por este conceito. Torna-se inevitável, a partir do momento em que um operador queira garantir o nível máximo de satisfação dos seus clientes, a construção de sistemas capazes de garantir que os seus serviços se façam sem interrupções, garantindo uma Qualidade de Experiência o quanto maior possível [3].

Pretende-se assim, com este projecto, aferir, segundo um conjunto de métricas (Requisitos Funcionais, Não-Funcionais e Métricas), a solução de alta disponibilidade com melhor desempenho para disponibilizar serviços com necessidades de tempo-real, mais precisamente voz, com base nos protocolos SIP e RTP.

Numa primeira fase, são estudadas as ferramentas de *software* livre possíveis para a realização de uma solução de alta disponibilidade tendo em conta os requisitos de qualidade necessários para disponibilizar os diferentes serviços envolvidos no sistema: serviço WEB, serviço de telefonia IP e serviço de base de dados.

Numa segunda fase, é realizado o estudo comparativo das soluções implementadas e feita a avaliação crítica da *performance* destas, segundo as referidas métricas.

No final deste projecto crê-se que a presente proposta de alta disponibilidade possa ser aplicada a sistemas complexos e sensíveis como os sistemas de voz e que, o estudo comparativo entre soluções possa dar uma visão mais alargada das hipóteses possíveis, de modo a aferir a melhor solução numa perspectiva de custo-benefício.

O relatório está organizado da seguinte forma: ao longo do presente capítulo é mostrado o funcionamento da solução actual da empresa e é feito um levantamento dos requisitos da solução a implementar. No capítulo 2 é feita uma análise do estado da arte com foco nas ferramentas disponíveis para a realização da solução de alta disponibilidade, de acordo com os objectivos definidos. O capítulo 3 apresenta os objectivos da solução e a metodologia a adoptar, justificando as ferramentas seleccionadas e definindo a arquitectura e métricas a utilizar na validação da solução. No capítulo 4 são apresentados os cenários de teste e de que forma e sob que condições eles foram realizados. No capítulo 5 é mostrado o planeamento que foi tido em conta para o primeiro e segundo semestres, bem como a justificação dos prazos escolhidos para a execução das tarefas e os eventuais desvios. Finalmente, no capítulo 6, são tecidas considerações finais e é referido o que é suposto vir a ser implementado no futuro, no sentido de acrescentar valor à solução da empresa.

1.2 Solução IPCentrex

IPCentrex - Alto Nível

A Wavecom apresenta, como solução de telefonia IP, o IPCentrex. Esta solução possibilita aos clientes suprimir todo um conjunto de operações necessárias à manutenção de um sistema de telefonia, como a necessidade de gerir redundâncias de serviços, servidores e a necessidade de espaço para os armazenar, bem como o consumo energético que lhes está associado. É garantido ao cliente a manutenção dos serviços prestados, tornando a disponibilidade destes melhor e mais barata. Desta forma, o cliente poupa no tempo e recursos a que tarefas de gestão de um sistema obrigam.

Em substituição aos dispendiosos PBXs (*Private Branch Exchange*) físicos, o cliente mantém, através de um IP-PBX (PBX baseado em IP, também designado IPBX) virtual, todas as funcionalidades associadas sem qualquer responsabilidade sobre a disponibilidade e manutenção deste, tornando possível a gestão de vários escritórios de uma empresa, mesmo que estes estejam em localizações diferentes.

De forma a se perceber como esta solução funciona, apresenta-se a figura em baixo:

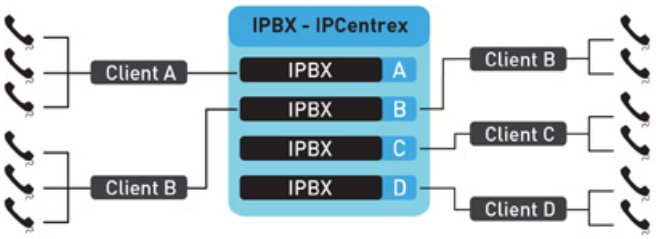

FIGURA 1.1: Solução IPCentrex Wavecom - fonte: http://www.wavecom.pt/pt/centrex.php?id_area=12

É possível verificar que as configurações de vários IPBX se concentram num único PBX virtual que é responsável pela gestão de todas as extensões associadas. Tratando-se de uma solução multi-domínio, os vários IPBX são disponibilizados sem recorrer à criação de máquinas virtuais, simplificando o seu processo de gestão e manutenção.

Por fim, a solução IPCentrex da Wavecom apresenta-se como uma solução escalável e robusta, de alto desempenho, oferecendo aos clientes uma solução comercial mais rentável.

IPCentrex - Baixo Nível

Para construir esta solução de telefonia IP, a Wavecom escolheu, de entre várias opções, o *sofware* livre, *OpenSource*, FreeSWITCH.

O FreeSWITCH é uma solução de telefonia IP bastante poderosa. Pode ser definido como um *soft-switch*[1] *OpenSource* multi-plataforma concebido para ser escalável e estabelecer a comunicação entre diferentes protocolos de comunicação, usando qualquer forma de *media*.

De forma a fazer a gestão do FreeSWITCH e a tornar a solução mais flexível e intuitiva para o cliente, a Wavecom apostou numa plataforma WEB, designada FsCloud, que divergiu de uma solução *OpenSource* e que permite a gestão e interacção com as funcionalidades que o FreeSWITCH oferece.

Finalmente, para garantir a manutenção dos dados relacionados com o FreeSWITCH e com a implementação de novos módulos adicionados ao FsCloud, foi escolhido como SGBD o PostgreSQL.

[1] Aplicação de *software* que interliga telefones entre redes, frequentemente encaminhando chamadas de um protocolo para outro ou para um terminal como um PBX físico

Estas tecnologias foram seleccionadas *a priori* pela instituição e fazem parte da sua solução de telefonia IP. No entanto, apresento no anexo A - Soluções de Telefonia IP informação mais detalhada sobre as ferramentas de telefonia IP mais utilizadas actualmente na construção de sistemas IPBX.

1.3 Objectivos da Solução

De acordo com o pretendido, para a criação de um sistema de alta disponibilidade importa saber que ferramentas existem para que, no caso de uma falha, o sistema saiba que recursos estão em falha e possa iniciá-los automaticamente, de forma transparente para o utilizador.

Como disse na secção anterior, há três tipos de serviços que, para o completo funcionamento da solução de telefonia a implementar, deverão estar sempre operacionais:

- Serviço de telefonia IP, FreeSWITCH;
- Serviço WEB, FsCloud;
- Serviço de Base de Dados, PostgreSQL.

Estes serviços serão tratados de forma independente na solução de alta disponibilidade. Por um lado, sempre que a máquina onde os serviços de telefonia IP e o serviço WEB estarão a ser executados falhar, terá de haver *software* capaz de detectar essa falha e, num processo designado de *failover*, recuperar estes mesmos serviços, activando-os e iniciando-os numa outra máquina. Por outro lado, para o serviço de Base de Dados será necessário *software* adicional que garanta que os dados alterados numa máquina sejam replicados para outra, de forma a que, no caso de uma falha nessa máquina, estes dados não se percam e estejam disponíveis na outra. As ferramentas de replicação podem ser consultadas no capítulo 2, secção Replicação de Bases de Dados.

No capítulo seguinte poderemos ver de que forma podemos replicar estes serviços e de que ferramentas dispomos para o fazer.

1.4 Enquadramento na Empresa

Ao nível das telecomunicações a empresa apresentava-se com uma solução de telefonia IP recente, em desenvolvimento, que nos remonta ao ano de 2012. Numa fase inicial tratava-se de uma solução de telefonia IP simples, com alguns aspectos interessantes como a fácil gestão de extensões, domínios e *gateways*, despertando a atenção de alguns clientes pequenos. Até aqui, a empresa lidava bem com alguma falha que pudesse ocorrer, conseguindo muito rapidamente repor os serviços que estivessem em baixo.

Posteriormente, com o crescimento da solução e visibilidade externa, esta solução ganhou alguma notoriedade e conseguiu reunir mais alguns clientes, um tanto ou quanto importantes. Começou-se, então, a pensar na construção de uma solução de alta disponibilidade que oferecesse redundância de serviços de telefonia IP e todos os serviços associados à sua disponibilização, como base de dados e serviço WEB. Nesta fase a empresa não apresentava qualquer solução de redundância dos seus serviços e é nesta fase que entro e se centram as minhas responsabilidades.

É, desde logo, necessária uma validação e comparação de soluções possíveis para a replicação da base de dados em tempo-real, de mecanismos que garantam que a comunicação entre extensões de um cliente não se faça apenas por uma máquina e que uma máquina nunca constitua um único ponto de falha. Portanto, no que respeita a disponibilidade, a empresa não apresentava qualquer solução criada e era minha responsabilidade criar uma solução de alta disponibilidade fiável e robusta. No entanto, importa referir que todas as decisões tomadas, para além da fundamentação teórica documentada ao longo deste relatório, justificando o porquê do uso de determinadas tecnologias em detrimento de outras é também corroborada e apoiada pela equipa em que trabalho, pela experiência dos profissionais que me permitem aferir, de forma mais directa, o caminho a seguir. Importa ainda referir que os membros desta equipa trabalham na construção de módulos adicionais para o enriquecimento da plataforma WEB e que todos esses módulos, uns mais que outros, contribuem para o bom desempenho dos nossos servidores.

Capítulo 2

Estado da Arte - Metodologias e Ferramentas

Neste capítulo são descritas as principais ferramentas que fazem face aos requisitos da solução a implementar. Em primeiro lugar são apresentadas algumas ferramentas que permitem a criação de um *cluster* de alta disponibilidade em Linux, plataforma segundo a qual será construída a solução. Em segundo lugar, de forma a contextualizar a solução no mercado, o leitor é remetido para um conjunto de soluções comerciais. Finalmente, são listadas as metodologias suportadas pelo PostgreSQL para a realização da replicação de uma base de dados entre diferentes máquinas.

2.1 Alta Disponibilidade

2.1.1 Soluções de *Software* Livre

Linux-HA - *High Availability Linux Project*

High Availability Linux Project ou Linux-HA é um projecto *OpenSource* dedicado à implementação de soluções de alta disponibilidade para *clusters*. Suporta Linux, permitindo construir sistemas robustos e resilientes a falhas [4].

A principal ferramenta deste projecto é o Heartbeat que permite a comunicação entre os nós de um *cluster*. Inicialmente esta ferramenta apenas suportava dois nós e carecia

de funcionalidades, não existindo qualquer tipo de gestor de recursos dos nós. Posteriormente, foi implementada uma solução que veio permitir criar *clusters* com N nós, gestor de recursos, dependências e políticas. Actualmente, a componente do gestor de recursos do *cluster*, responsável pela monitorização, arranque e paragem dos recursos está separada do Heartbeat e deu lugar a um novo projecto, designado Pacemaker [5].

De forma a perceber em que camada é que as ferramentas se localizam no *cluster* e de maneira a tornar a visualização mais fácil, segue-se a figura abaixo que serve de referência ao longo da apresentação das ferramentas *OpenSource* de alta disponibilidade de que dispomos.

FIGURA 2.1: Estruturação do Cluster - Adaptado de: http://clusterlabs.org/wiki/Pacemaker

Heartbeat

O Heartbeat é um projecto criado pela Linux-HA que funciona como gestor do *cluster*. Localiza-se na camada de *Messaging* que é responsável pela comunicação entre os nós.

Através de pacotes pequenos, designados *heartbeats*, é feita a sinalização de presença dos nós de um *cluster*. Um servidor redundante verifica se o principal está disponível através do envio de uma mensagem *heartbeat*. Se o servidor principal não responder a esta mensagem, é considerado indisponível e o servidor redundante assume automaticamente o controlo dos serviços que eram disponibilizados pelo outro servidor, preservando o funcionamento do sistema [6].

De forma a ser útil para os utilizadores, o Heartbeat pode ser combinado com um CRM (*Cluster Resource Manager*) que é responsável pela paragem e pelo arranque dos serviços que o *cluster* tornará disponíveis [4].

As suas principais características são:

- Não existe um número máximo de nós. O Heartbeat pode ser usado tanto para construir pequenos, como grandes *clusters*;
- Monitorização de recursos: os recursos podem ser iniciados automaticamente ou movidos para outro nó na ocorrência de uma falha;
- Mecanismo de detecção e remoção de nós falhados do *cluster*;
- Gestão de recursos sofisticada baseado em políticas (que serviço iniciar e em que máquina iniciar), interdependências e restrições;
- Regras que permitem diferentes políticas ao longo do tempo;
- Existência de vários *scripts* de recursos como Apache, DB2, Oracle, PostgreSQL, etc.;
- GUI que permite a configuração, controlo e monitorização dos recursos e nós do *cluster*.

OpenAIS/Corosync

Tal como o Heartbeat, também o OpenAIS/Corosync se localiza na camada de *Messaging* de um *cluster*. Originalmente o OpenAIS/Corosync era um só projecto, separando-se posteriormente em dois projectos diferentes. O projecto Corosync foi formalmente anunciado numa conferência, em Julho de 2008 [7], tendo-se separado do OpenAIS que sofreu algumas alterações significativas. As características inerentes à camada de *Messaging* são agora atribuídas ao Corosync.

Os principais contribuidores do projecto OpenAIS decidiram não continuar a desenvolver implementações adicionais, dedicando o seu tempo a manter um *roadmap* para o Corosync. Acreditam que o Pacemaker combinado com o Corosync são a escolha ideal para a solução de alta disponibilidade [8].

O Corosync apresenta-se como uma ferramenta posterior ao Heartbeat, fornecendo características adicionais para implementar uma solução de alta disponibilidade, através de quatro APIs [9]. São elas:

- Um modelo de comunicação com garantias de sincronização virtual para a criação de máquinas de estado replicadas;
- Um gestor de disponibilidade que reinicia o processo das aplicações quando ele falhar;
- Uma base de dados em memória para configuração e estatísticas que fornecem a capacidade de definir, recuperar, e receber notificações de alteração de informação;
- Um sistema de quorum[1] que notifica as aplicações quando o quorum é alcançado ou perdido.

Pacemaker - Gestor de Recursos

O Pacemaker é um gestor do *cluster* que permite iniciar, parar e monitorizar recursos. A alta disponibilidade é conseguida através da detecção e recuperação de falhas nos nós e/ou serviços. Isto é possível devido às capacidades de comunicação fornecidas pela infraestrutura do *cluster* que preferirmos adoptar (Heartbeat ou Corosync). O Heartbeat foi a primeira *stack* a ser suportada pelo Pacemaker, enquanto que o Corosync foi a segunda [10].

O Pacemaker, por si só, apenas necessita do Corosync de forma a funcionar. Contudo, algumas aplicações que ele pode ser responsável por gerir, como o OCFS2 e o GFS2, que são dois tipos de sistemas de ficheiros partilhados, requerem também o OpenAIS.

Características:

- Detecção e recuperação de falhas nos nós e serviços do *cluster*;
- É independente do armazenamento, não necessitando de armazenamento partilhado;
- É independente dos recursos. Qualquer recurso que consiga ser posto em *script* pode fazer parte do *cluster*;

[1] Diz-se que um *cluster* tem quorum quando mais de metade dos nós estão *online*. No caso de de um *cluster* com dois nós, terá de ter os dois nós *online*.

- Suporta pequenos e grandes *clusters*;
- Suporta praticamente qualquer tipo de configuração de redundância incluindo as seguintes arquitecturas: Activo/Activo, Activo/Passivo, N+1, N+M, N-para-1 e N-para-N;
- Opcionalmente, assegura a integração de dados com o STONITH, que é uma ferramenta que se encarrega de eliminar os nós que falharam, impedindo-os de realizar acções que possam vir a corromper dados, como por exemplo a escrita concorrente num sistema de ficheiros partilhado;
- Capacidade de especificação da ordem de execução dos serviços em grandes *clusters*;
- Suporte de serviços que devem estar activos em múltiplos nós;
- Suporte de serviços com múltiplos modos (mestre/escravo);
- Existência de uma *cluster shell*.

Em conclusão, o Pacemaker mantém as aplicações em execução mesmo quando estas, ou as máquinas onde elas correm, falham. Suporta outras implementações, como o Heartbeat e o Corosync que, como vimos, fornecem um mecanismo fiável de troca de mensagens entre os nós, notificando quando as máquinas aparecem ou desaparecem e ainda uma lista das máquinas que estão activas no *cluster*.

LVS - *Linux Virtual Server*

O objectivo principal deste projecto é poder criar um *cluster* de alto desempenho e alta disponibilidade para sistemas Linux, fornecendo escalabilidade, confiabilidade e a disponibilidade dos serviços. A arquitectura é transparente para os utilizadores que interagem com o sistema, como se de um único servidor se tratasse [11].

Normalmente é adoptada uma arquitectura *three-tier* como mostra a figura seguinte:

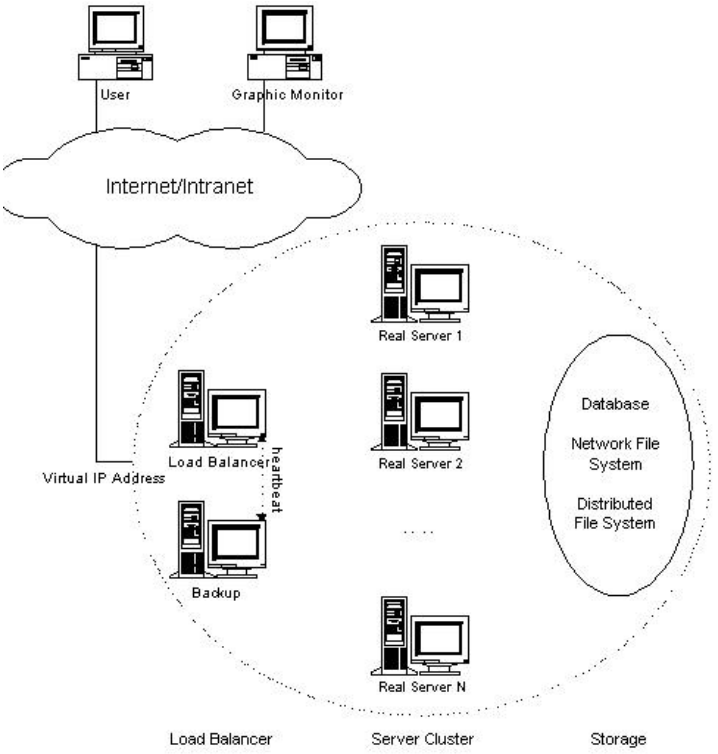

FIGURA 2.2: Arquitectura LVS - fonte: http://www.LinuxVirtualServer.org/architecture.html

O *Load Balancer* é a máquina *frontend* do *cluster* e faz o balanceamento dos pedidos dos utilizadores pelo conjunto de servidores existentes. Desta forma, o cliente acede aos serviços que estão alojados nos servidores de forma transparente, a partir de um único IP [12].

O *Server Cluster* é um conjunto de servidores que disponibilizam os serviços do sistema.

Finalmente, a camada de armazenamento disponibiliza os dados necessários aos servidores, de forma a que possam fornecer os mesmos serviços e partilhem os mesmos dados.

Para o funcionamento do sistema, existe um serviço no servidor de balanceamento de carga que monitoriza o estado dos servidores. No caso de não haver conectividade com algum servidor, este é removido da lista de servidores disponíveis. Desta forma, o balanceador de carga tem a particularidade de mascarar a falha de serviços ou servidores.

Como o balanceador de carga também pode constituir um SPOF (*Single Point of Failure*), existe um de *backup*. Existem dois *daemons* Heartbeat em execução no primeiro balanceador de carga e no segundo de *backup*. Quando o Heartbeat do servidor de *backup* não consegue receber a mensagem do primeiro no espaço de tempo definido, irá assumir

o controlo, respondendo a pedidos dos clientes e distribuindo-os de igual forma entre os servidores [13].

Keepalived

O Keepalived é uma ferramenta *OpenSource* escrita em C com o objectivo de fornecer balanceamento de carga e alta disponibilidade para sistemas Linux. Basicamente, consiste na reinicialização de processos que providenciam essas características. Disponibiliza uma *framework* para balanceamento de carga que consiste no IPVS (*IP Virtual Server*), que é um módulo do Kernel que permite ter um servidor virtual, responsável pelo redireccionamento de pedidos de clientes para os servidores reais. Para além disso, implementa um conjunto de verificadores que fazem a gestão da *pool* de servidores de balanceamento de carga do sistema. A alta disponibilidade é conseguida através do protocolo VRRP que permite o *failover* entre máquinas. Consiste num *daemon* que é separado em três processos. O processo pai verifica se o processo responsável pela verificação dos balanceadores de carga está activo e também se a *framework* VRRP está a funcionar, através do envio de uma mensagem periódica "hello". Basicamente, consiste na reinicialização automatizada de serviços no caso da detecção de uma falha.

Em tom de resumo apresenta-se a tabela seguinte, respeitante às tecnologias livres que possibilitam a construção de *clusters* de alta disponibilidade.

Solução	**Licença**	**Várias Arquitecturas**	**Balanc. de Carga**	**Multi-recursos**	**Plataforma**
Heartbeat	*OpenSource*	✓	×	×	Linux
OpenAIS/ Corosync	*OpenSource*	✓	×	✓	Linux
LVS	*OpenSource*	×	✓	×	Linux
Keepalived	*OpenSource*	×	✓	×	Linux

Tabela 2.1: Soluções Livres para a criação de *clusters* de Alta Disponibilidade

Esta tabela permite-nos concluir que, apesar de todas as ferramentas apresentadas serem *OpenSource*, o LVS e o Keepalived não nos permitem criar grandes *clusters*, uma vez que foram desenhados para topologias mais simples, possibilitando, contudo, o balanceamento de carga entre dois servidores. Se um dia for necessário escalar a solução de alta disponibilidade em questão, será necessário adoptar uma ferramenta com essas características, como é o caso do Heartbeat e do OpenAIS/Corosync que possibilitam a

construção de *clusters* de maior dimensão. No entanto, tal como é visível, o OpenAIS/-Corosync é a ferramenta que nos garante um leque maior de funcionalidades, tal como foi referido ao longo deste capítulo.

2.1.2 Soluções Comerciais de Alta Disponibilidade

Foi possível efectuar um apanhado de algumas soluções de alta disponibilidade com suporte para *failover*, balanceamento de carga e outras características, que se apresentam na tabela em baixo. No entanto, para maior detalhe é possível consultar a informação referente a cada uma das soluções apresentadas que fiz questão de descrever ao longo do Anexo C - Soluções de Alta Disponibilidade no Mercado.

Solução	**Ambiente (Físico/ Virtual)**	**Monito-rização**	**Balanc. de Carga**	**Replic. de Dados**	**Armazen. Partilhado**	**Plataforma**
VMware VSphere	Virtual	✓	✓	✓	×	Win Linux
Red Hat Cluster Suite	Físico	✓	✓	✓	✓	Linux
Oracle Solaris Cluster	Físico	✓	✓	✓	✓	Linux
Suse HA	Físico Virtual	✓	✓	✓	✓	Linux
Veritas Cluster	Físico Virtual	✓	✓	×	✓	Win Linux
Microsoft Cluster Server	Físico	✓	✓	✓	✓	Win
HP Service Guard	Físico	✓	✓	✓	✓	Linux
IBM HA/CMP	Físico	✓	×	✓	✓	Linux
Lifekeeper	Físico	✓	×	✓	✓	Win Linux

Tabela 2.2: Soluções Comerciais de Alta Disponibilidade

Dependendo do ambiente que desejarmos adoptar (físico ou virtualizado), é possível observar que estas soluções comerciais são muito completas. Apesar de algumas não disporem nativamente de características como o balanceamento de carga ou a replicação de dados, isto pode ser contornado através da instalação de ferramentas complementares,

quer através de extensões da própria marca ou através do suporte a outras ferramentas, também designadas de *third-party*.

2.2 SIP e telefonia IP

Trabalhos Relacionados e Aspectos Inovadores

Sobre a alta disponibilidade encontram-se vários trabalhos que se dedicam ao estudo de diversos fins. No entanto, não se relacionam directamente com o foco desta tese (a telefonia IP), divergindo para outras temáticas como por exemplo o estudo do desempenho da alta disponibilidade em *clusters* de servidores WEB, o impacto causado pelo *failover* de uma base de dados no desempenho de um sistema, os efeitos da tecnologia redundante num sistema, entre outros.

No que respeita a SIP e à telefonia IP, a quantidade de trabalhos desenvolvidos é substancialmente menor, o que é entendível pela especificidade do tema. Contudo, é possível encontrar alguns trabalhos sobre alta disponibilidade relacionados com o *software* Asterisk.

Uns propõem um modelo de *failover* baseado em *middleware* e em IP virtual, onde existem dois servidores com o *software* Asterisk instalado, comportando-se um como nó activo e outro como passivo. No caso de uma falha, o *middleware* do nó passivo detecta-a e atribui automaticamente o IP virtual a esse nó, para o qual será encaminhado todo o tráfego [14]. Outros há que propõem um algoritmo de balanceamento de carga capaz de, através do conhecimento profundo do funcionamento do protocolo SIP e do estudo de aspectos como a distinção de tráfego, estimativa da carga no servidor e reconhecimento das diferenças no processamento de diferentes transacções SIP, escalar mais do que outras abordagens tradicionais de balanceamento que não são tão refinadas [15]. Outro exemplo, não menos importante, até pelas métricas que avalia, é o caso de um trabalho realizado com o intuito de perceber até que ponto ter um *cluster* de servidores SIP *proxy* pode ser fundamental na disponibilidade de serviços SIP como voz e conferências [16].

Dada a natureza e complexidade deste projecto, acredito que o desenvolvimento de uma solução de alta disponibilidade para um sistema de telefonia IP será um contributo importante para a engenharia e sobretudo para a área de conhecimento dos Sistemas de Informação, uma vez que não existe uma solução *off-the-shelf* para a construção de um sistema de alta disponibilidade com estas necessidades. Para além disso, o projecto

FreeSWITCH, apesar de não ser o mais utilizado, é um projecto que está cada vez mais a ganhar adeptos e, tratando-se de uma tecnologia emergente, parece lógico entender que esta solução poderá despertar a atenção de futuros investigadores nesta área. O facto de, para além de garantir a disponibilidade do sistema de telefonia IP, incluir replicação de bases de dados em tempo-real entre as máquinas do *cluster*, acrescenta valor a este sistema, tornando-o particular.

2.3 Balanceamento de Carga

O FreeSWITCH suporta algumas ferramentas que oferecem balanceamento de carga, de entre as quais se destacam o DNS, o OpenSIPS e o UltraMonkey [17].

DNS

O balanceamento de carga com recurso ao DNS apresenta várias vantagens como a distribuição dos pedidos dos clientes com base na definição de prioridades em cada um dos nós. Imaginemos que uma máquina disponibiliza mais serviços que a outra. Neste caso, poder-se-ia ter uma prioridade de chegada de pedidos de clientes a uma máquina de 60%, enquanto que a outra apenas lidaria com 40% dos pedidos, não a sobrecarregando em demasia. Trata-se de uma solução simples de configurar e fácil de gerir. Para além disso, no caso de uma falha na máquina do DNS, poderá ser utilizado um DNS secundário. No entanto, apresenta algumas desvantagens como o uso do algoritmo de *Round Robin* para definir a probabilidade com que os pedidos são distribuídos pelas máquinas. Desta forma, esta técnica não olha para o consumo de recursos de cada uma das máquinas e, como tal, um pedido pode ser encaminhado para a máquina com maior carga, sobrecarregando-a ainda mais.

OpenSips

O OpenSips é um servidor SIP *OpenSource* que permite, através de um módulo dedicado para o balanceamento de carga, fazer o encaminhamento do tráfego com base na carga dos servidores. Quando o OpenSips encaminha chamadas para os servidores de destino, consegue verificar o estado dos servidores através do número de chamadas em curso a que cada um está a responder num determinado momento, de forma a optar pelo servidor com menos carga. O OpenSips sabe, por omissão, qual a capacidade máxima de cada servidor de destino. Assim, durante o encaminhamento, considerará não o

destino com o menor número de chamadas em curso, mas antes, aquele que tiver mais recursos disponíveis.

UltraMonkey

O UltraMonkey é um projecto *OpenSource* criado para fornecer balanceamento de carga e alta disponibilidade que se apresenta como uma solução fácil de instalar e gerir. Suporta um conjunto largo de protocolos como o DNS, LDAP, FTP e HTTP, com verificadores nativos da sua actividade nas máquinas, entre outros. Funciona com base nos projectos *OpenSource* LVS e Ldirectord, suportando pequenos e grandes *clusters*, escalando até milhares de ligações por segundo. O seu foco destina-se ao fornecimento de balanceamento de carga através de requisitos mínimos de *hardware*. O seu funcionamento pode ser visto na figura em baixo:

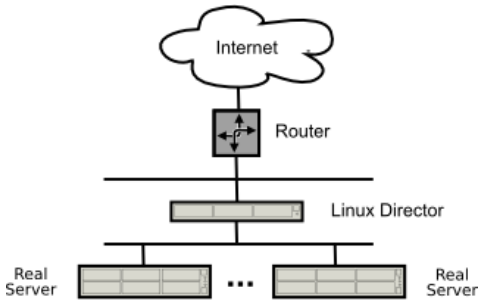

FIGURA 2.3: UltraMonkey - fonte: http://www.ultramonkey.org/3/topologies/lb-eg.html

O Linux-Director funciona como um *gateway* para a rede e faz o balanceamento do tráfego com base no LVS. Ao receber uma ligação de um utilizador, toma uma decisão com base nos servidores reais, encaminhando o tráfego para um deles. O Ldirectord é um *daemon* que realiza a monitorização do estado dos servidores reais. Se um falhar, o servidor será descartado da *pool* de servidores reais e será novamente inserido quando estiver de novo *online*.

Para concluir, segue-se a tabela abaixo que sumariza os pontos fundamentais relativos às soluções de balanceamento de carga apresentadas.

Solução	**Licença**	**Puro**	**Balanc. de Carga**	**Percepção de Carga**	**Plataforma**
DNS	N/A	×	✓	×	N/A
OpenSips	*OpenSource*	✓	✓	✓	Linux
Ultra Monkey	*OpenSource*	✓	✓	×	Linux

Tabela 2.3: Soluções de balanceamento de carga

Nesta tabela é possível verificar que, embora o DNS não seja um balanceador de carga puro, ou nativo, ele consegue fazê-lo de forma muito simples, podendo inclusivamente atribuir pesos aos servidores, encaminhando mais ou menos volume de tráfego, consoante as necessidades. Quanto ao OpenSips e ao UltraMonkey, apresentam-se como duas boas escolhas para balanceamento de carga. No entanto, o OpenSips tem a vantagem de perceber a carga actual dos servidores-alvo, tomando uma decisão com base nesta percepção.

2.4 Replicação de Bases de Dados

Um servidor de base de dados pode ser suficiente numa solução em que existem vários servidores de aplicação e algumas centenas de clientes. Contudo, com a alta disponibilidade, surge a necessidade de ter esses dados replicados num outro servidor de base de dados, de forma a que, aquando um *failover*, os dados estejam sempre disponíveis. Para além disso, a replicação de bases de dados é útil não só em termos de disponibilidade de um serviço como também poderá ser útil na construção de um *cluster* com necessidades de alto desempenho e balanceamento de carga. Pretende-se ter um *cluster* capaz de manter os seus serviços o máximo de tempo disponíveis, mas também, tirar partido de uma solução de replicação capaz de fazer balanceamento de carga e aumentar a *performance* global do nosso sistema.

Os servidores WEB, ao servirem páginas estáticas, podem ser facilmente combinados entre si para fazer balanceamento de carga dos pedidos HTTP feitos pelos utilizadores. De facto, os servidores de bases de dados *read-only*, podem ser combinados facilmente também. A questão é que, infelizmente, muitos dos servidores de bases de dados recebem tanto pedidos de operações de leitura, como de escrita, tornando estes servidores mais difíceis de combinar. Isto porque, os dados de leitura só precisam de ser colocados uma única vez em cada servidor, enquanto que a escrita de dados tem de ser propagada por

todos os servidores sempre que uma escrita for feita, de forma a que as futuras leituras de dados a estes servidores sejam consistentes nos seus resultados. Por exemplo, a execução de duas *queries* iguais a servidores diferentes deverão retornar o mesmo resultado.

Este problema de sincronização é a dificuldade principal que deve ser ultrapassada para que os servidores possam trabalhar em conjunto. Como não há uma única solução capaz de eliminar o impacto do problema da sincronização para todos os casos de uso possíveis, surgem várias soluções que tentam resolver este problema, cada uma à sua maneira.

Os servidores de bases de dados distinguem-se dos outros tipos de servidores na medida em que é necessário garantir a atomicidade das alterações dos dados, ao passo que, em servidores WEB, DNS e servidores de aplicação, que contêm dados estáticos, isso não acontece. Por esta razão, as soluções de replicação de bases de dados são mais complexas e podem ser classificadas quanto a **sincronia**, **escrita**, **fragmentação**, **envio** e **modo**.

Quanto à **sincronia**, podem distinguir-se duas formas de replicação:

- **Síncrona**: é realizado o *commit* de uma transacção em todos os servidores em simultâneo. Isto garante que um *failover* não implicará a perda de quaisquer dados e que todos os servidores retornarão dados consistentes, não importando qual o servidor onde foi executada determinada *query*.
- **Assíncrona**: uma transacção é feita num único servidor e depois propagada para os restantes, havendo a possibilidade de algumas transacções serem perdidas aquando uma troca de um servidor por um outro de *backup*, tornando eventualmente os resultados retornados pelos restantes servidores, inconsistentes. Por norma, a replicação assíncrona é usada quando a comunicação síncrona se verifica muito lenta.

Quanto à **escrita**, os dados tanto podem ser escritos num único servidor mestre, como em múltiplos servidores-mestre em simultâneo. Os servidores podem classificar-se como **mestres** e **escravos**. O servidor mestre (ou primário) é aquele que aceita escritas de dados, enquanto que o servidor escravo (secundário ou *standby*), só aceita leituras/consultas de dados.

Quanto à **fragmentação**, os dados são divididos por vários servidores. Consideram-se dois tipos de fragmentação:

- **Horizontal**: cada linha de uma relação ou tabela é atribuído a um ou mais fragmentos. Por exemplo: linhas com o campo Alberto na coluna "Nome" vão para um fragmento, enquanto que linhas com outro nome no mesmo campo, vão para outro fragmento.
- **Vertical**: o *schema* de uma relação ou tabela é dividido em vários mais pequenos. Por exemplo: se tivermos uma tabela com quatro colunas, dividimo-la em duas de duas colunas.

Quanto ao **envio**, podemos ter:

- **Envio de ficheiro (*file-based Log Shipping*)**: o ficheiro de *logs* de transacção é enviado aos servidores secundários. Normalmente esta técnica é combinada com a replicação assíncrona, já que estes ficheiros são temporários e aplicados posteriormente nos servidores *standby*;
- **Envio de registo (*record-based Log Shipping*)**: depois de uma transacção ser feita (*commit*), o *log*, ou registo da transacção, é enviado e aplicado nos servidores *standby* que participam na replicação.

Finalmente, quanto ao **modo**, podemos ter:

- ***Warm Standby***: os servidores secundários não aceitam escritas nem leituras de dados.
- ***Hot Standby***: os servidores secundários aceitam ligações, permitindo consultas que não modifiquem os dados.

2.4.1 Comparação de soluções de replicação de base de dados

Existem vários métodos de replicação de bases de dados, segundo os quais, algumas ferramentas se regem. Em baixo destacam-se os métodos de replicação que o PostgreSQL suporta, bem como de algumas ferramentas que os implementam. O estudo destas ferramentas teve por base a consulta da documentação oficial do PostgreSQL. [1]

[1] Disponível *online* em http://www.postgresql.org/docs/9.2/interactive/high-availability.html Consultado a 8/11/2012

Shared Disk Failover

Este método permite ter uma cópia da base de dados num dispositivo de armazenamento de rede. Uma vez que existe apenas uma única cópia da base de dados, este método evita um *overhead* de sincronização de dados. Usa um único *array* de discos que é partilhado pelos servidores. Desta forma, se a base de dados principal falhar, o servidor *standby* está disponível para a montar e iniciar no caso de se estar a recuperar de uma falha, permitindo um rápido *failover* sem perda de dados.

Um problema deste método é que o dispositivo de armazenamento partilhado constitui um SPOF. Isto é, se o *array* de discos falhar ou ficar corrompido, tanto o servidor primário como o secundário ficarão inoperacionais. Para além disso, o servidor secundário não pode aceder ao armazenamento enquanto o servidor primário o estiver a fazer, inviabilizando o poder que esta solução poderia trazer para o sistema.

Na figura abaixo pode ver-se o funcionamento desta técnica de replicação de dados:

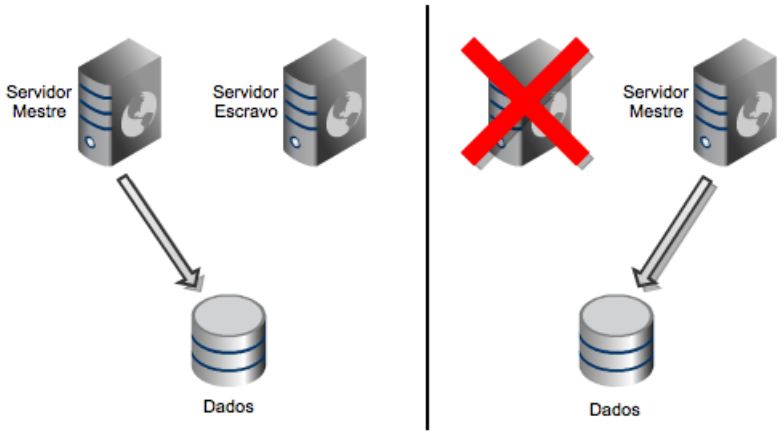

FIGURA 2.4: Replicação via NAS

Do lado esquerdo da figura podemos observar que o servidor mestre detém os dados, enquanto que o servidor escravo não está ligado ao armazenamento. Usando esta tecnologia, o servidor escravo não pode executar *queries*, sob pena de alterar os dados numa situação de acesso concorrente.

Quando o servidor mestre falha (lado direito da figura), o antigo servidor escravo é promovido a mestre e detém os dados, podendo responder a *queries* dos clientes.

Replicação do Sistema de Ficheiros

Neste método todas as alterações num sistema de ficheiros são espelhadas num sistema de ficheiros existente noutro computador. O DRBD (*Distributed Replicated Block Device*) é uma solução muito comum de replicação de sistemas de ficheiros para Linux e a figura abaixo exemplifica o seu uso:

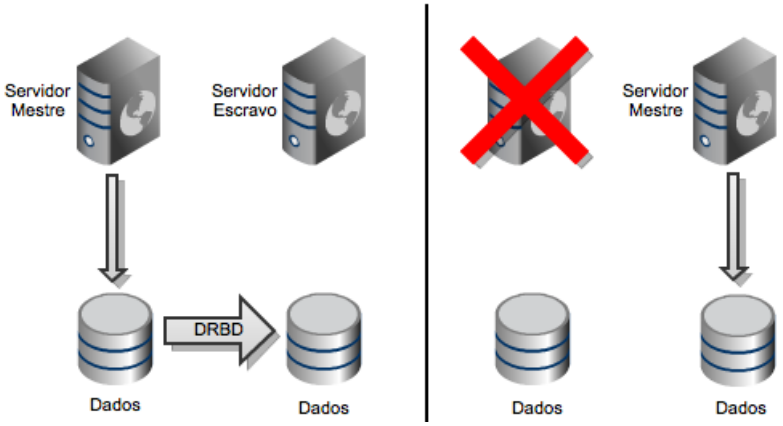

FIGURA 2.5: Replicação via DRBD

No nó activo a partição está montada com o DRBD, ao contrário do nó *standby* que é impedido de realizar quaisquer operações de leitura de dados. O espelhamento dos dados pode ser feito de modo síncrono ou assíncrono. A implementação de um *cluster* de alta disponibilidade, dependendo dos requisitos da solução, poderá seguir o modo síncrono onde não queremos perder quaisquer dados, no caso de uma falha do servidor activo.

Transaction Log Shipping

Através da leitura de ficheiros de *log* designados WAL, os servidores secundários *Warm Standby* e *Hot Standby* podem ser mantidos actualizados. Desta forma, caso o servidor primário falhe, o servidor secundário pode rapidamente recriar um servidor primário.

O arquivamento contínuo de *logs* de transacção pode ser usado para criar um *cluster* de alta disponibilidade com um ou mais servidores *standby* preparados para tomar o controlo das aplicações no caso de o servidor principal falhar. Esta capacidade é vulgarmente referida como ***Warm Standby*** ou ***Log Shipping*** e é geralmente aplicada em *clusters* sem necessidade de sincronização de dados, em que a disponibilização imediata dos ficheiros nos servidores *standby* não é importante dado que estes estão inactivos e apenas entrarão em funcionamento no caso de o servidor principal falhar.

Por outro lado, enquanto que numa configuração *Warm Standby* apenas os registos num ficheiro WAL completamente escrito são enviados (*Log Shipping* baseado em ficheiros), na *Streaming Replication* quaisquer registos escritos num ficheiro de *log* parcialmente preenchido são enviados (*Log Shipping* baseado em registos). Isto significa que a janela de perda de dados na *Streaming Replication* é muito menor do que no *Warm Standby*. A introdução deste conceito permite não só aumentar a rapidez de transferência e actualização de dados ao longo dos servidores, como permite a configuração de *clusters Hot Standby* em que os servidores *Standby* podem responder a *queries* de consulta de dados, retornando sempre dados consistentes.

Esta metodologia foi aprofundada e pode ser visualizada com maior detalhe no Anexo B - *Log Shipping*.

Replicação Mestre-Escravo *Trigger-Based*

Numa replicação Mestre-Escravo todas as *queries* de alteração de dados são enviadas ao servidor principal que, por sua vez se encarrega de, assincronamente, as enviar ao servidor secundário. O servidor secundário pode responder a pedidos de consultas de dados.

O **Slony** é um exemplo deste tipo de replicação. Para além de permitir múltiplos servidores escravo, permite a execução de *queries* de consulta de dados, sendo ideal para *Data Warehouse*. Por ser um tipo de replicação assíncrono, pode ser usado em redes de ligações lentas. Para além disso, oferece granularidade por tabela permitindo a fragmentação dos dados ao longo da base de dados. Dispõe de um processo chamado Slonik para onde são enviadas as *queries*. Posteriormente, através de *triggers*, esse processo é responsável por enviar as *queries* ao servidor primário que, por sua vez, se encarrega de enviar os dados assincronamente para o(s) servidor(es) escravo(s).

No entanto, por ser assíncrona, há a possibilidade de perda de dados durante o *failover*.

Replicação através de *middleware*

Neste tipo de replicação existe um *middleware* que é responsável por interceptar os pedidos de execução de *queries* ao servidor de base de dados. Ao fazê-lo, decide enviar essas *queries* para um ou para todos os servidores, de forma a distribuir o processamento. Poderá também, fazer o balanceamento de carga das *queries* a executar, escolhendo o servidor de base de dados para o qual deverá enviar a *query*.

As *queries* de escrita devem ser enviadas para todos os servidores de forma a que todos os servidores recebam as alterações efectuadas. Por outro lado, as *queries* de consulta de dados podem ser enviadas para apenas um servidor.

Se as *queries* se tratarem de execuções de funções não determinísticas, como random() e current_timestamp(), a execução destas pode ter diferentes valores em diferentes servidores. Isto porque cada servidor opera de forma independente. Tanto o *middleware* como a aplicação devem executar uma *query* a estes valores a partir de um único servidor e só então usá-los em *queries* de escrita de dados.

O **Pgpool-II** é um exemplo deste tipo de replicação.

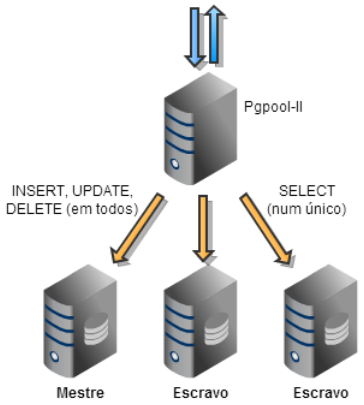

FIGURA 2.6: Funcionamento Pgpool-II

O Pgpool-II consegue balancear *queries* de leitura automaticamente e permite a execução de *queries* paralelas, podendo ser processadas em todos os servidores concorrentemente de forma a reduzir o tempo de execução final. Para além disso, fornece *Connection Pooling* e *Query Caching*. O *Connection Pooling* permite guardar as ligações aos servidores PostgreSQL, reutilizando-as sempre que uma ligação com as mesmas características seja necessária. O *Query Caching* permite que os resultados da execução de uma *query* fiquem em cache. Desta forma, quando houver um pedido de execução de uma *query* igual, os resultados guardados em cache serão devolvidos ao utilizador, aumentando a *performance* global do sistema.

Esta ferramenta pode ser aliada à *Streaming Replication*, evitando o problema das *queries* não determinísticas, já que as *queries* de alteração de dados são enviadas apenas ao servidor primário e depois propagadas por todos os servidores *standby*. A figura que se segue representa esse cenário:

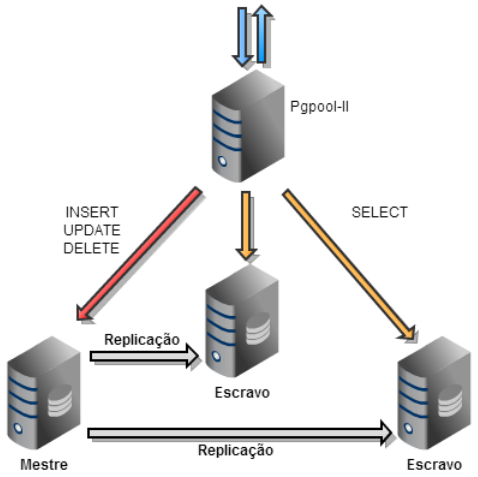

FIGURA 2.7: Pgpool-II com *Streaming Replication*

Replicação Multimestre Assíncrona

Este é o tipo de replicação indicado para computadores portáteis ou servidores remotos. Uma vez que não se encontram sempre ligados aos servidores de base de dados, manter os dados consistentes ao longo de todos os dispositivos é uma tarefa muito complicada.

Por exemplo, poderíamos ter utilizadores com uma cópia local da base de dados nos seus portáteis. Quando estivessem desligados dos servidores de base de dados poderiam efectuar alterações e mudanças na sua base de dados local e quando se voltassem a ligar aos servidores seria feito um *merge* para a base de dados com resolução de conflitos.

Com este tipo de replicação, cada servidor comunica periodicamente com os outros servidores para identificar conflitos. No caso de existirem são resolvidos ou pelo utilizador ou com base em regras de resolução de conflitos.

O **Bucardo** é um exemplo deste tipo de replicação.

O Bucardo é muito parecido com o Slony. A diferença fundamental reside na capacidade de, para além da replicação multi-escravo, permitir replicação entre servidores mestre com resolução de conflitos, através de regras de resolução de conflitos configuradas pelo utilizador.

Replicação Multimestre Síncrona

Trata-se de um tipo de replicação em que todos os servidores aceitam escritas de dados. Como tal, os dados alterados num servidor são transferidos primeiramente para todos os outros servidores presentes e só depois é realizado o *commit* das transacções envolvidas.

Por este método de replicação permitir a escrita de dados em vários servidores poderá haver uma degradação na *performance* do sistema. Por esta razão, algumas soluções adoptam a partilha de disco de forma a reduzir o *overhead* envolvido na escrita de dados. Apesar de o PostgreSQL não permitir este tipo de replicação, ela pode ser contornada através de, segundo a documentação oficial, um *commit* realizado em duas fases: *prepare transaction* e *commit prepared*.

No capítulo seguinte, de forma a validar a escolha da solução a usar, apresento uma tabela comparativa destas tecnologias de replicação.

2.5 Virtualização

Sabe-se que, em redes de computadores, as máquinas que alojam ficheiros e aplicações têm de ser potentes o suficiente para servir os seus clientes. Algumas máquinas têm múltiplos processadores que oferecem a estes servidores a capacidade de executar com facilidade tarefas complexas. É comum, em redes, os administradores dedicarem cada servidor a uma aplicação ou tarefa específica. Também é verdade que muitas destas aplicações não lidam bem com outras em simultâneo, isto é, cada uma precisa da sua própria máquina. Por outro lado, uma aplicação por servidor torna a gestão mais fácil e permite detectar problemas mais facilmente, caso apareçam. No entanto, esta abordagem levanta alguns problemas. Um deles é não tomar partido do grande poder de processamento que os servidores modernos disponibilizam. A maioria dos servidores usa apenas uma pequena parte das suas capacidades de processamento. Outro problema surge quando a rede de computadores fica maior e mais complexa. Neste caso, maior será o espaço ocupado pelos servidores num *datacenter* e, consequentemente, maior será o calor gerado por estes, obrigando a encargos redobrados com despesas de refrigeração.

A virtualização de servidores surge para resolver este tipo de problemas. Ao utilizar-se *software* desenhado especificamente para o efeito, um administrador consegue converter um servidor físico em múltiplas máquinas virtuais. Cada servidor virtual consegue actuar como se de um único servidor físico se tratasse, capaz de executar o seu próprio sistema operativo. A virtualização de servidores é um tema de investigação à décadas. No entanto, no mundo das tecnologias da informação, é vista como uma tecnologia em maturação com um grande caminho e potencial a percorrer.

Até há pouco tempo a única maneira de criar um servidor virtual era desenhando *software* especial, capaz de dividir o poder de processamento por múltiplas máquinas virtuais. Hoje, os fabricantes de processadores como a Intel e a AMD oferecem processadores com capacidade de suportar servidores virtuais. No entanto, não é o *hardware* que cria as máquinas virtuais. É necessário o *software* certo para as criar.

2.5.1 Vantagens da Virtualização

Há muitas razões que levam as organizações empresariais a investir na virtualização de servidores. Algumas dessas razões são financeiramente apelativas, enquanto que outras levantam algumas preocupações técnicas. Eis algumas das vantagens da utilização de virtualização:

- A virtualização de servidores conserva o espaço físico através da consolidação. É prática comum dedicar cada servidor a uma única aplicação. Se várias aplicações usarem apenas um pequeno poder de processamento, o administrador de redes consegue consolidar várias máquinas virtuais num único servidor físico. Para organizações que têm centenas ou milhares de servidores, a necessidade de espaço físico pode diminuir significativamente.

- A virtualização de servidores fornece uma forma de as organizações praticarem redundância sem a necessidade de comprar *hardware* adicional. A redundância refere-se a executar a mesma aplicação em múltiplos servidores. Trata-se de uma medida de segurança. Deste modo, se um servidor falhar por alguma razão, há outro servidor a executar a mesma aplicação que o pode substituir, minimizando qualquer interrupção de serviço. Não faz sentido criar dois servidores virtuais a executarem a mesma aplicação no mesmo servidor físico porque se este falhar, ambas as máquinas virtuais falharão. Na maioria dos casos, os administradores de redes criam servidores virtuais redundantes em máquinas físicas distintas.

- Os servidores virtuais oferecem isolamento, isto é, são sistemas independentes capazes de executar diferentes aplicações ou sistemas operativos diferentes. Em vez de comprar máquinas físicas dedicadas, um administrador de redes pode criar um servidor virtual numa máquina física existente.

- O *hardware* de um servidor pode eventualmente ficar obsoleto e trocá-lo pode ser um problema. De forma a manter a continuidade dos serviços fornecidos, um administrador de redes pode criar uma versão virtual do sistema e colocá-la num servidor moderno. Do ponto de vista aplicacional nada se altera. Isto pode dar tempo à organização para transitar entre máquinas sem se preocupar com eventuais falhas de *hardware*, particularmente se a organização que construiu o *hardware* que falhou já não existir e não puder reparar o equipamento.

- Uma tendência na virtualização de servidores é a chamada migração. Refere-se a mover o ambiente de um servidor de um sítio para outro. Com o *hardware* e *software* próprios é possível mover servidores virtuais de uma mesma máquina física para outra dentro da mesma rede. Originalmente, isto era possível se ambas as máquinas físicas tivessem o mesmo *hardware*, sistema operativo e processador. Hoje em dia, é possível migrar servidores virtuais de uma máquina para outra mesmo se as máquinas tiverem diferentes processadores. No entanto, é necessário que os processadores sejam feitos pelo mesmo fabricante.

- Se um servidor físico necessitar de manutenção, portar um servidor virtual entre máquinas físicas pode reduzir o tempo de indisponibilidade dos serviços.

2.5.2 Tipos de Virtualização

Há três formas de criar servidores virtuais: *full virtualization*, *para-virtualization* e virtualização ao nível do sistema operativo. Todas estas técnicas têm traços em comum. O servidor físico é chamado de *host* enquanto que os servidores virtuais são chamados de *guests*. Os servidores virtuais comportam-se como máquinas físicas. Cada uma destas técnicas utiliza uma abordagem diferente para alocar recursos físicos do servidor físico para os servidores virtuais.

- *Full Virtualization* - usa um *software* específico chamado *hypervisor*. Este interage directamente com o CPU e disco do servidor físico. Serve como plataforma para os sistemas operativos dos servidores virtuais que correm na mesma máquina física. Cada servidor virtual, *guest*, corre o seu próprio sistema operativo e podemos ter servidores virtuais que correm em Linux ou em outros sistemas operativos. O *hypervisor* monitoriza os recursos do servidor físico. Como os servidores virtuais

executam aplicações, o *hypervisor* faz a gestão dos recursos da máquina física para cada servidor virtual de forma apropriada. Ele próprio necessita de poder de processamento. Desta forma, esta técnica de virtualização pode afectar a *performance* global do servidor e diminuir a rapidez das aplicações em execução na máquina.

- *Para-virtualization* - Trata-se de uma abordagem um pouco diferente. Ao contrário da *full virtualization*, os servidores ou *guests*, têm consciência uns dos outros. Um *hypervisor* neste tipo de virtualização não necessita de tanto poder de processamento para gerir os sistemas operativos dos servidores virtuais, uma vez que cada sistema operativo está a par das necessidades dos outros sistemas operativos em execução nos outros servidores virtuais. Todo o sistema trabalha em conjunto como uma única unidade coesa.

- Virtualização ao nível do sistema operativo - neste tipo de virtualização não é necessário o uso de um *hypervisor*. Em vez disso, a capacidade de virtualizar faz parte do sistema operativo do *host*, servidor físico. Este, trata de todas as funções de um *hypervisor* virtualizado completo. A grande vantagem desta abordagem é que todos os servidores virtuais, *guests*, permanecem independentes em relação uns aos outros e é possível ter diferentes sistemas operativos. O KVM, Xen e VMWare são três exemplos conhecidos deste tipo de virtualização.

Escolher a técnica de virtualização mais adequada vai depender das necessidades do administrador de rede. Se os servidores físicos do administrador executarem todos no mesmo sistema operativo, então a virtualização ao nível do sistema operativo poderá ser a melhor. Esta abordagem tem tendência a ser a mais rápida e mais eficiente em comparação com os outros métodos. Por outro lado, se o administrador executar os servidores em diferentes sistemas operativos a para-virtualização poderá ser a melhor escolha. Uma potencial desvantagem deste tipo de virtualização é que esta técnica é relativamente recente e só algumas organizações oferecem este tipo de *software*. A maioria das organizações oferecem a *full virtualization*, apesar de o interesse na para-virtualização estar a crescer e possa vir a tomar o seu lugar.

2.5.3 Limitações da Virtualização

As vantagens da virtualização são imensas, mas não nos podemos esquecer das suas limitações. Dependendo do processamento necessário para executar uma determinada aplicação, a virtualização poderá não ser a melhor escolha, isto porque a virtualização essencialmente divide o poder de processamento de um servidor por todos os servidores virtuais. Quando o poder de processamento dos servidores não responde às necessidades das aplicações, tudo fica mais lento. Tarefas que não deveriam demorar muito tempo a executar, demoram imenso tempo. Pior: é possível que o sistema falhe se o servidor não tiver processamento suficiente para as necessidades das aplicações que nele estão a executar. Não é inteligente sobrecarregar o processador do servidor criando demasiados servidores virtuais numa máquina física. Quantas mais máquinas virtuais um servidor físico tenha de suportar, menor será o poder de processamento que cada servidor conseguirá ter. Em adição, existe um limite de espaço em disco. Muitos servidores virtuais podem causar impacto na capacidade de o servidor armazenar dados.

2.5.4 Limitações da Virtualização - Caso Wavecom

No caso concreto da Wavecom não será utilizada virtualização. A virtualização revela-se uma má prática tendo em conta os seguintes aspectos:

- *Timers*: tratando-se de uma solução de telefonia IP com requisitos de tempo-real, este factor revela-se importante. Com a virtualização do CPU algumas máquinas virtuais, passado algum tempo, começam a deteriorar-se devido à dessincronização do seu relógio com o do servidor físico, *host*, levando a um decréscimo na qualidade de experiência sentida. No caso concreto do sistema de telefonia, notar-se-iam diferenças na qualidade e sincronia entre a voz recebida e a esperada. Mesmo em máquinas virtualizadas, com técnicas de virtualização só ao nível do sistema operativo, os ciclos de relógio são diferentes porque é o *hypervisor* que aloca processamento para estas máquinas virtuais. Ou seja, num eventual cenário com máquinas virtuais poderá dar-se o caso de as máquinas não responderem bem em determinado momento, sacrificando o FreeSWITCH e, por sua vez, a comunicação em tempo-real.

- Base de Dados: a virtualização não pressupõe a sincronização dos dados da base de dados entre máquinas virtuais. Normalmente fazem-se cópias de *backup* de máquinas virtuais diariamente ou seguindo outras políticas. Desta forma, quando uma máquina falhar, há uma cópia de *backup* que permite, com um delta de perda de dados associado, continuar com os serviços. Em determinados casos o que acontece é ter-se armazenamento partilhado, levando as máquinas em questão a irem consultar os dados a um dispositivo de armazenamento remoto. Neste caso, seria necessário garantir que este dispositivo por si só não constituísse um ponto único de falha. Este nível de redundância levaria-nos a despesas adicionais. Desta forma, a virtualização não é útil, uma vez que é necessário haver engenharia ao nível dos serviços capaz de tratar do processo de sincronização das bases de dados entre os dois servidores em tempo-real.

- Outra questão pertinente surge: se quisermos ter a nossa solução de alta disponibilidade na *cloud* não podemos prever saltos na virtualização entre máquinas. Isto porque numa *cloud*, saltar entre máquinas virtuais é complicado se tivermos necessidade de usar *hardware* adicional, como por exemplo placas PRI ou BRI. Para além de este *hardware* ter de ser suportado pela *cloud*, no caso de uma placa destas avariar seria necessário haver mecanismos de *failover* destas mesmas placas, o que não é nada comum em regime de *cloud*.

Para concluir, os desafios que dizem respeito à construção da solução de alta disponibilidade em questão são os mesmos quer optasse por utilizar máquinas virtuais ou reais, sendo possível, inclusivamente, suportar ambos os cenários, com as eventuais limitações que o facto de ter máquinas virtuais iria trazer.

Capítulo 3

Objectivos da Investigação e Método de Abordagem

Este capítulo centra-se na metodologia a adoptar e de que maneira os objectivos da solução serão levados a cabo. Numa primeira fase é feita uma escolha justificada das ferramentas a utilizar e, em seguida, são apresentadas as arquitecturas a montar. Por fim, são definidos os requisitos da solução e as métricas a utilizar na validação da solução obtida.

3.1 Critérios de selecção/exclusão de tecnologias

Ferramentas escolhidas para *Failover*

De forma a tornar possível a implementação da solução de alta disponibilidade foi necessário seleccionar de entre várias, as ferramentas que melhor se adequam e cumprem os requisitos de qualidade necessários à solução.

Ao nível do *cluster* serão adoptadas as ferramentas Corosync e o Pacemaker. O Corosync será responsável pela camada de comunicação entre os nós enquanto que o Pacemaker será o responsável pela paragem, arranque e gestão dos serviços e aplicações. Existem mais ferramentas que suportam estas características, no entanto, não fazem parte da documentação oficial do projecto vocacionado para a construção deste tipo de soluções em ambientes Linux (Linux-HA) nem sequer detêm metade da documentação pelo que, as excluí à partida.

No entanto, foi possível estudar o funcionamento do LVS como solução de alta disponibilidade. Como resultado, assumi que o facto de pegar nas ferramentas em particular, como o Corosync e o Pacemaker, me permitirá ter um maior controlo sobre a implementação e configuração do sistema de alta disponibilidade, não ficado restringido a um tipo de arquitectura pré-concebido com balanceamento de carga, como é o caso do LVS.

O Keepalived poderia ser uma boa ferramenta a utilizar. No entanto, a quantidade de informação e documentação do Corosync e do Pacemaker como solução de alta disponibilidade é francamente maior. Para além disso, o Corosync e o Pacemaker fazem parte de uma das soluções válidas a implementar e são referidas na documentação oficial do FreeSWITCH, ao contrário do Keepalived [17].

Ferramenta de telefonia IP

Quanto à ferramenta de telefonia IP a usar, esta foi previamente escolhida pela empresa e será o FreeSWITCH. Contudo, foi possível fundamentar esta escolha através de um estudo comparativo que contrapõe a ferramenta FreeSWITCH à sua maior concorrente, que detém mais projectos e mais utilizadores, amplamente usada no mercado e com uma comunidade de desenvolvimento maior, o Asterisk. Essa comparação é visível no Anexo A - Soluções de Telefonia IP.

Ferramenta de balanceamento de carga

Para fazer o balanceamento de carga dos pedidos SIP e HTTP dos clientes que se ligam à solução de alta disponibilidade, optou-se pelo DNS. Apesar de o balanceamento de carga ser feito com base em probabilidades sem ter em conta a carga actual dos servidores, não deixa de ser uma opção válida para a solução já que, ao contrário de outras soluções, acaba por ser mais fácil de configurar, sendo a solução mais rentável. Se mais tarde, ao analisar o tráfego, se reparar que há uma máquina que está a ser mais sobrecarregada que outra, poder-se-á encaminhar-lhe menos tráfego ou até, dependendo do caso, optar por uma outra solução de balanceamento de carga. Trata-se de uma solução bastante popular e adoptada em muitos sistemas, cumprindo nesta fase, os requisitos de qualidade da solução a implementar.

Ferramenta de Base de Dados

Quanto à base de dados, coube também à empresa o poder de decisão da ferramenta a utilizar. Desta forma, a solução escolhida foi o PostgreSQL. No entanto, uma vez mais, de forma a consubstanciar esta escolha, foi possível fazer um apanhado das características

que distinguem algumas das ferramentas de base de dados relacionais *OpenSource* mais utilizadas e chegar a uma conclusão, tal como pode ser visto no Anexo D - Soluções de Base de Dados.

Ferramenta de Replicação de Base de Dados

No que respeita à replicação de base de dados, existem algumas ferramentas para o PostgreSQL. No entanto, algumas delas foram abandonadas e nunca atingiram um grau de maturidade equiparável àquelas que constam da súmula de soluções estudadas e que se apresentam nas duas tabelas abaixo. Tendo em conta a comparação realizada, suportada pela documentação oficial do PostgreSQL e tendo em conta os requisitos de qualidade da solução de alta disponibilidade a implementar, considera-se que a ferramenta Pgpool-II é solução que melhores características oferece.

A tabela que se segue pretende resumir as funcionalidades-chave das ferramentas descritas para cada um dos métodos de replicação apresentados no capítulo 2 - Estado da Arte - Metodologias e Ferramentas, que correspondem às implementações mais comuns para a configuração de sistemas de alta disponibilidade.

Característica	Shared Disk Failover	Replicação do Sistema de Ficheiros	Transaction Log Shipping	Replicação Mestre-Escravo TriggerBased	Replicação através de Middleware	Replicação Multimestre Assíncrona	Replicação Multimestre Síncrona
Implementação mais comum	NAS	DRBD	Streaming Replication	Slony	Pgpool-II	Bucardo	N/A
Sem Hardware Adicional		✓	✓	✓	✓	✓	✓
Múltiplos Mestre					✓	✓	✓
Sem Overhead Mestre	✓		✓		✓		
Sem necessidade de espera por múltiplos servidores	✓		modo Sync OFF	✓		✓	
Falha do Servidor Primário não implica perda de dados	✓	✓	modo Sync ON		✓		✓
O Servidor Standby aceita queries de leitura			modo Hot Standby	✓	✓	✓	✓
Granularidade por Tabela				✓		✓	✓
Não necessita de resolução de conflitos	✓	✓	✓	✓			✓

TABELA 3.1: Métodos de Replicação do PostgreSQL - Adaptado de: http://www.postgresql.org/docs/9.2/static/different-replication-solutions.html

A partir desta tabela é possível justificar a escolha do Pgpool-II como ferramenta de replicação de base de dados. Senão, vejamos:

- O NAS é um dispositivo de armazenamento central que permite o acesso a dados numa rede através de protocolos de acesso remoto. No entanto, a sua implementação implica gasto na sua aquisição e, para além disso, constitui um SPOF, inviabilizando a sua inclusão. Por outro lado, aceder a uma base de dados através de um NAS implicaria de imediato um *overhead* no acesso aos dados devido aos protocolos de acesso remoto e aos eventos I/O a que operações de acesso a disco obrigam. Para além disto, não permite o acesso concorrente de dois servidores ao mesmo dispositivo NAS, sob pena de ambos alterarem o mesmo segmento de dados.

- O DRBD é um sistema concebido para a alta disponibilidade, permitindo replicar sistemas de ficheiros e blocos de disco. No entanto, não foi desenhado para sistemas com múltiplos servidores mestre. Desta forma, apenas um deles pode montar a partição correspondente ao sistema de ficheiros de cada vez, não permitindo ao servidor *standby* executar *queries*.

- A *Streaming Replication* é um método de replicação aceitável que permite replicação síncrona entre servidores de base de dados. No entanto, serve unicamente esse propósito, a replicação, não permitindo ter múltiplos servidores primários já que a replicação é feita num único servidor mestre a partir do qual os dados são depois replicados para os outros servidores. Os servidores *standby* podem executar *queries*.

- O Slony é um método de replicação baseado em *triggers*, o que por si só adiciona *overhead* ao servidor mestre. É uma solução assíncrona e por isso não serve os requisitos da solução a implementar, como veremos mais à frente. No entanto, é uma solução interessante para *clusters* grandes em que queremos ter replicação em cascata.

- O Bucardo é em tudo igual ao Slony, à excepção de que permite ter uma configuração com vários servidores mestre no mesmo *cluster*, oferecendo um mecanismo de resolução de conflitos aquando a replicação assíncrona.

O Pgpool-II pode ser combinado com a *Streaming Replication* podendo, não só replicar os dados sincronamente, como também fornecer funcionalidades bastante vantajosas ao nível da implementação de uma solução de *cluster*, como é visível na tabela seguinte.

Ferramenta	**Método Replicação**	**Con. Pooling**	**Bal. Carga**	**Frag. Queries**
Bucardo	Assíncrona	×	×	×
Pgpool-II	Síncrona	✓	✓	✓
Rubyrep	Assíncrona	×	×	×
Londiste	Assíncrona	×	×	×
Slony	Assíncrona	×	×	×
Mammoth	Assíncrona	×	×	×
Postgres-XC	Síncrona	×	✓	×
Postgres-R	Assíncrona	×	×	×

Tabela 3.2: Ferramentas de Replicação do PostgreSQL

Esta tabela pretende focar nas ferramentas de replicação existentes e, de uma forma mais abrangente, explicitar o porquê da escolha do Pgpool-II como ferramenta de replicação de base de dados.

Uma vez mais, é possível observar que a ferramenta que melhor serve os requisitos da solução é o Pgpool-II já que, para além das características verificadas na tabela anterior, oferece *Pooling* de ligações, balanceamento de carga e Fragmentação de *Queries*, apresentando-se como a ferramenta mais completa.

Para além disso existe outra ferramenta que é uma extensão ao Pgpool-II chamada Pgpool-HA. É uma ferramenta que oferece um bom nível de redundância permitindo através da detecção de falha de um Pgpool-II, utilizar e promover uma réplica do Pgpool-II a nó principal, garantindo a sua disponibilidade.

No Anexo B - Soluções de Replicação de Dados poderão ser vistas, de forma mais detalhada, as características inerentes a cada uma destas ferramentas.

3.2 Arquitectura

De forma a implementar uma solução de alta disponibilidade que garanta que, na ocorrência de uma falha, o sistema de telefonia IP possa, de uma forma transparente para o cliente, continuar a executar os seus serviços e a responder de forma eficaz aos seus pedidos, serão tidas em conta duas implementações: Activo/Activo e Activo/Passivo.

3.2.1 Arquitectura Activo/Activo

Esta configuração admite duas máquinas activas a responder a pedidos dos clientes. Ambas detêm os mesmos tipos de serviços (serviço WEB FsCloud, serviço IPBX FreeSWITCH, serviço de base de dados PostgreSQL e a ferramenta de replicação Pgpool-II). No caso de uma falhar, todo o tráfego que lhe era destinado é entregue à máquina que se mantém em pleno estado de funcionamento. Na figura abaixo apresenta-se a arquitectura desta solução:

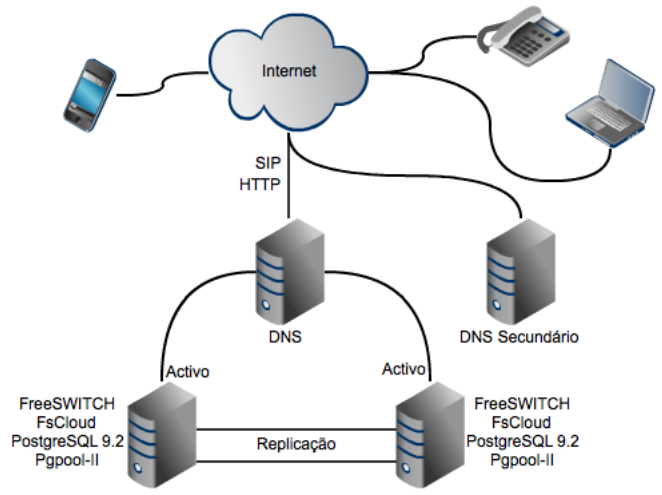

FIGURA 3.1: Arquitectura Activo/Activo

Para este cenário, foi tida em consideração a necessidade de balanceamento de carga dos pedidos SIP e HTTP que chegassem aos servidores. Pelos motivos já referidos, decidiu-se usar o DNS. De forma a evitar pontos únicos de falha, existirá um DNS Secundário que resolverá os pedidos dos clientes no caso de o servidor primário falhar. Por sua vez, os serviços de telefonia IP, WEB e base de dados estarão replicados em ambas as máquinas.

Desta forma, na indisponibilidade de um serviço ou numa falha de um servidor, o DNS reencaminhará os pedidos para o outro, que continuará activo e responderá aos pedidos.

É essencial que haja replicação da base de dados entre as máquinas de forma a que os dados, quando retornados por um servidor ou por outro, sejam consistentes.

3.2.2 Arquitectura Activo/Passivo

Esta configuração admite a existência de uma máquina que está em modo activo (*online*) e outra em modo passivo (*offline*). A máquina em modo passivo é uma máquina redundante que ficará activa quando a primeira, ou os serviços que nela estão a ser executados, falharem. Na ocorrência de uma falha no nó activo, o nó passivo assume todos os serviços e aplicações que estavam a ser executados de forma transparente para o utilizador. Na figura abaixo pode ver-se a arquitectura deste cenário:

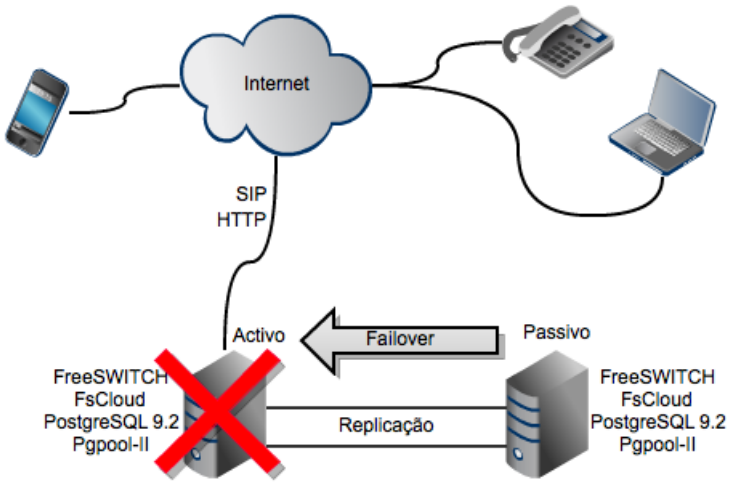

FIGURA 3.2: Arquitectura Activo/Passivo

Uma vez mais, é fundamental que haja replicação da base de dados entre as máquinas para que, em caso de falha do nó activo, a máquina em modo passivo possa arrancar com a base de dados actualizada.

A razão da utilização de dois cenários de teste vai além das necessidades da empresa. A empresa necessita de uma solução Activo/Activo de forma a garantir a redundância de todos os serviços envolvidos no sistema de telefonia IP dos seus clientes. Esta implementação prende-se com a possibilidade de se ter não só redundância, como também

balanceamento de carga já que, tendo duas máquinas activas, seria um desperdício de recursos ter apenas uma máquina a servir os clientes (FreeSWITCH) e outra só com a base de dados, por exemplo. Isto permite-nos antever o seguinte: este cenário é teoricamente mais favorável do que o cenário Activo/Passivo e cumpre por completo as necessidades da empresa. No entanto, de forma a consubstanciar esta teoria, serve o cenário Activo/-Passivo para provar isso mesmo: que o facto de se ter duas máquinas ligadas, em que uma tem um papel passivo, não só permite um menor número de chamadas no sistema, como também representa um custo acrescido mal aproveitado da segunda máquina, que apenas existe para oferecer redundância de serviços no caso de a primeira máquina falhar.

3.3 Requisitos Funcionais, Não-Funcionais e Métricas

Como seria de prever, a plataforma de telefonia IP tem bastantes requisitos que se apresentam na sua totalidade em anexo. Por agora, de forma breve, é possível citar os fundamentais:

Requisitos Funcionais

- O sistema deverá permitir o cliente efectuar chamadas através da Internet quer via *softphones* ou *hardphones* IP;
- O sistema deverá permitir o cliente gerir os dados da sua plataforma de telefonia IP via WEB.

Requisitos Não-Funcionais

- O sistema deverá garantir a integridade e consistência através da replicação da base de dados entre os dois servidores. Desta forma, quer um cliente seja tratado por um servidor ou por outro, verá os mesmos resultados;
- No caso de uma falha, o sistema deverá recuperar e disponibilizar todos os serviços da máquina que falhou (serviço WEB FsCloud, base de dados e FreeSWITCH);
- O sistema deverá escalar. O número máximo de chamadas em simultâneo a definir não deverá implicar a degradação no desempenho das chamadas em curso nem dos servidores;

- Segurança/Confidencialidade: os servidores deverão usar encriptação nas suas ligações. A sessão SIP dos clientes deverá ser encriptada com TLS, enquanto que a *media* RTP deverá ser encriptada via SRTP.

Para uma consulta mais informada de todos os requisitos inerentes à plataforma de gestão do sistema de telefonia IP, deverá ser consultado o anexo E, Requisitos Funcionais e Não-Funcionais.

De forma a avaliar o desempenho dos cenários de alta disponibilidade a implementar e de forma a garantir a solução que oferece melhores resultados, serão avaliadas as seguintes métricas:

- Número de chamadas em simultâneo;
- Consumo de recursos do processador, medido em percentagem (%);
- Consumo de recursos da memória RAM, medido em percentagem (%).

Estas métricas são fundamentais e permitirão, de forma objectiva e credível, fundamentar a escolha de uma solução em detrimento de outra.

De forma a consubstanciar os resultados obtidos pelas métricas descritas acima, serão avaliados os seguintes parâmetros de rede:

- Jitter, em milissegundos (ms);
- Atraso, em milissegundos (ms);
- Débito, em Mbps (Mbps);
- Taxa de perda de pacotes, em percentagem (%);
- MOS - *Mean Opinion Score*, (0-5).

Estas são métricas *standard* para medir a qualidade de áudio num sistema de telefonia IP. Esta escolha é ainda suportada e consta em alguns dos artigos referidos ao longo do relatório.

Capítulo 4

Planeamento e Realização de Testes

Este capítulo pretende mostrar o estado de maturidade em que o trabalho se encontra e de que forma os resultados nele apresentados sustentam e validam esta tese. Nele, são também apresentadas as estratégias tomadas e os procedimentos tidos na concepção e realização dos testes. Para além disso, são tecidas considerações relativamente aos resultados obtidos e é também feita uma crítica, com o intuito de dar a perceber ao leitor de que forma os resultados apresentados são válidos e, no contexto do trabalho, os poderíamos prever.

4.1 Descrição dos cenários de teste

De forma a levar a cabo a realização de testes aos cenários de alta disponibilidade implementados (Activo/Passivo e Activo/Activo), tornou-se vital pensar nos recursos das máquinas e na distribuição dos serviços pelas mesmas. Como tal, ficou decidido que, de modo a tirar o maior proveito dos recursos de processamento da máquina, todos os serviços deveriam estar concentrados num mesmo servidor (serviço WEB, telefonia IP, base de dados e replicação de base de dados), enquanto que deveria existir um terceiro servidor, cliente, responsável por originar o tráfego SIP e que, no cenário Activo/Activo, serviria também de DNS para o balanceamento das chamadas entre os dois servidores. É também neste servidor cliente que se localizam instalados e configurados os *softwares*

Nagios e MRTG, no sentido de monitorizar o estado dos servidores em produção via SNMP. Esta decisão foi tomada de forma a que todo o processamento e memória RAM necessária à execução das ferramentas de monitorização e geração de tráfego SIP não interferisse com os servidores em produção, o que poderia levar a uma sobrecarga adicional nestes e ainda a uma desvirtuação dos resultados finais obtidos.

Com base na arquitectura definida e, de acordo com as ferramentas escolhidas ao longo do capítulo anterior, foram tidas em consideração para a implementação das soluções, as seguintes especificações para cada servidor:

- Sistema Operativo CentOS 6.4, por se tratar de um sistema operativo livre e muito estável, sobre o qual as actualizações realizadas têm lugar em períodos de tempo relativamente grandes, poupando o utilizador a constantes actualizações necessárias, por vezes, ao funcionamento de certas funcionalidades;
- Plataforma WEB FsCloud;
- PostgreSQL 9.1 como base de dados;
- Pgpool-II 3.1.3 para a replicação da base de dados.

Adicionalmente, no servidor-cliente, foram instalados os *softwares* de monitorização Nagios e MRTG (*Multi Router Traffic Grapher*) que permitiram uma monitorização passiva da carga dos servidores via SNMP, nomeadamente processador, memória RAM e tráfego gerado. Este servidor foi útil também para a análise de alguns parâmetros de rede através do *software* VoIPMonitor que permitiu, através do conhecido *sniffer* de rede Wireshark, com base em capturas de tráfego efectuadas, analisar a qualidade de áudio dos sistemas sob teste. Por último, foi instalado o serviço DNS para balanceamento de tráfego no cenário Activo/Activo. Por fim, será utilizada outra máquina que servirá de cliente e será responsável por espoletar chamadas com recurso ao *software* SIPp e também com recurso a *scripts* na linguagem de programação Lua.

Por limitação de recursos, todos os servidores foram criados através do *software* VMWare ESXi, responsável pela criação de máquinas virtuais, num servidor IBM x3550 com quatro CPU *Cores* a 2,493GHz. O processador utilizado foi um Intel Xeon E5420 de arquitectura x64. Este servidor foi colocado numa rede partilhada a 100Mbps e os recursos foram divididos da seguinte forma:

Cenário Activo/Passivo:

- Servidor 1: 1 *core*, 2Gb RAM;
- Servidor 2: 1 *core*, 2Gb RAM;
- Servidor Cliente: 2 *cores*, 4Gb RAM;

Cenário Activo/Activo:

- Servidor 1: 1 *core*, 2Gb RAM;
- Servidor 2: 1 *core*, 2Gb RAM;
- Servidor Cliente: 4 *cores*, 4Gb RAM;

A razão de se ter um servidor-cliente mais potente do que os servidores-alvo tem a ver com o facto de que a máquina geradora tem uma carga de processamento relativamente elevada. Em alguns testes iniciais foi possível verificar que este servidor não aguentava o processamento que a geração de tráfego SIP exigia para duas máquinas em simultâneo. Por esta razão, houve necessidade de alocar mais recursos para este servidor, passando de 2 *cores* para 4 no cenário Activo/Activo.

Ao contrário do que inicialmente foi proposto, a utilização do SIPp como ferramenta de geração de tráfego SIP verificou-se inviável, uma vez que as sessões SIP inicializadas neste programa não eram registadas na máquina à qual eram destinadas, não simulando, por isso, um cenário real. Por esta razão foram desenvolvidos alguns *scripts* na linguagem LUA que, através da API do FreeSWITCH, nos permitem gerar quantas chamadas SIP quisermos e com a duração que pretendermos.

De forma a realizar os testes foram tidas em consideração as seguintes especificações:

	Cenário Activo/Passivo
Chamadas em simultâneo	20, 40, 60, 80, 100, 120, 140
Número de *runs*	15
Duração das *runs*	10 minutos
Repetições numa *run*	4
Tempo das chamadas	150 segundos (2,5 minutos)

	Cenário Activo/Activo
Chamadas em simultâneo	40, 80, 120, 160, 200, 240
Número de *runs*	15
Duração das *runs*	10 minutos
Repetições numa *run*	4
Tempo das chamadas	150 segundos (2,5 minutos)

TABELA 4.1: Especificações dos Cenários de Teste

O principal objectivo deste projecto incidia na implementação e configuração dos cenários descritos. Depois disso, tornou-se fundamental realizar alguns testes de modo a perceber como é que o sistema respondia. Ficou acordado que a melhor forma de perceber o comportamento do sistema seria sujeitá-lo a testes de *stress*. Como tal, foi necessário perceber quais seriam os limites do sistema e a partir de quando é que já não era viável sobrecarregá-lo mais, sob pena de degradar o áudio das chamadas em curso no sistema. A decisão do número e chamadas em simultâneo foi feita tendo em conta as características dos servidores, neste caso, das máquinas virtuais. Decidiu-se, então, fasear os testes seguindo uma progressão aritmética. No caso do cenário Activo/Passivo a razão dessa progressão é de 20 chamadas, enquanto que no cenário Activo/Activo (por termos o dobro dos recursos) é de 40. No que respeita ao número de *runs*, tanto num cenário como no outro foram realizadas 15 para cada fase de testes, isto é, 15 *runs* para 20 chamadas em simultâneo, 15 *runs* para 40 e assim sucessivamente. Durante esse intervalo de tempo foram realizadas 4 repetições de chamadas de 150 segundos cada. Estes valores não foram escolhidos ao acaso. A razão de ter 15 *runs* prende-se com o facto de serem em número suficiente para a obtenção de resultados com um desvio padrão associado relativamente baixo, o que me levou a concluir que os resultados convergem para um valor estável. Quanto ao tempo das chamadas foi escolhido 150 segundos, de acordo com o relatório obtido no 1º trimestre de 2013, segundo a ANACOM [18].

4.1.1 Cenário Activo/Passivo

Este cenário foi criado no sentido de perceber de que forma é que o sistema lidava com o facto de se ter apenas uma máquina activa a responder a pedidos dos clientes. Foi criado também como termo de comparação com o cenário Activo/Activo e por forma a saber até que ponto seria vantajoso ter um sistema destes numa solução de telefonia IP. Para a realização deste cenário foi utilizado o Corosync como camada de comunicação entre os nós, com o fim de detectar falhas na resposta do servidor activo e também o Pacemaker, de forma a gerir os serviços.

Como resultado deste cenário apresentam-se os gráficos que ditam o comportamento do sistema ao longo da fase de testes.

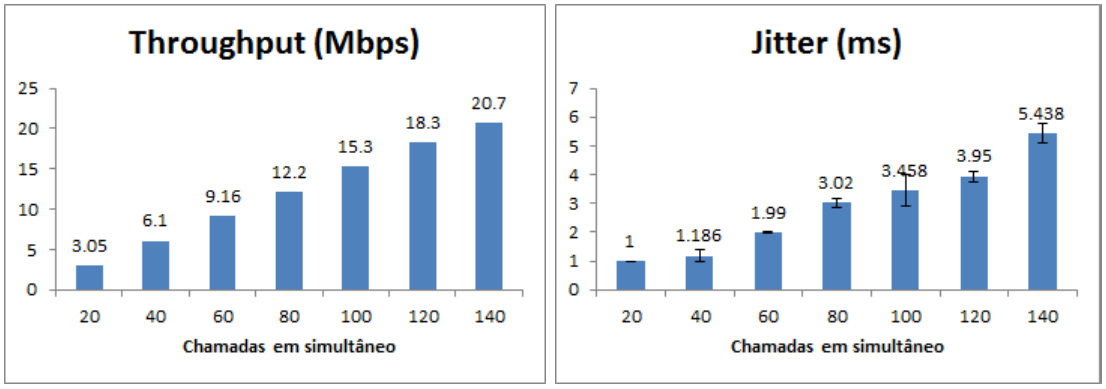

FIGURA 4.1: Throughput e Jitter

É possível observar que o valor do débito é proporcional ao número de chamadas no sistema à medida que o número de chamadas aumenta. Para além disso, não se registou qualquer desvio padrão o que significa que não houve perdas de chamadas e que os fluxos de dados tiveram um comportamento natural no sistema. Quanto ao Jitter o caso muda um pouco de figura. Apesar de o débito apresentar valores normais não significa que a rede em que o cenário foi montado seja, isto é, nota-se alguma variação de atraso, apesar de não ser muito significativo (estamos a falar de milissegundos). Através do desvio padrão associado é também perceptível que os valores não oscilam muito e que estes valores são fidedignos.

Quanto à taxa de perda de pacotes é-nos possível constatar que os valores apresentados são muito pouco significativos, tendo o seu pico máximo aquando as 140 chamadas em simultâneo no sistema. É no atraso que percebemos de imediato a nossa condição de

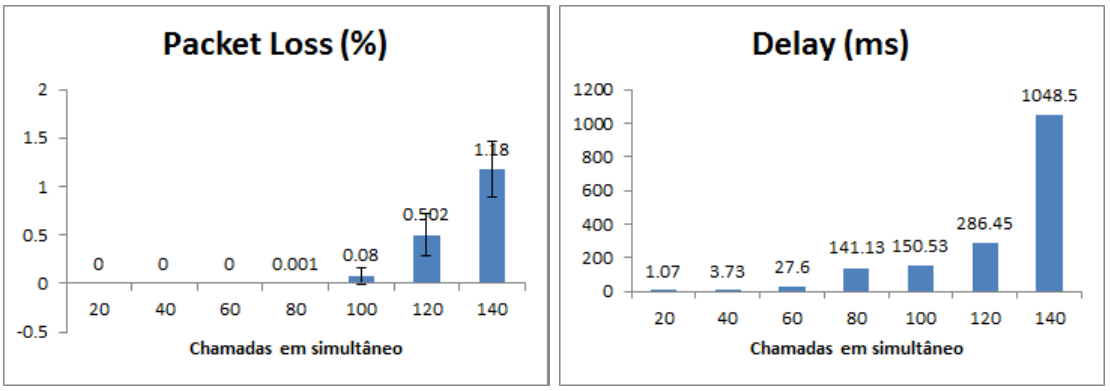

FIGURA 4.2: Packet Loss e Delay

paragem nos testes. Verificamos que ao longo das chamadas em simultâneo efectuadas os atrasos se vão conseguindo comportar e a qualidade de áudio é suficientemente boa para se ter uma conversação *end-to-end* aceitável. Tudo muda de figura quando atingimos as 140 chamadas simultâneas. Aqui o atraso medido dispara e, segundo as normas da ITU, um valor de atraso superior a 400ms torna inaceitável uma conversação normal [19]. Contudo, não esqueçamos que estamos numa rede partilhada e, portanto, os servidores sob teste não são agnósticos ao tráfego e eventual congestão que haja na rede. Esta decisão foi tomada, como disse no início do capítulo, de forma a simular um cenário real, em que os nossos clientes podem estar atrás de uma rede partilhada, com algum nível de congestão.

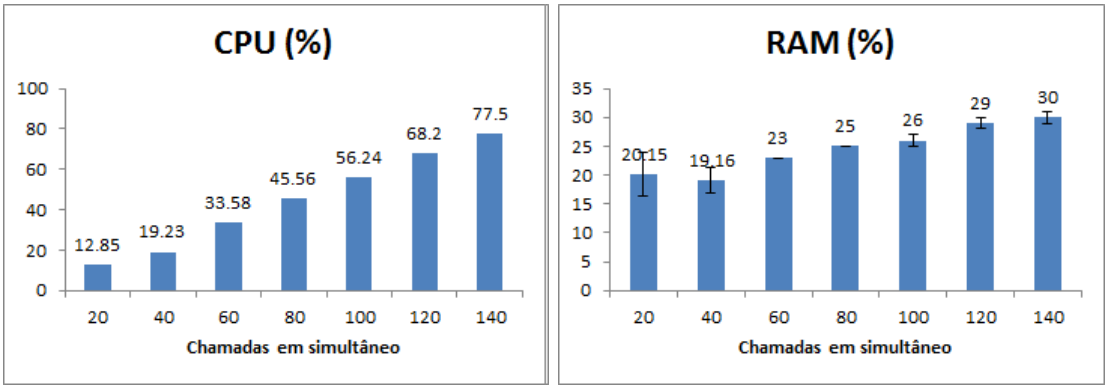

FIGURA 4.3: Consumo dos recursos do processador e memória RAM

Ao analisar os valores obtidos para o CPU e para a RAM percebemos que, apesar de as 140 chamadas em simultâneo roçarem os 80%, sentimos que ainda podíamos chegar um pouco mais longe. No entanto, tendo em consideração os valores da rede, assumi que o ideal será ter no máximo, para estas características, 120 chamadas em simultâneo.

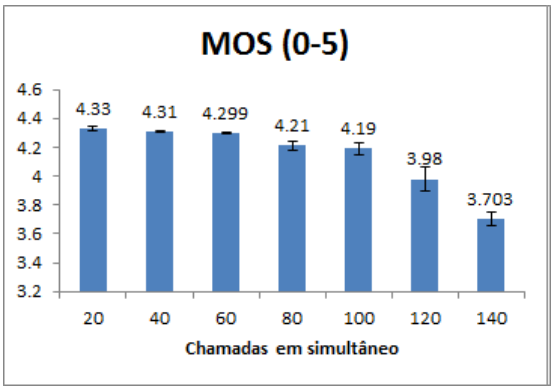

FIGURA 4.4: MOS (0-5)

À luz dos resultados obtidos nas métricas anteriores, os valores do MOS são perfeitamente aceitáveis. Considera-se que o nível 4 de MOS é o valor mínimo aceitável para uma conversação plena, não-disruptiva.

4.1.2 Cenário Activo/Activo

O cenário Activo/Activo era a grande necessidade da empresa. Este cenário foi construído de forma a tirar proveito dos recursos de dois servidores e usá-los para responder a pedidos SIP e HTTP. Neste caso, as duas instâncias do FreeSWITCH, uma em cada servidor, comportam-se como se de uma só se tratasse e toda a arquitectura é transparente para o utilizador. Desta forma, um mesmo domínio coexiste em dois servidores. Todos os dados associados a extensões, *gateways*, etc, devem ser partilhados. Essa partilha poderia ser feita de duas formas: a primeira seria através de uma base de dados partilhada, enquanto que a segunda seria ter duas instâncias da base de dados actualizada. De forma a evitar pontos únicos de falha, optámos pela segunda. Isto foi conseguido à custa do Pgpool que detém a capacidade de replicar sincronamente os dados entre duas bases de dados, nos dois sentidos (*master-master*). Por outro lado, era fundamental que, no caso de falha de um servidor, o outro pudesse assumir as suas funções de forma independente. No caso da reposição de um servidor que tivesse falhado, deveria ser possível consegui-lo arrancar sem que para isso fosse necessário desligar o servidor activo ou interromper os seus serviços. Sobre esta questão falarei no final deste capítulo, secção Online Recovery.

Uma vez mais, agora para este cenário, apresentam-se os resultados obtidos durante a fase de testes.

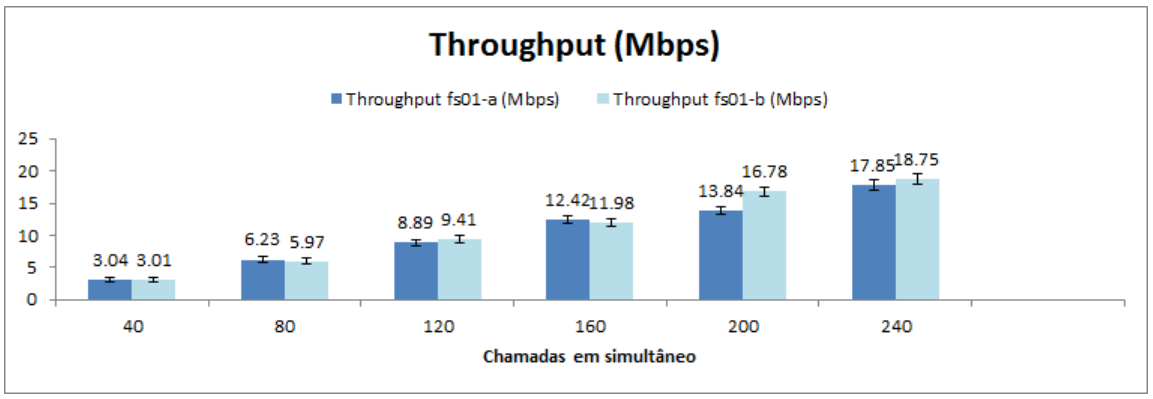

FIGURA 4.5: Throughput

Ao analisar este gráfico conseguimos perceber que, ao longo das chamadas em simultâneo realizadas, o débito apresenta uma distribuição equitativa uniforme. Quer isto dizer que o DNS foi capaz de balancear quase sempre o mesmo volume de chamadas pelos dois servidores. Se somarmos o débito dos dois servidores e o compararmos, em cada fase, com o débito obtido no cenário Activo/Passivo, reparamos que é aproximadamente sempre o dobro, o que vai de encontro às expectativas, uma vez que estamos a gerar o dobro das chamadas em cada iteração. Este gráfico mostra ainda, a concluir pelo débito gerado, que não houve perda de chamadas. No entanto, claro está, poderá haver degradação das mesmas. O desvio padrão é muito pouco notório o que favorece e credibiliza estes resultados.

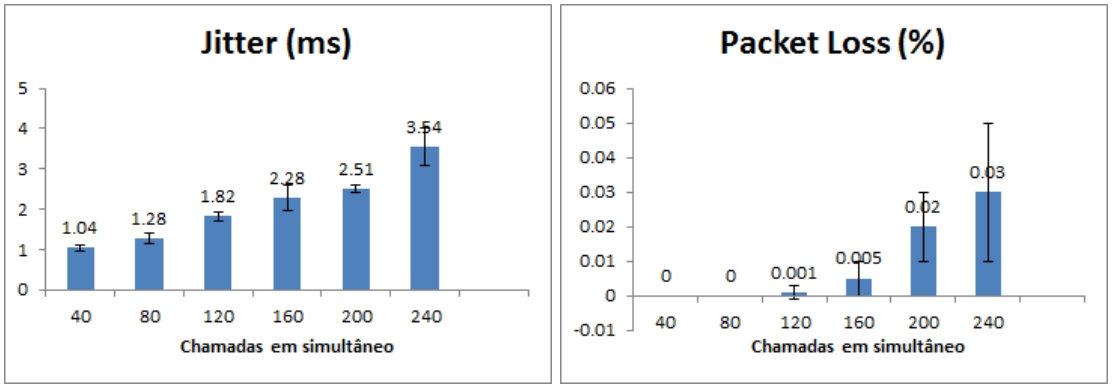

FIGURA 4.6: Jitter e Packet Loss

Analisando o Jitter e a taxa de perda de pacotes, à semelhança com o ocorrido no cenário anterior, verifica-se que os valores são muito pouco acentuados, o que significa que, não

são valores alarmantes ou que nos permitam inferir que a qualidade das chamadas está comprometida. Em primeiro lugar porque a variação de atraso não apresenta valores assim tão grandes e, em segundo, porque a taxa de perda de pacotes é praticamente nula.

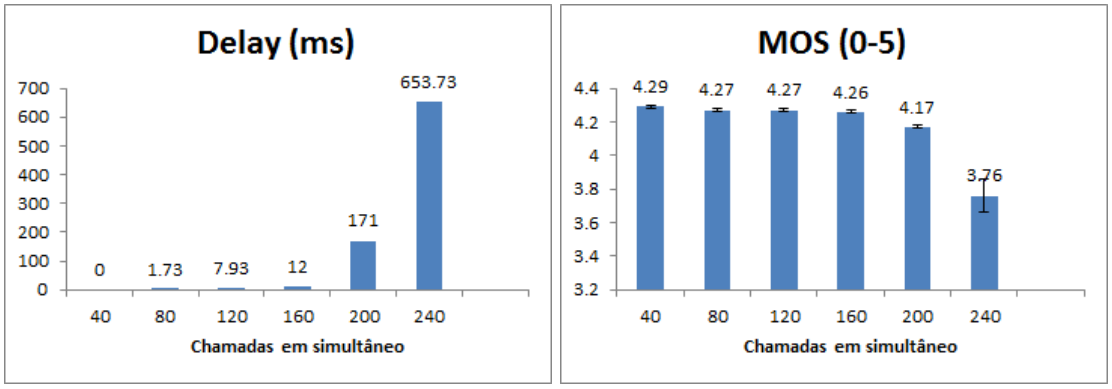

FIGURA 4.7: Delay e MOS

Quanto ao atraso, uma vez mais é visível que ao longo dos testes ele não se manifesta muito, levando-nos a crer que as chamadas se realizam com relativa fluidez e qualidade excepto quando atingimos as 240 chamadas simultâneas. Aqui é sentido um atraso médio superior a 650ms, inviabilizando, segundo as normas da ITU, uma conversação razoável. Por esta razão não é possível garantir que tenhamos 240 chamadas em simultâneo. Os valores do MOS também são conclusivos e adequam-se perfeitamente aos resultados obtidos.

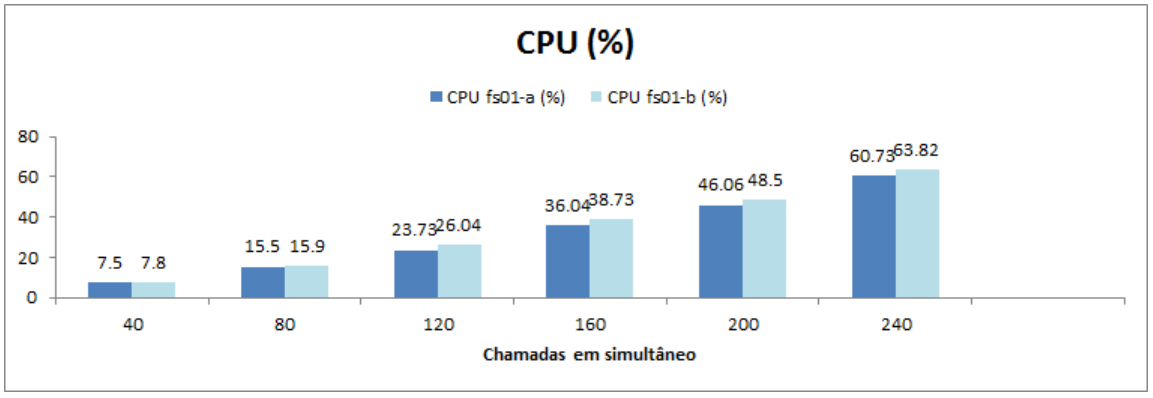

FIGURA 4.8: Consumo dos recursos do processador

Foi possível verificar que os dados do CPU corroboram os resultados obtidos no débito, ou seja, conseguimos verificar que a carga é igualmente distribuída pelos dois servidores. No entanto, uma vez mais, é a rede que limita os resultados obtidos neste cenário, já

que os 63,82% de CPU máximo verificado ficam um bocado aquém do limite máximo dos recursos de processamento dos servidores.

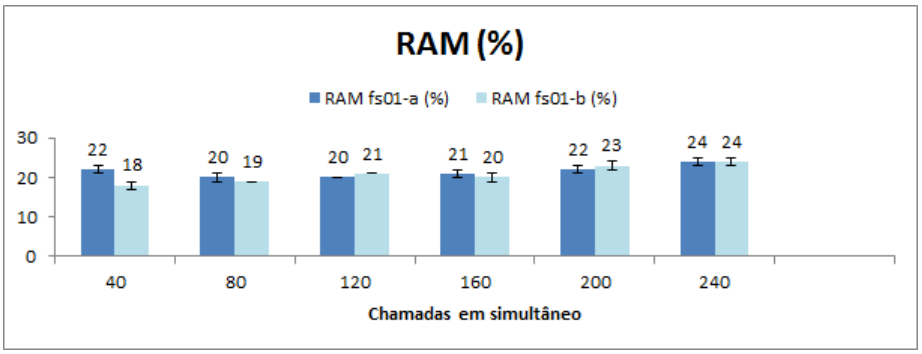

FIGURA 4.9: Consumo dos recursos da memória RAM

Finalmente, os valores da memória RAM apresentam-se estabilizados, não nos permitindo inferir nada de negativo.

4.2 Conclusões preliminares

Estes resultados permitiram-me perceber de que forma é que os sistemas de telefonia IP em questão se comportam e o que podemos esperar deles nestas condições. O facto de querer ter um cenário real levou-me a testar os cenários numa rede partilhada. Esta decisão foi crucial no desempenho dos servidores, limitando-os no número máximo de chamadas que são capazes de tratar. Por conseguinte, não sendo agnóstico ao tráfego da rede, acredito que o número máximo de chamadas em simultâneo verificado tenha ficado aquém do que eventualmente se conseguiria atingir, tanto num cenário como noutro, se tivesse usado uma rede interna para o mesmo efeito. Em teoria era expectável que, tendo o dobro do processamento no cenário Activo/Activo, conseguiria efectuar o dobro das chamadas em simultâneo. Os valores aproximam-se muito. No cenário Activo/Passivo, nestas condições, consigo garantir 120 chamadas em simultâneo com MOS a roçar os 4, permitindo-nos inferir uma qualidade de voz aceitável, enquanto que no cenário Activo/Activo ficamo-nos pelas 200 chamadas, salvaguardando a qualidade das chamadas em curso no sistema.

4.3 Testes Complementares

De forma a complementar os primeiros testes, outros foram realizados no sentido de permitir fazer um *check-up* às funcionalidades gerais que as plataformas em questão deveriam cumprir.

As tabelas que se seguem pretendem especificar os tipos de teste realizados, os procedimentos seguidos, os resultados esperados e os resultados obtidos em ambos os cenários de teste.

Activo/Passivo

Teste	Procedimento	Resultado Esperado	Resultado Obtido	Comentário	Estado
Parar o Apache	Desligar o serviço Apache	O servidor passivo assume o controlo	O servidor passivo assume o controlo	-	OK
Parar o FreeSWITCH	Desligar o serviço FreeSWITCH	O servidor passivo assume o controlo	O servidor passivo assume o controlo	-	OK
Parar o Corosync	Desligar o serviço Corosync	O servidor passivo assume o controlo	O servidor passivo assume o controlo	-	OK
Parar o Pgpool	Desligar o serviço Pgpool	O servidor passivo assume o controlo	O servidor passivo assume o controlo	-	OK
Desligar o servidor Activo	Desligar o servidor activo com todos os serviços associados activos	O servidor passivo assume o controlo	O servidor passivo assume o controlo	-	OK
Detecção de falhas no Apache	Executar o *shell script* correspondente à detecção de falhas do Apache como uma cronjob no sistema, com um intervalo entre verificações de 1 minuto	Detectar HTTP Responses diferentes de "200 OK" e receber notificação via email. Detectar ainda o consumo do processador	O administrador recebe uma notificação sempre que o serviço detecta um HTTP Response diferente de "200 OK". No caso de o serviço estar a consumir mais de 50% do processador é notificado por email. No caso de ultrapassar os 70% o servidor é desligado e o administrador é notificado por email	Assim que o serviço Apache é desligado pelo administrador ou é tomada a decisão por parte da monitorização do script de o desligar, o segundo servidor passa ao estado activo	OK
Detecção de falhas no FreeSWITCH	Executar o *shell script* correspondente à detecção de falhas do FreeSWITCH como uma cronjob no sistema, com um intervalo entre verificações de 1 minuto	Receber notificações via email quando o processo FreeSWITCH excede os 50% do consumo do processador. Desligar este serviço quando ultrapassar os 70%, notificando o administrador	O administrador recebe uma notificação quando este processo ultrapassa os 50%, permitindo-o tomar algum tipo de acção sobre o mesmo. No caso em que o consumo ultrapassa os 70% o servidor é automaticamente desligado	Assim que o serviço FreeSWITCH é desligado pelo administrador ou é tomada a decisão por parte da monitorização do script de o desligar, o segundo servidor passa ao estado activo	OK

Tabela 4.2: Activo/Passivo 1 (parte 1)

Detecção de falhas no Postgres	Executar o *shell script* correspondente à detecção de falhas do Postgres como uma cronjob no sistema, com um intervalo entre verificações de 1 minuto	Receber notificações via email quando o processo Postgres excede os 50% do consumo do processador. Desligar este serviço quando ultrapassar os 70%, notificando o administrador	O administrador recebe uma notificação quando este processo ultrapassa os 50%, permitindo-o tomar algum tipo de acção sobre o mesmo. No caso em que o consumo ultrapassa os 70% o servidor é automaticamente desligado	Assim que o serviço Postgres é desligado pelo administrador ou é tomada a decisão por parte da monitorização do script de o desligar, o segundo servidor passa ao estado activo	OK
Detecção de falhas no Pgpool	Executar o *shell script* correspondente à detecção de falhas do Pgpool como uma cronjob no sistema, com um intervalo entre verificações de 1 minuto	Receber notificações via email quando o processo Pgpool excede os 50% do consumo do processador. Desligar este serviço quando ultrapassar os 70%, notificando o administrador	O administrador recebe uma notificação quando este processo ultrapassa os 50%, permitindo-o tomar algum tipo de acção sobre o mesmo. No caso em que o consumo ultrapassa os 70% o servidor é automaticamente desligado	Assim que o serviço Pgpool é desligado pelo administrador ou é tomada a decisão por parte da monitorização do script de o desligar, o segundo servidor passa ao estado activo	OK
Online Recovery	Depois de recuperar o nó falhado, é executada a função pcp_recovery_node no servidor que está activo. Este, por sua vez vai proceder à cópia da sua base de dados para o nó recuperado. Nesta fase de cópia, tratando-se de uma recuperação online, mais transacções podem ocorrer durante o processo. Por esta razão existe uma segunda fase de cópia destas transacções que entretanto podem ter ocorrido	Base de dados igual nos dois servidores, actualizada e completamente funcional	Base de dados igual nos dois servidores, actualizada e completamente funcional	-	OK

Tabela 4.3: Activo/Passivo 2 (parte 2)

Foi feito um esforço no sentido de conseguir o comportamento adequado das soluções. No cenário Activo/Passivo, para além da configuração necessária ao funcionamento da solução, que nos permitiu ao desligarmos um serviço que a máquina activa se desligasse e a passiva assumisse o controlo, foi necessária a implementação de mecanismos de detecção de falhas de *software*. Se, por alguma razão, um serviço não se desligasse mas apresentasse um comportamento estranho, como por exemplo ter um elevado consumo de recursos sem o justificar, seria necessário detectá-lo, notificar o administrador e, em último caso, desligar o servidor. Por essa razão, foram desenvolvidos alguns *scripts* que permitiram monitorizar o estado dos serviços críticos envolvidos na solução de telefonia IP. Poderá ser vista a configuração destes *scripts* no anexo G, Mecanismos de detecção de falhas, juntamente com a sua explicação e forma de execução.

Activo/Activo

Teste	Procedimento	Resultado Esperado	Resultado Obtido	Comentário	Estado
Parar o Apache	Desligar o serviço Apache num servidor	Os pedidos HTTP são balanceados para o outro servidor. Os pedidos SIP continuam a ser servidos por este servidor	Os pedidos HTTP são balanceados para o outro servidor. Os pedidos SIP continuam a ser servidos por este servidor	Quando este serviço deixa de funcionar o servidor correspondente é automaticamente desligado	OK
Parar o FreeSWITCH	Desligar o serviço FreeSWITCH num servidor	O servidor detecta que o serviço não está disponível e desliga-se automaticamente	O servidor detecta que o serviço não está disponível e desliga-se automaticamente	Quando este serviço deixa de funcionar o servidor correspondente é automaticamente desligado	OK
Parar o Pgpool	Desligar o serviço Pgpool num servidor	O servidor detecta que o serviço não está disponível e desliga-se automaticamente	O servidor detecta que o serviço não está disponível e desliga-se automaticamente	Quando este serviço deixa de funcionar o servidor correspondente é automaticamente desligado	OK
Desligar o primeiro servidor	Desligar o servidor com todos os serviços associados activos	O outro servidor assume o controlo dos serviços	O outro servidor assume o controlo dos serviços	-	OK
Desligar o segundo servidor	Desligar o servidor activo com todos os serviços associados activos	O outro servidor assume o controlo dos serviços	O outro servidor assume o controlo dos serviços	-	OK
Detecção de falhas no Apache	Executar o *shell script* correspondente à detecção de falhas do Apache como uma cronjob no sistema, com um intervalo entre verificações de 1 minuto	Detectar HTTP Responses diferentes de "200 OK", receber notificação via email e desligar o servidor automaticamente	O administrador recebe uma notificação sempre que o serviço detecta um HTTP Response diferente de "200 OK". No caso de o serviço estar a consumir mais de 50% do processador é notificado por email. No caso de ultrapassar os 70% é	Assim que o serviço Apache é desligado pelo administrador ou é tomada a decisão por parte da monitorização do script de o desligar, o segundo servidor activo assume todas as requisições HTTP que entretanto cheguem	OK

Tabela 4.4: Activo/Activo 1 (parte 1)

			enviada uma notificação ao administrador e o servidor é automaticamente desligado.		
Detecção de falhas no FreeSWITCH	Executar o *shell script* correspondente à detecção de falhas do FreeSWITCH como uma cronjob no sistema, com um intervalo entre verificações de 1 minuto	Receber notificações via email quando o processo FreeSWITCH excede os 50% do consumo do processador. Desligar este serviço quando ultrapassar os 70%, notificando o administrador	O administrador recebe uma notificação quando este processo ultrapassa os 50%, permitindo-o tomar uma acção. No caso em que o consumo ultrapassa os 70% o servidor é automaticamente desligado	Assim que o serviço FreeSWITCH é desligado pelo administrador ou é tomada a decisão por parte da monitorização do script de o desligar, o segundo servidor assume o tráfego SIP	OK
Detecção de falhas no Postgres	Executar o *shell script* correspondente à detecção de falhas do Postgres como uma cronjob no sistema, com um intervalo entre verificações de 1 minuto	Receber notificações via email quando o processo Postgres excede os 50% do consumo do processador. Desligar este serviço quando ultrapassar os 70%, notificando o administrador	O administrador recebe uma notificação quando este processo ultrapassa os 50%, permitindo-o tomar uma acção. No caso em que o consumo ultrapassa os 70% o servidor é automaticamente desligado	Assim que o serviço Postgres é desligado pelo administrador ou é tomada a decisão por parte da monitorização do script de o desligar, o segundo servidor assume os serviços com a sua BD actualizada	OK
Detecção de falhas Pgpool	Executar o *shell script* correspondente à detecção de falhas do Pgpool como uma cronjob no sistema, com um intervalo entre verificações de 1 minuto	Receber notificações via email quando o processo Pgpool excede os 50% do consumo do processador. Desligar este serviço quando ultrapassar os 70%, notificando o administrador	O administrador recebe uma notificação quando este processo ultrapassa os 50%, permitindo-o tomar uma acção. No caso em que o consumo ultrapassa os 70% o servidor é automaticamente desligado	Assim que o serviço Pgpool é desligado pelo administrador ou é tomada a decisão por parte da monitorização do script de o desligar, o segundo servidor assume os serviços com a sua BD actualizada	OK
Online Recovery	Depois de recuperar o nó falhado, é executada a função	Base de dados igual nos dois servidores, actualizada e	Base de dados igual nos dois servidores, actualizada e	-	OK

Tabela 4.5: Activo/Activo 2 (parte 2)

	pcp_recovery_node no servidor que está activo. Este, por sua vez vai proceder à cópia da sua base de dados para o nó recuperado. Nesta fase de cópia, tratando-se de uma recuperação online, mais transacções podem ocorrer durante o processo. Por esta razão existe uma segunda fase de cópia destas transacções que entretanto podem ter ocorrido	completamente funcional	completamente funcional		

Tabela 4.6: Activo/Activo 3 (parte 3)

O mesmo que foi dito em relação ao cenário anterior se aplica a este. Os *scripts* de avaliação do estado dos serviços são os mesmos, podendo ser configurados à medida das nossas necessidades. Por exemplo, se soubermos que não é um comportamento normal que o serviço de base de dados Postgresql tenha um processamento superior a 70%, podemos configurar o *script* correspondente de forma a notificar o administrador de sistemas e/ou, inclusivamente, desligar este serviço.

4.4 Online Recovery

A *online recovery* apresenta-se neste projecto como um *nice-to-have*. Inicialmente não estava pensada a criação de um mecanismo deste tipo, nem foi sequer alocado tempo para a execução desta tarefa. Este módulo faz parte da ferramenta de replicação de dados Pgpool e diz respeito à reposição de um servidor de base de dados, depois de ele ter falhado. Imaginemos que temos dois nós com sincronização de dados activa. A partir do momento em que um nó, por alguma razão, falha, deixamos de ter os dados actualizados entre os dois servidores de base de dados. Desta forma, aquando a reposição do nó que falhou, num caso normal, só teríamos uma possibilidade: parar o nó activo para proceder à cópia dos dados entre os dois servidores, de modo a ficarem ambos actualizados e poderem responder aos pedidos dos clientes de forma consistente. De outra forma, caso tentássemos copiar a base de dados do nó activo sem o desligar primeiro, poderíamos correr o risco de perder dados, já que, durante o processo de cópia, poderia haver clientes a inserir e/ou modificar dados no sistema. É aqui que entra a *online recovery*. Quando é detectada uma falha num servidor e se pretende fazer uma reposição sem que seja necessário desligar a base de dados do servidor activo, este mecanismo garante que a cópia dos dados se faz sem perda de dados.
O fluxo de recuperação de um servidor através deste método segue-se na figura abaixo. A explicação encontra-se depois da figura.

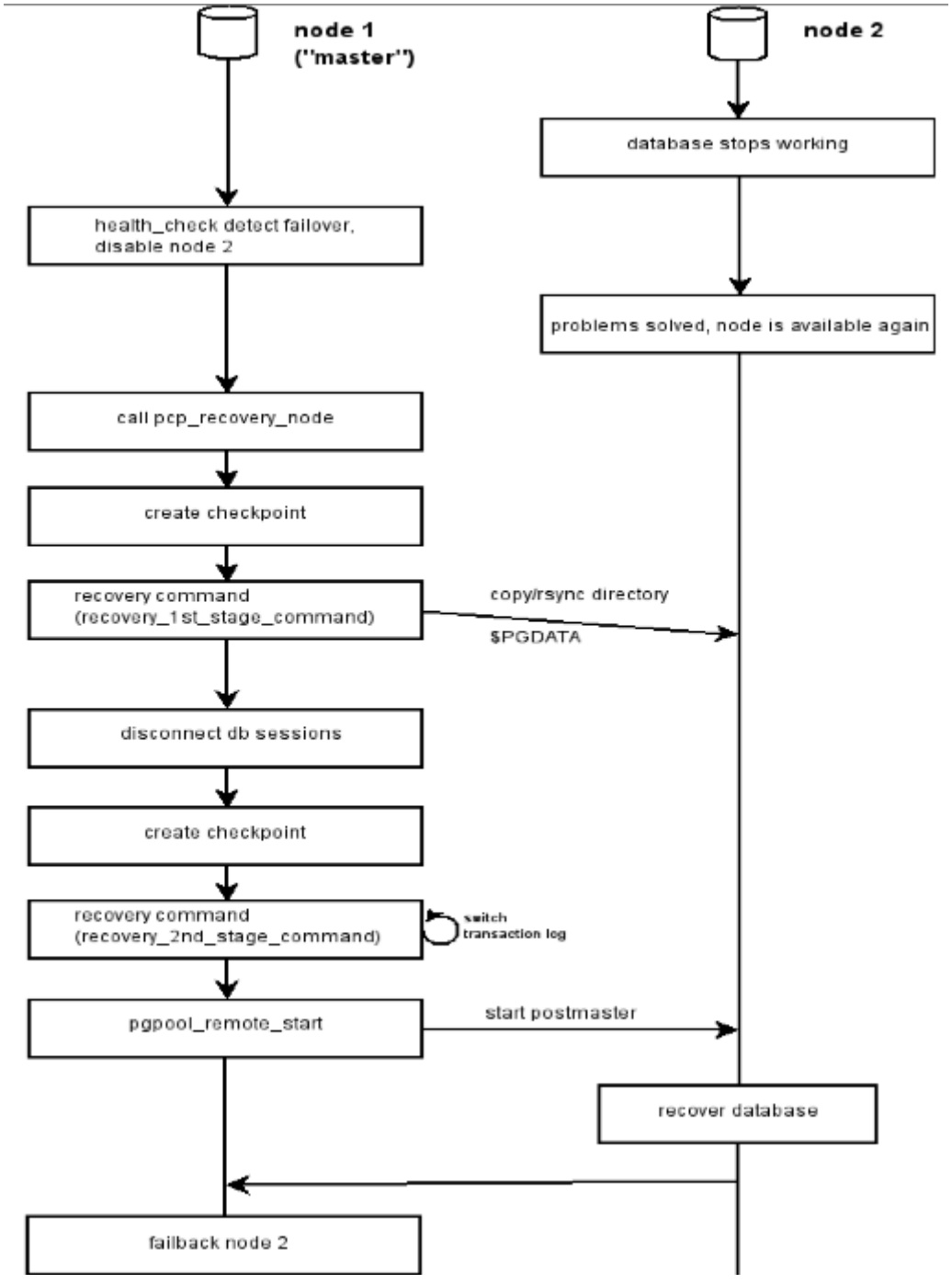

FIGURA 4.10: Pgpool Online Recovery - fonte: http://www.pgpool.net/pgpool-web/contrib_docs/pgpool-II_for_beginners.pdf

Quando é detectada uma falha de conectividade ao segundo servidor pelo Pgpool, este é automaticamente removido da *pool* de servidores de base de dados disponíveis para executar *queries* à base de dados. Para recuperar este nó, no servidor activo, executa-se a função pcp_recovery_node que vai fazer uma recuperação do servidor que falhou em duas fases:

- Na primeira fase é realizada uma cópia da base de dados para o servidor que tinha inicialmente falhado e que está agora activo, livre de problemas. Como durante esta fase de cópia podem ocorrer *queries* que podem adicionar ou alterar dados na base de dados do servidor-mestre, torna-se vital ter um ficheiro de *logs* que guarde todas as transacções realizadas à base de dados durante este período;
- Na segunda fase, esses *logs* são copiados para o servidor remoto, assegurando que este fica o mais actualizado e sincronizado com a base de dados do servidor-mestre possível.

Capítulo 5

Plano de Trabalho e Implicações

Este capítulo visa a apresentação do planeamento definido para o primeiro e segundo semestres, bem como a descrição das tarefas agendadas e eventuais desvios sofridos.

5.1 Planeamento 1º Semestre

Para o primeiro semestre foi criado um plano de trabalho à medida das necessidades do projecto a desenvolver, tendo em conta o eventual impacto que a elaboração de trabalho interno na empresa Wavecom pudesse trazer. Por esta razão estipulei prazos o suficientemente grandes de forma a poder, dentro dos prazos previstos, atingir os objectivos a que me propus inicialmente. Foi possível cumprir com todos os objectivos dentro dos prazos previstos, à excepção da definição da arquitectura do sistema que levou sensivelmente mais tempo, por razões que se prendem com a prioritização de trabalho interno à empresa. Isto não causou qualquer impacto no desenvolvimento do trabalho e no cumprimento dos restantes objectivos dentro dos prazos especificados, pelo que, salvo essa excepção, estou em crer que o trabalho realizado não fugiu ao previsto e que os objectivos definidos no planeamento para o primeiro semestre foram largamente cumpridos.

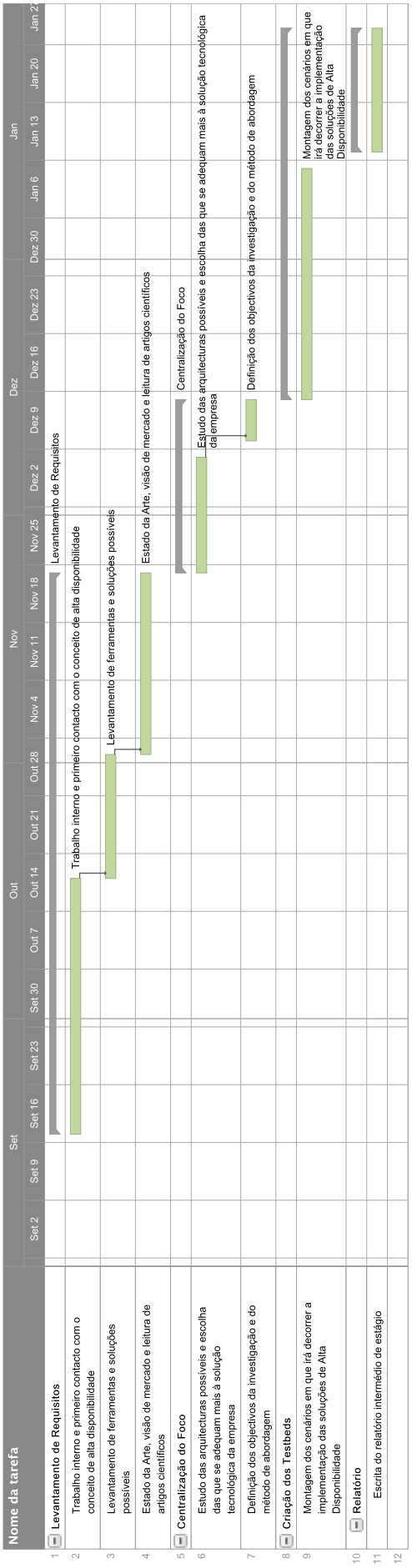

FIGURA 5.1: Planeamento 1º Semestre

5.2 Planeamento 2º Semestre

Este planeamento foi elaborado de modo a apresentar os objectivos a que me propus realizar no segundo semestre, tendo em consideração os prazos de que dispunha para os atingir, bem como as dificuldades que deveriam surgir e a margem de tempo estimada para a sua resolução. Neste planeamento, foram ainda representados os tempos estimados para testar as soluções e tratar os dados a incluir no relatório final.

A primeira tarefa agendada para o segundo semestre dizia respeito à preparação dos cenários em que iria decorrer a implementação. Esta tarefa pressupunha a criação de uma bancada de testes que, através da instalação de máquinas e das ferramentas necessárias à solução, permitiria realizar testes e perceber se a solução de alta disponibilidade se comportava de acordo com os requisitos previstos. Para esta tarefa foi definido um prazo relativamente grande de forma a contornar eventuais problemas que a configuração das máquinas nos dois cenários a implementar pudessem trazer. No entanto, a necessidade da empresa em implementar uma solução de recuperação da base de dados em modo *Online*, isto é, sem que fosse necessário desligá-la, levou ao atraso da construção da solução Activo/Passivo. Foi também nesta altura que decidi dar início à construção dos *scripts* de detecção de falha dos serviços, sugeridos pelo júri aquando da defesa intermédia de estágio. Toda esta implementação e fase de testes acabaram por atrasar um pouco o que estava planeado.

A segunda tarefa remeteu-me para uma fase de testes. A realização destes testes permitiu-me não só validar as soluções implementadas como também antecipar algumas conclusões. Nesta tarefa tinha, uma vez mais, definido um intervalo de tempo que me desse algum conforto, que me permitisse, em caso de necessidade, reconfigurar os cenários e ferramentas adoptadas. Tal não foi necessário e, por essa razão, o atraso associado à primeira tarefa anulou-se.

Seguiu-se a recolha e tratamento de dados. Esta tarefa incumbiu-me de realizar um plano de testes à medida, que me permitisse obter resultados que servissem de suporte à crítica final das minhas soluções, com base nas métricas de QoS que tinham sido definidas. Esta tarefa levou-me à elaboração de gráficos estatísticos que me permitiram dar a conhecer o comportamento das soluções quando sujeitas a condições de *stress* e, também, facilitar a visualização dos resultados ao leitor.

FIGURA 5.2: Planeamento 2º Semestre

A última tarefa diz respeito à escrita do relatório. Esta permitiu-me desenvolver uma reflexão sobre os resultados obtidos e referir de que forma este trabalho pode catapultar a realização de novos projectos.

Capítulo 6

Conclusões

Foi possível, ao longo do primeiro semestre, identificar as ferramentas de que disponho para a criação de uma solução de alta disponibilidade e de que forma elas devem coexistir de modo a cumprir com os requisitos de qualidade da solução a implementar. Foi igualmente importante perceber que metodologias existem para a replicação da base de dados e qual a ferramenta que melhor se adequa à concretização desse fim, já que é na base de dados que consta toda a informação dinâmica, relevante para o arranque de um sistema de telefonia IP em pleno estado de funcionamento no caso de uma falha. No segundo semestre, foi possível proceder à implementação dos sistemas de alta disponibilidade e perceber a sua relevância no contexto da telefonia IP, respondendo às necessidades dos clientes e garantindo a qualidade e preservação das suas comunicações. Apesar de os resultados obtidos serem favoráveis à implementação deste tipo de sistemas, concluo que a utilização do DNS para balanceamento de carga poderá não ser a melhor opção. Havendo *software* de balanceamento de carga nativo capaz de, não só balancear, como também olhar para a carga actual de cada um dos servidores em determinado momento, leva-me a colocar esta implementação como uma tarefa a realizar futuramente. Futuramente, também, serão levados a cabo novos testes. O sistema de telefonia IP usado permite o barramento de chamadas quando os recursos de processamento de um servidor atingem um determinado valor percentual. Desta forma, como tivemos oportunidade de ver, ao invés de o sistema continuar a aceitar o processamento de chamadas sob pena de degradar todas as outras no sistema, ele irá descartá-las, garantindo a qualidade das chamadas em curso e preservando o bom desempenho do sistema. Com a realização

deste trabalho pode concluir-se que a melhor solução a implementar numa perspectiva de custo-benefício é a do cenário Activo/Activo. Isto porque, a ter dois servidores ligados, em que apenas um está activo a responder a pedidos dos utilizadores e o outro está em modo passivo, ligado, a consumir recursos, pronto a realizar um *failover*, mais vale ter dois servidores activos a responder a pedidos dos utilizadores. Desta forma, não só se garante a rentabilidade máxima do nosso sistema, como também conseguimos suportar um maior número de clientes. No entanto, não deve ser esquecido o seguinte facto: apesar de todas as vantagens inerentes à escolha do cenário Activo/Activo, numa situação em que ambos os servidores estiverem com elevada carga, é possível que o servidor de *failover* não consiga suportar a carga de ambos em caso de falha. No futuro, serão realizados mecanismos que permitam prever este tipo de situações. Estes testes foram realizados em máquinas virtuais com grandes limitações dos recursos físicos. É expectável que, em servidores físicos com outro tipo de recursos, estes resultados escalem e ultrapassem em muito estas limitações. Por fim, apresento as minhas contribuições fundamentais para a solução de alta disponibilidade da Wavecom:

- Desenvolvimento da solução de alta disponibilidade, que era o grande objectivo da empresa (cenário Activo/Activo);
- Desenvolvimento de uma segunda solução de alta disponibilidade como termo de comparação e prova de conceito (cenário Activo/Passivo);
- Implementação de uma solução de replicação síncrona de base de dados entre dois servidores (físicos ou virtuais) distintos;
- Implementação de uma plataforma em Lua que permite realizar um *benchmark* a qualquer solução de telefonia IP montada em FreeSWITCH;
- Desenvolvimento de mecanismos de detecção de falhas de *software* ao nível dos serviços;
- Elaboração de testes-guia que permitem percorrer todas as funcionalidades e eventuais defeitos das soluções com vista à sua validação.

Anexo A

Soluções de Telefonia IP

Existem algumas ferramentas de telefonia IP, umas mais robustas que outras. No entanto, ao fazer uma pesquisa sobre as ferramentas principais existentes, reparamos que a ferramenta mais popular, da qual derivaram mais projectos, é o Asterisk. De forma a consubstanciar a escolha da empresa pela ferramenta FreeSWITCH, este anexo pretende descrever de forma geral estas duas soluções, com base na documentação oficial do FreeSWITCH[1] e do Asterisk[2].

A.1 FreeSWITCH

O FreeSWITCH é uma solução de telefonia IP bastante poderosa. Podemos definilo como um *soft-switch OpenSource* multi-plataforma concebido para ser escalável e estabelecer a comunicação entre diferentes protocolos de comunicação, usando qualquer forma de *media*.

Trata-se de uma ferramenta capaz de lidar com várias necessidades no que respeita a aplicações de voz, desde encaminhamento de sinalização SIP, *media* RTP/SRTP, IVRs (*Interactive Voice Response*), etc.

Para além de fornecer uma plataforma estável de telefonia onde muitas aplicações de telefonia podem ser desenvolvidas, permite a fácil integração com aplicações externas.

[1] http://www.freeswitch.org/
Consultado a 30/10/2012

[2] http://www.asterisk.org/
Consultado a 30/10/2012

Suporta várias tecnologias de comunicação como o Skype, SIP, H.323 e GoogleTalk, tornando fácil a comunicação com outros sistemas PBX *OpenSource* como o Asterisk.

Foi originalmente concebido e implementado por Anthony Minessale com a ajuda de Brian West e Michael Jerris. Estes três eram os programadores mais antigos do *software OpenSource* Asterisk. O projecto teve como foco inicial vários objectivos como a introdução de modularidade, suporte multi-plataforma, escalabilidade e estabilidade. Actualmente, muitos mais programadores contribuem para o projecto.

O FreeSWITCH suporta características avançadas como o *Presence* que nos permite saber se determinado telefone está ou não ligado, o BLF que permite a monitorização de linhas telefónicas num sistema e o SLA que permite ter múltiplos telefones registados na mesma extensão. No que diz respeito a segurança, suporta encriptação TLS e SRTP, podendo ser usado como SBC. Para além da possibilidade de enviar fax através da *media* RTP, permite também o envio sobre IP.

Suporta codecs de banda larga ou estreita, tornando-o uma solução ideal para interligar dispositivos antigos e recentes. O Codec G.729 também está disponível sob uma licença comercial.

Disponibiliza uma API através da qual é possível que programas escritos em linguagens de programação como Python, Java, Perl, Lua, Javascript, PHP (entre outras) e ainda algumas *frameworks*, se possam ligar ao *Event Socket* do mesmo e possam, desta forma, desencadear chamadas, entre outras funcionalidades, conseguindo-se desta forma um bom nível de abstracção.

A arquitectura do FreeSWITCH foi montada de forma a evitar *deadlocks* e outros problemas enfrentados por utilizadores de outras ferramentas, nomeadamente o Asterisk. Para além disso, o núcleo do FreeSWITCH é consistente e não depende de módulos externos, coisa que acontece no Asterisk.

Os estados de cada sessão nova criada são geridos por uma máquina de estados que é parte do núcleo do FreeSWITCH. Cada transição de estado implica a criação de um evento, podendo qualquer módulo actuar sobre ele. Esta coerência elimina muitas resoluções de problemas necessárias para alcançar os mesmos objectivos num *soft-switch* moderno.

O FreeSWITCH lê as suas configurações a partir de ficheiros XML que contêm as configurações necessárias ao sistema. No entanto, qualquer módulo pode ir ler estas configurações. Desta forma, é possível usar o módulo *mod_xml_curl* do FreeSWITCH para solicitar todas as configurações via HTTP, em vez de ler os ficheiros XML do disco.

Portanto, estamos perante um *software* de arquitectura simples, com configuração centralizada e desempenho escalável. Quer isto dizer que, se fosse necessário expandir a rede por forma a termos mais clientes e portanto, mais extensões, apenas teríamos de adicionar mais servidores com o FreeSWITCH, igual aos outros já existentes. As configurações podem ser puxadas remotamente e não se limitam à adição de utilizadores, mas também a toda e qualquer opção configurável do *software*, incluindo o *dialplan* - ficheiro XML que contém toda a lógica de comunicação, usado para tomar decisões de encaminhamento sobre uma determinada chamada.

Resta dizer que o FreeSWITCH surge depois do Asterisk e que veio colmatar alguns dos seus erros. Os programadores envolvidos neste projecto de telefonia não só contribuem para o seu próprio projecto, como também disponibilizam o seu código e outros recursos a outros projectos de telefonia IP.

A.2 Asterisk

O Asterisk apresenta-se como a solução de telefonia IP *OpenSource* mais adoptada em plataformas de comunicações.

Trata-se de um *software OpenSource* que permite a transformação de um simples computador numa central telefónica com suporte para múltiplos protocolos. Funciona em várias distribuições Unix com ou sem ligação à rede pública de telefonia PSTN.

Tal como o FreeSWITCH, apresenta características como distribuição automática de chamadas, IVRs, correio de voz, *call detail records*, administração via Web, compatibilidade com PBXs analógicos, telefones digitais IP, conectividade a partir de *trunks* analógicos e digitais, etc.

O Asterisk funciona com módulos. Estes módulos são usados para implementar protocolos específicos como SIP, adicionar aplicações como IVRs customizáveis e ligar-se a outras interfaces externas, como por exemplo a interface de gestão. O núcleo do

Asterisk funciona num modelo em *threads* e apenas os canais que originam determinadas aplicações as detêm. Por exemplo, a *leg B* [1] de uma chamada opera unicamente com a mesma *thread* da *leg A* que a detém. Quando algum evento acontece, como por exemplo a transferência de canal de uma chamada, a *leg B* vai ser manipulada e, ao não ser detentora da *thread*, deverá ser transferida para um modo de *thread*, obrigando a que o conteúdo de um canal seja tirado de um objecto de memória dinâmico e seja colocado noutro. Isto é o exemplo de uma má prática que foi inclusive descrita no código do Asterisk. Está má prática implica fazer uma espécie de *hacking* à estrutura do CDR de forma a evitar ter-se o registo de uma nova chamada. De outra forma, seriam vistas 3 ou 4 canais para uma única chamada durante uma transferência de chamada.

Portanto, este mecanismo incerto de *threads* foi um dos factores que motivou os programadores antigos do Asterisk a fazer o FreeSWITCH.

Outro ponto contra o Asterisk é o facto de usar um tipo de estrutura de dados designado por listas-ligadas, de forma a gerir os seus canais abertos. Trata-se de uma boa prática de programação mas, quando usada numa aplicação baseada em *threads*, torna-se muito difícil de gerir. Podem ser usados semáforos mutex de forma a garantir que apenas uma *thread* de cada vez tem acessos de escrita à lista-ligada ou corre-se o risco de estar a destruir um canal quando ele está a ser utilizado por outro recurso, levando a um tão conhecido erro - *Segmentation Fault*, que, em muitos dos casos, implicará a perda total das chamadas do sistema.

Para além dos defeitos descritos nos pontos anteriores, o núcleo do Asterisk tem dependências em alguns dos seus módulos, o que significa que determinada aplicação não começará se um determinado módulo não estiver presente. Portanto, há certo tipo de funcionalidades que deveriam ser arrancadas unicamente a partir do núcleo do Asterisk mas que, pelo contrário, só são possíveis de realizar através do código existente noutros módulos.

Finalmente, o Asterisk não tem protecção para a sua API. A maioria das funcionalidades e estruturas de dados são públicas e podem ser facilmente mal utilizadas. Muitos algoritmos estão repetidos ao longo do código de maneiras completamente diferentes, com cada aplicação a fazer algo diferente em operações aparentemente idênticas.

[1] Uma chamada tem duas *legs*. A *leg A* é a parte da chamada que bate no sistema. A *leg B* é a parte da chamada que sai do sistema em direcção ao destino.

Por todas estas razões, pelo tempo que demoraria a reescrever, apagar e a repensar o código do Asterisk, os três programadores Anthony Minessale, Brian West e Michael Jerris decidiram criar e lançar uma nova plataforma, o FreeSWITCH, que, em comparação com o FreeSWITCH implementa muitos mais procotolos, como por exemplo os de *messaging* tipo XMPP, entre outros. No fundo, muito resumidamente, o Asterisk é um IPPbx enquanto que o FreeSWITCH é um *softswitch* mais rico em protocolos e funcionalidades.

Anexo B

Soluções de Replicação de Dados

Existem várias ferramentas à nossa disposição para a realização da replicação de bases de dados PostgreSQL. No entanto, com o passar do tempo, alguns projectos ficaram parados e outros nunca atingiram um grau de maturação aceitável. Este anexo pretende consubstanciar as ferramentas referidas ao longo do capítulo 3, descrevendo-as com maior grau de detalhe.

B.1 Bucardo

É um mecanismo de replicação *OpenSource* que permite operar em modo multi-mestre ou multi-escravo. É assíncrono e baseado em *triggers*, ou seja, sempre que algo for alterado na base de dados principal, será executado um *trigger* que se responsabiliza por notificar um *daemon* que replica as modificações de dados na base de dados do servidor escravo. É também baseado em linhas (*row-based*), logo, qualquer modificação numa linha é registada no servidor mestre e depois aplicada no servidor escravo.

Permite a criação de dois modelos de replicação num *cluster*:

- Replicação assíncrona mestre-escravo em cascata, *row-based*, utilizando *triggers*;
- Replicação assíncrona multi-mestre, *row-based*, utilizando *triggers* e resolução de conflitos.

Requer uma base de dados dedicada e utiliza um *daemon* Perl que comunica com esta base de dados e com todas as outras bases de dados envolvidas no processo de replicação.

A replicação multi-mestre é limitada a duas bases de dados, com resolução de conflitos para lidar com a mesma actualizaçao de ambos os lados.

A replicação mestre-escravo envolve a replicação dos dados do servidor mestre para um ou mais servidores escravo.

É possível ter encadeamento, como por exemplo: o servidor A e B podem ser multi-mestre e o B pode também ser servidor mestre dos servidores escravos C, D e E. Por sua vez, o servidor E pode ser mestre dos servidores escravo F, G e H.

Vantagens:

- Suporta balanceamento de carga através dos servidores escravo;
- Suporta *data warehousing* através dos servidores escravo;
- Os servidores escravo podem aceitar escritas de dados;
- Permite a alteração de dados em tempo real, mesmo durante a replicação;
- Permite replicação parcial de uma base de dados;
- Permite replicação *on demand* - as alterações podem ser replicadas automaticamente ou quando se queira.

Desvantagens:

- Não permite a execução de *queries* em paralelo;
- Não consegue usar *triggers* DDL, uma vez que o PostgreSQL não suporta este tipo de *triggers*;
- Não consegue replicar tabelas que não tenham uma chave única;
- Não funciona em versões anteriores ao PostgreSQL 8.

O estudo desta ferramenta foi feito com base na documentação oficial do Bucardo[1].

[1] http://bucardo.org/
Consultado a 5/11/2012

B.2 Pgpool-II

O Pgpool-II é um *middleware* que funciona entre a aplicação (*frontend*) e o(s) servidor(es) PostgreSQL (*backend*).

Tem como características fundamentais:

- *Connection Pooling*: Guarda as ligações aos servidores PostgreSQL e volta a usá-las sempre que uma nova ligação é necessária, com as mesmas propriedades (*username*, *database*, versão do protocolo). Isto reduz o *overhead* de ligação à base de dados e melhora o débito final do sistema;
- Replicação: O Pgpool-II consegue gerir múltiplos servidores PostgreSQL. Permite criar um *backup* em tempo real em dois ou mais discos físicos de forma a que o serviço continue sem ter de parar os servidores no caso de uma falha de disco;
- Balanceamento de carga: Se uma base de dados é replicada, executando uma *query* SELECT em qualquer servidor, retornará sempre o mesmo resultado. O Pgpool-II tira partido da replicação para reduzir a carga em cada servidor PostgreSQL, distribuindo as *queries* SELECT por múltiplos servidores, melhorando o débito final do sistema. Na melhor das hipóteses, a *performance* melhora proporcionalmente ao número de servidores PostgreSQL que existirem. O balanceamento de carga funciona melhor numa situação em que haja muitos utilizadores a executarem *queries* ao mesmo tempo;
- Limite de ligações excedidas: Existe um limite máximo de ligações concorrentes ao PostgreSQL e, ultrapassado este limite, as ligações são rejeitadas. Ao alterar o número máximo de ligações, aumenta o consumo de recursos e afecta a *performance* global do sistema. O Pgpool-II também tem um limite no número máximo de ligações. No entanto, as ligações extra são colocadas em fila, ao invés de retornar logo um erro;
- *Queries* Paralelas: Os dados podem ser divididos ao longo de múltiplos servidores de forma a que uma *query* possa ser processada em todos os servidores concorrentemente de modo a reduzir o tempo de execução final. As *queries* paralelas representam um acréscimo de *performance* quando se realizam operações de consulta de grandes quantidades de dados.

No que respeita a escalabilidade, o Pgpool-II é escalável até 128 nós. Quanto a operações de leitura é completamente escalável. Quanto a operações de escrita é possível efectuar escritas até 128 nós mas a performance cai para 60-70%.

Funcionalidades e respectivos modos suportados:

- Permite replicação síncrona;
- Suporta *triggers* e procedimentos;
- Sistema de replicação baseado em *queries*;
- Permite ter três modos de execução: *replication* (R), *master/slave* (M) and *parallel query* (P):
  - O *Connection Pooling* funciona nos três modos de execução: R, M e P;
  - O *Automatic Failover* funciona nos seguintes modos: R e M;
  - O *Online Recovery* pode ser usado nos modos: R e M, se se usar *Streaming Replication*).
- O modo mestre/escravo pode ser usado com o Slony-I e com *Streaming Replication*;
- Suporta uma GUI dedicada, chamada pgpoolAdmin, que permite monitorizar as bases de dados, administração e alterações em ficheiros de configuração.

O estudo desta ferramenta foi feito com base na documentação oficial do Pgpool[1].

[1] http://www.pgpool.net/
Consultado a 5/11/2012

B.3 Pgpool-HA

O Pgpool-HA é uma ferramenta que serve para evitar um único ponto de falha relativo ao Pgpool-II. Para a detecção de que um nó Pgpool-II falhou, utiliza o Heartbeat para promover a réplica do Pgpool-II a nó principal, garantindo a alta disponibilidade.

B.4 Rubyrep

É uma ferramenta *OpenSource* de replicação assíncrona de base de dados. Fornece replicação mestre-mestre para o PostgreSQL e para o MySQL.

Apresenta-se como fácil de instalar e configurar, sendo independente da plataforma, podendo ser executada em Linux e em Windows, apesar de em Windows ser consideravelmente mais lenta e ter sido menos testada, dado que se trata de um projecto ainda em fase de maturação.

É independente do *schema* da base de dados:

- Todos os comandos funcionam nas tabelas, não importando se estas têm ou não chave primária;
- Suporta grandes quantidades de dados. No MySQL foi testado com os tipos de dados "blob" e "text", enquanto que no PostgreSQL foi testado com o "bytea" e "text".

Permite comparar tabelas de forma a perceber divergências nos seus dados.

O estudo desta ferramenta foi feito com base na documentação oficial do Rubyrep[1].

[1] http://www.rubyrep.org/
Consultado a 5/11/2012

B.5 SkyTools - Londiste

O SkyTools é uma *framework* inicialmente desenvolvida e usada pelo Skype para simplificar a gestão e a replicação de bases de dados distribuídas, bem como para implementar soluções de *failover*. As suas características principais são o **PgQ queuing** e a replicação de sistemas com base no **Londiste**, para além de uma livraria para *scripts* escritos em Python. O PgQ é a implementação de uma fila na base de dados onde eventos são postos em fila e depois processados pelos clientes. Esta técnica é usada quando queremos processar dados assíncronos e quando queremos replicar dados ao longo de vários servidores ou distribuir transacções pelos mesmos. Basicamente, uma base de dados pode ter várias filas. Os *Producers* (servidores mestre) podem adicionar eventos a qualquer fila e podem haver vários *Consumers* (servidores escravo) que vão ler essa informação.

Na figura abaixo pode verificar-se o funcionamento do PgQ:

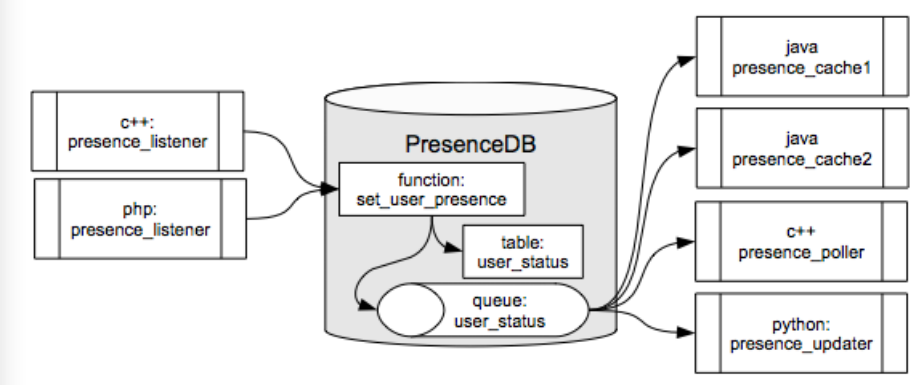

FIGURA B.1: PostgreSQL Queues - Fonte: http://wiki.postgresql.org/images/2/28/Moskva_DB_Tools.v3.pdf

Neste exemplo, é possível observar a base de dados que guarda o estado de presença dos utilizadores (*online*, *offline*, *busy*, etc.). É possível verificar que o *Producer* pode ser qualquer entidade que consiga executar o procedimento presence_listener, quer esteja escrito em C++, PHP, Java ou outra linguagem, tal como o *Consumer*.

O Londiste é uma ferramenta de replicação assíncrona mestre-escravo escrita em Python. Usa o PgQ como camada de transporte, necessitando do *Ticker* que é um *daemon* de gestão do PgQ.

No Skype, o Londiste é usado para:

- Transferir dados *online* para bases de dados internas;
- Criar bases de dados de *failover* para bases de dados *online*;
- Actualizar as versões do PostgreSQL;
- Distribuir dados internos para servidores *online*;
- Criar réplicas da base de dados (que aceitam *queries* de leitura) para balanceamento de carga.

Na figura abaixo pode perceber-se o funcionamento dos processos utilizados nesta ferramenta:

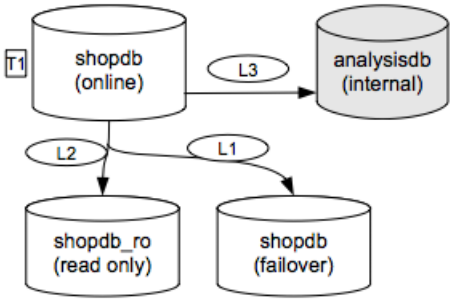

FIGURA B.2: Processos independentes - Fonte: http://www.pgcon.org/2009/schedule/attachments/101_Londiste3.pdf

Um processo Londiste faz a gestão do *Producer* (ou Produtor) e do *Consumer* (ou Consumidor). Por exemplo, o processo L3 trata da replicação das tabelas da base de dados shopdb para a analysisdb. A adição ou remoção de uma tabela não afecta a replicação das outras tabelas. Cada processo é independente, não comunicando com os outros.

O estudo desta ferramenta foi feito com base na documentação do PostgreSQL, referente ao Skytools[1] e também com recurso à documentação oficial [2].

[1] http://wiki.postgresql.org/wiki/Skytools
Consultado a 6/11/2012

[2] http://www.pgcon.org/2009/schedule/attachments/101_Londiste3.pdf
Consultado a 6/11/2012

B.6 Slony

O Slony é uma ferramenta de replicação assíncrona mestre-escravo para PostgreSQL que suporta servidores em cascata, isto é, podem ser criadas réplicas num nó que é responsável pela criação de réplicas noutros nós e assim sucessivamente, ou seja, não comunicam directamente com o servidor mestre, melhorando o desempenho deste. Também suporta *failover*. Pode ser combinado com um balanceador de carga como o Pgpool-II de forma a criar um sistema de *clustering* completo.

É normalmente escolhido para os seguintes casos:

- Replicação em cascata;
- Replicação parcial dos dados;
- Actualização do PostgreSQL;
- *Failover* e promoção do servidor escravo a mestre;
- Aprovisionamento *Online*;
- Maturo e amplamente testado.

É portanto uma ferramenta de replicação usada para replicação em cascata de um servidor mestre para múltiplos servidores escravos. Inclui as características necessárias para replicar bases de dados muito grandes até um número de servidores escravos razoável. Foi desenhado para ser usado em *datacenters* e para *backup* de *sites*, onde o modo normal de operação consiste em ter todos os nós disponíveis.

Desvantagens:

- Não se trata de uma solução de *cluster* completa;
- É complexa de configurar e de administrar;
- Tem limitações no que respeita a alterações no *schema* da base de dados a replicar;
- Desempenho no processamento de replicação de grandes dados;
- *Overhead* na escrita de dados;

- Atraso na replicação considerável o que poderá fazer com que os utilizadores possam visualizar dados inconsistentes da base de dados.

O seu funcionamento pode ser observado na figura abaixo:

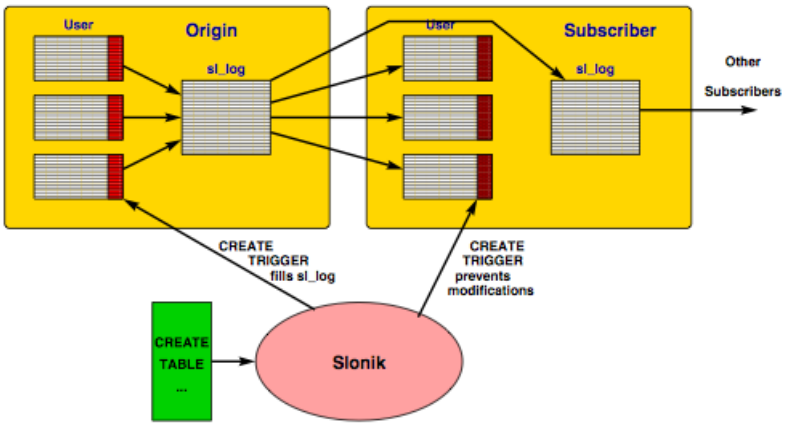

FIGURA B.3: Funcionamento Slony - Fonte: http://momjian.us/main/writings/pgsql/replication.pdf

Dispõe de um processo chamado Slonik para onde são enviadas as *queries*. Posteriormente, através de *triggers*, esse processo é responsável por enviar as *queries* ao servidor primário que, por sua vez, se encarrega de enviar os dados assincronamente para o(s) servidor(es) escravo(s).

O estudo desta ferramenta foi feito com base na documentação oficial Slony[1].

[1] http://slony.info/
Consultado a 7/11/2012

B.7 Mammoth

É uma ferramenta de replicação *OpenSource* assíncrona para *clusters* de configuração mestre-escravo, com suporte para *failover*. Trata-se de uma modificação do PostgreSQL, inserindo suporte para replicação.

Apresenta como características fundamentais:

- A possibilidade de ter *clusters* com múltiplos servidores escravo;
- Suporta a replicação de objectos grandes de dados;
- Suporta *switchover* e *failover*;
- Pode coexistir com outras ferramentas de replicação, como por exemplo o Slony;
- Suporta SSL, garantindo segurança na replicação dos dados;
- Permite replicação por tabela.

No entanto, apresenta as seguintes limitações:

- Não suporta a replicação de comandos DDL, como por exemplo o CREATE TABLE;
- Só consegue interagir com uma única base de dados;
- Não suporta servidores escravos em cascata, não permitindo a replicação entre os servidores escravo.

O estudo desta ferramenta foi feito com base na documentação do PostgreSQL, referente ao Mammoth[1].

[1] http://wiki.postgresql.org/wiki/Mammoth
Consultado a 7/11/2012

B.8 Postgres-XC

O Postgres-XC é um projecto *OpenSource* desenvolvido para fornecer soluções de *cluster* escaláveis, síncronas e transparentes. Trata-se de um conjunto de complementos de base de dados que podem ser instalados em mais do que um único *hardware* ou máquinas virtuais.

Apresenta as seguintes características:

- *Write-scalable*: Pode ser configurado com quantas bases de dados quantas quisermos, aguentando muitas mais escritas em comparação com o facto de se ter apenas um único servidor de bases de dados;
- Multi-mestre: Podemos ter mais do que um servidor de base de dados ao qual os clientes se ligam, permitindo uma perspectiva da base de dados única e consistente;
- Síncrona: Qualquer *update* realizado numa base de dados a partir de um servidor é imediatamente visível para quaisquer outras transacções que estejam a ser executadas em diferentes servidores primários;
- Transparente: As nossas aplicações não se têm de preocupar sobre como é que os dados são armazenados internamente nos servidores de base de dados.

Pode ser configurado para executar em múltiplos servidores. Os dados são armazenados de uma forma distribuída, fragmentada ou replicada, escolhido por nós. Ao requisitarmos *queries*, o Postgres-XC determina onde estão os dados armazenados e executa-as nos servidores de base de dados correspondentes.

O estudo desta ferramenta foi feito com base na documentação oficial do Postgres-XC[1].

[1] http://postgres-xc.sourceforge.net/
Consultado a 7/11/2012

B.9 Postgres-R

O Postgres-R é uma extensão ao sistema relacional de base de dados Postgres, fornecendo uma eficiente, rápida e consistente replicação multi-mestre em *clusters* de topologia *Shared Nothing*. Foi desenhado para ser tão transparente quanto possível para o cliente, sendo estável.

O principal motivo de utilização do Postgres-R é a implementação de alta disponibilidade e balanceamento de carga em sistemas de bases de dados. Devido à arquitectura flexível que apresenta, é facilmente possível estender ou ajustar o processo de replicação em variadas vertentes. Comparado com sistemas de bases de dados de um único servidor, um *cluster* Postgres-R é mais confiável e escala melhor.

Quanto a Confiabilidade, Disponibilidade e Escalabilidade, o espelhamento dos dados em múltiplos nós aumenta a confiabilidade do sistema de base de dados. Como o Postgres-R implementa um GCS (*Group Communication System*), é facilmente possível adicionar ou remover nós no sistema. Os nós que falham são automaticamente detectados e removidos. Assim, a normal execução do sistema de base de dados não é afectada. Isto facilita as tarefas de administração e garante melhor disponibilidade e escalabilidade do sistema de base de dados.

Quanto à *performance*, trata-se de uma solução de replicação assíncrona em que as *queries* de leitura podem ser facilmente distribuídas pelos nós do sistema. No entanto, é um projecto experimental, inviabilizando, por agora, a sua aplicação em sistemas reais.

O estudo desta ferramenta foi feito com base na documentação oficial do Postgres-R[1].

[1] http://www.postgres-r.org/
Consultado a 8/11/2012

B.10 NAS - *Network Attached Storage*

O NAS é um dispositivo dedicado ao armazenamento de ficheiros numa rede, permitindo aos utilizadores acederem aos dados através de protocolos de acesso remoto. Pode ser composto por vários discos e suportar RAID, permitindo a fragmentação dos dados pelos discos. Desta forma, quando determinado ficheiro for acedido, todos os discos envolvidos nesse processo de fragmentação executarão simultaneamente, aumentando a rapidez no acesso. Para além disso permite libertar recursos nos outros servidores, já que é o NAS que detém a responsabilidade de disponibilizar os ficheiros numa rede.

O desempenho de um NAS depende da quantidade de memória RAM e do tráfego da rede. Exige muito do processador quando existem muitos utilizadores em operações de I/O o que rapidamente contribui para a degradação de*performance* de um sistema.

É limitado ao seu *hardware*, não podendo ser combinado com outros servidores NAS com o objectivo de distribuir processamento.

B.11 DRBD

Foi concebido pela comunidade Linux-HA para configurar *clusters* de alta disponibilidade. Consiste num módulo que pode ser adicionado ao Kernel de um sistema Linux, permitindo o espelhamento de blocos de dados através da rede. Por esta razão é commummente associado a um sistema RAID-1. Os blocos de dados são espelhados de uma partição de um nó activo para uma outra de um nó *Standby*. No nó activo a partição está montada com o DRBD, ao contrário do nó *standby*, que fica impedido de realizar quaisquer operações de leitura de dados.

A partir da versão 8 foi possível a configuração de dois nós primários, podendo haver balanceamento de carga para além da alta disponibilidade. No entanto, isto requer sistemas de ficheiros distribuídos.

O DRBD funciona com partições de disco ou volumes virtuais LVM, espelhando cada bloco de dados escrito de um nó para outro.

O espelhamento pode ser feito de modo síncrono ou assíncrono. A implementação de um *cluster* de alta disponibilidade deverá seguir o modo síncrono onde não queremos perder quaisquer dados, no caso de uma falha do servidor activo.

O facto de os dados serem espelhados ao nível dos dispositivos de bloco permite que possamos aceder aos dados usando um sistema de ficheiros (apenas no nó activo).

O estudo desta ferramenta foi feito com base na documentação oficial do DRBD[1].

[1] http://www.drbd.org/
Consultado a 8/11/2012

B.12 *Log Shipping*

Servidores *Standby* de *Log Shipping*

O arquivamento contínuo de *logs* de transacção pode ser usado para criar um *cluster* de alta disponibilidade com um ou mais servidores *Standby* preparados para tomar o controlo das aplicações no caso de o servidor principal falhar. Esta capacidade é vulgarmente referida como ***Warm Standby*** ou ***Log Shipping***.

Normalmente, num servidor de base de dados, a memória principal é utilizada para se conseguir um bom nível de desempenho na gestão dos dados. No entanto, no caso de uma falha, os dados que estiverem em memória são perdidos. Por esta razão surge a necessidade de se ter uma solução de armazenamento intermédio no disco que permita o acesso rápido aos dados. Surgem então os ditos *logs* de transacção que possuem uma estrutura sequencial de forma a facilitar o acesso ao disco. No PostgreSQL designam-se **WAL - *Write Ahead Logs*** e localizam-se no directório $PGDATA/pg_xlog, tendo normalmente 16MB cada, podendo este valor ser alterado durante a fase de configuração do servidor. Todas as transacções que sofrerem *commit* são transferidas do WAL para o arquivo de dados e os *logs* são reciclados (***checkpoint***).

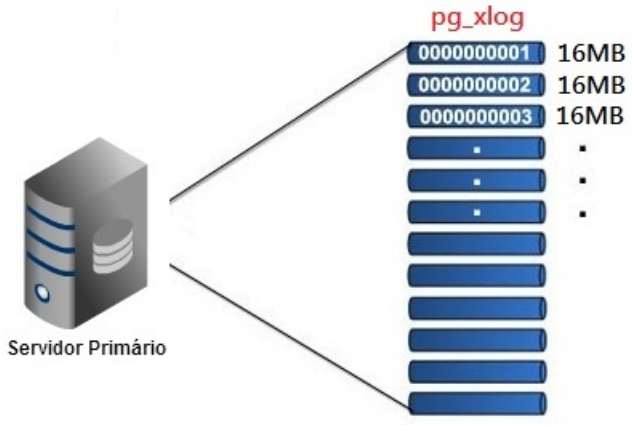

FIGURA B.4: Directorio do WAL - Adaptado de: http://blogs.dextra.com.br/bdextra/2009/replicacao-postgresql-warm-standby/

O servidor primário e o servidor *standby* trabalham juntos de forma a fornecer esta capacidade, embora sejam *loosely coupled*, ou seja, trabalham independentemente. O servidor primário opera em modo de arquivamento contínuo, enquanto que o servidor

standby opera em modo de recuperação contínua, lendo os ficheiros de WAL do servidor primário. Não são necessárias alterações nas tabelas da base de dados para disponibilizar esta capacidade, não havendo um grande *overhead* de administração associado. Esta configuração tem um impacto baixo no servidor primário, já que não afecta o processamento de *queries* que lhe sejam feitas.

As transacções em memória são, de tempos em tempos gravadas no WAL. As que sofreram *commit* são gravadas nos *logs* de forma síncrona. A partir deste momento estas podem ser recuperadas, através do *log*, no caso de uma falha no servidor. As transacções que não sofreram *commit* podem ter dois destinos:

- São perdidas, uma vez que se encontravam totalmente em memória.
- São desfeitas, se tiverem sido parcialmente gravadas no WAL.

A figura que se segue pretende representar essa mesma situação.

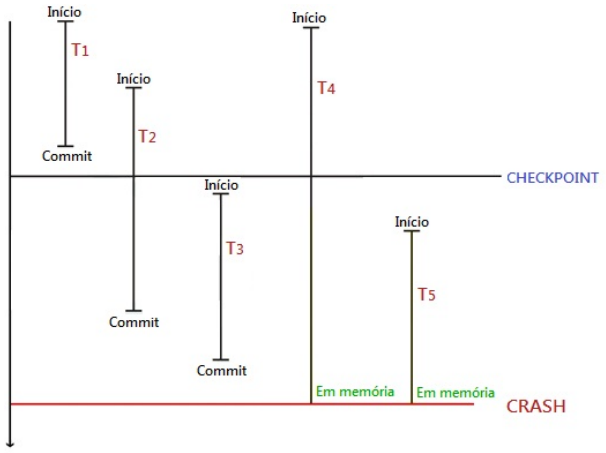

FIGURA B.5: Gestão de Transacções - Adaptado de: http://blogs.dextra.com.br/bdextra/2009/replicacao-postgresql-warm-standby/

Para recuperar as transacções do WAL, o PostgreSQL utiliza o método **REDO** que consiste em refazer toda a transacção com base nas informações presentes no WAL. Desta forma, no caso de uma falha, a transacção T1 estaria no arquivo após o *checkpoint*, enquanto que a T2 e a T3 estariam na sua totalidade no WAL, sendo necessária a execução do ***REDO*** para as refazer. A transacção T4 sofreria um ***Rollback*** enquanto que a T5 não sofreria qualquer operação uma vez que foi completamente eliminada por

não haver um *checkpoint* que a contemple e por o *crash* do sistema se ter dado quando esta transacção estava em memória.

No que respeita à replicação, depois de um *checkpoint* os ficheiros WAL são transferidos para um directório do servidor *standby*.

O armazenamento do WAL ocorre de forma assíncrona, não interferindo no desempenho do servidor de base de dados principal. A constante restauração do servidor *standby*, permite o início imediato do mesmo após um *crash* do servidor principal. Os *logs* podem permanecer para *backup*, tanto no servidor principal como no servidor *standby* ou podem ser eliminados após a restauração do servidor.

Esta solução (Warm Standby) não só barra o acesso a consultas no servidor secundário como consegue atingir um nível de actualização razoável entre este e o primário. O ***Warm Standby*** ou ***file-based Log Shipping*** é, portanto, uma técnica de replicação baseada em ficheiros que não cria qualquer canal de comunicação entre os dois servidores (primário e *standby*). Neste caso, o servidor primário é responsável por enviar segmentos de *logs* para o servidor *standby* de acordo com os seguintes critérios:

- Ocorrência de um *checkpoint* através de um intervalo de tempo definido no campo checkpoint_timeout, segundo o qual os segmentos de *log* são enviados;
- Preenchimento de determinada quantidade de segmentos de *log*, definida no campo checkpoint_segments;
- Ocorrência de uma cópia de segmentos de *log*, de acordo com um intervalo de tempo definido no campo archive_timeout.

Em baixo, pode ver-se o funcionamento deste tipo de replicação:

Trata-se de uma técnica de replicação realizada com base no ficheiro dos *logs* de transacção, onde só os registos num ficheiro completamente preenchido são enviados. O PostgreSQL arranca um processo chamado **pg_standby** no servidor *standby* que fica a aguardar por segmentos de *logs* que são enviados de tempos a tempos pelo servidor primário sempre que o ficheiro esteja completamente escrito. O servidor *standby* mantém-se ligado com este processo em execução, em modo de recuperação.

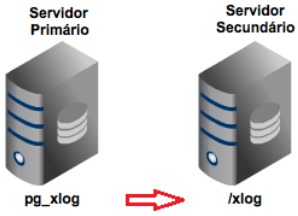

FIGURA B.6: Warm Standby

O PostgreSQL implementa ***Log Shipping* baseado em ficheiros** pela transferência de um ficheiro (ou segmento WAL) de cada vez. A largura de banda necessária para esta técnica varia de acordo com a taxa de transacção do servidor primário. Por outro lado, o ***Log Shipping* baseado em registos** é mais granular e as alterações no WAL são enviadas incrementalmente através de um canal (ou *stream*) de comunicação TCP/IP entre os dois servidores, num processo designado ***Streaming Replication***.

O *Log Shipping* é assíncrono, isto é, os registos WAL são enviados depois de um *commit* de uma transacção no servidor primário. Como resultado, existe uma janela de perda de dados associada. Se o servidor primário sofrer uma falha, as transacções que ainda não foram enviadas serão perdidas. O tamanho desta janela no *Log Shipping* baseado em ficheiros pode ser minimizado através do parâmetro arquive_timeout, que pode ser alterado para um valor reduzido de segundos. Contudo, quanto menor for o número de segundos, maior será a largura de banda necessária para a transferência dos ficheiros e portanto maior será o *overhead* global do sistema.

A *Streaming Replication* permite ter janelas muito menores de perda de dados. Permite que um servidor *standby* esteja o mais actualizado possível em comparação com o *Log Shipping* baseado em ficheiros, uma vez que oferece a capacidade de enviar de forma contínua e aplicar os registos WAL a um número variado de servidores de forma a mantê-los o mais frequentemente actualizados possível. Possibilita que um servidor secundário se mantenha em comunicação com o servidor primário através de um canal segundo o qual são transferidos os registos WAL para o servidor *standby* assim que estes tenham sido gerados no servidor primário, sem ter de se esperar que o ficheiro WAL seja completamente preenchido, tornando desta forma o envio e a recepção dos segmentos de *logs* mais fácil. A comunicação estabelece-se através dos processos ***Walsender*** e

Walreceiver, iniciados nos servidores primário e secundário, respectivamente. A criação deste canal tem início quando o ***Walreceiver*** faz um pedido de ligação ao servidor primário. Se a ligação for estabelecida, é criado o processo ***Walsender***. A partir deste momento, o ***Walsender*** envia ao ***Walreceiver*** as alterações no servidor primário a serem aplicadas no servidor secundário.

É assíncrona por defeito (mas pode ser configurada para ser síncrona), portanto existe ainda um pequeno atraso entre o *commit* de uma transacção no servidor primário e as mudanças se tornarem visíveis no servidor *standby*. O atraso, contudo, é muito menor do que no *Log Shipping* baseado em ficheiro, tipicamente menor que 1 segundo, assumindo que o servidor *standby* é suficientemente poderoso para lidar com a carga causada no sistema. Com a *Streaming Replication*, o parâmetro arquive_timeout não é necessário para reduzir a janela de perda de dados.

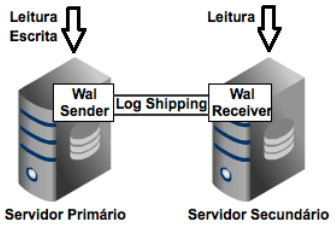

FIGURA B.7: Streaming Replication - Hot Standby

Pela complexidade de gestão e eventual perda de desempenho, a escrita simultânea em múltiplos servidores não é muito comum. Nos cenários com múltiplos servidores, estes deverão garantir que as aplicações tenham conhecimento de que há vários servidores que aceitam escritas, de forma a evitar conflitos ou, se quisermos, *deadlocks* (alteração concorrente do mesmo dado em dois servidores distintos). As soluções mais comuns são as que recebem alterações apenas num único servidor principal, que é responsável por enviar as alterações realizadas para os servidores secundários. Desta forma, caso o servidor principal falhe, podemos promover um secundário a servidor principal.

A *performance* deve ser considerada em qualquer escolha. Existe, normalmente, um *trade-off* entre funcionalidade e desempenho. Por exemplo, uma solução completamente síncrona, numa rede lenta, talvez tenha o seu desempenho cortado em mais de metade, enquanto que uma solução assíncrona terá um impacto menor no desempenho.

Anexo C

Soluções de Alta Disponibilidade no Mercado

Tendo já alguns anos, a alta disponibilidade, face às limitações que consegue ultrapassar, apresenta-se como uma solução cada vez mais necessária e de grande valor comercial. Por todas estas razões tem vindo a ser, ao longo dos anos, alvo de estudo por parte da comunidade científica, havendo também soluções de mercado bastante interessantes.

Desta forma, existem várias soluções comerciais que permitem a instalação de vários sistemas e aplicações. Em baixo, apresentam-se algumas delas.

C.1 VMware vSphere

O VMware vSphere permite a construção de um *datacenter* flexível e eficiente. Permite a execução de aplicações críticas e responder mais rapidamente às nossas necessidades, oferecendo uma plataforma de virtualização virada para a construção de infraestruturas de *cloud*. Este *software* permite a mudança de *datacenters* para a *cloud*, permitindo a gestão de máquinas.

Fornece alta disponibilidade em todo o ambiente virtualizado sem o custo ou complexidade das soluções tradicionais de *cluster*. É fácil de utilizar, rentável e garante a disponibilidade para aplicações que estejam a ser executadas em máquinas virtuais. No caso de

uma falha num servidor físico, as máquinas virtuais afectadas são reiniciadas automaticamente em outros servidores de produção. No caso de uma falha do sistema operativo, o vSphere HA reinicia a máquina virtual afectada no mesmo servidor físico.

O vSphere HA permite às organizações TI:

Minimizar o *downtime* de uma falha no servidor ou no sistema operativo

O vSphere HA oferece a disponibilidade necessária por muitas aplicações que corram em máquinas virtuais, independentemente do sistema operativo e da aplicação que esteja em execução. Fornece uma protecção rentável através de *failovers* contra falhas de *hardware* e do sistema operativo.

- Monitoriza as máquinas virtuais para detectar falhas de *hardware* e do sistema operativo;
- Reinicia as máquinas virtuais em servidores físicos sem que haja intervenção manual quando uma falha é detectada num servidor;
- Protege as aplicações de falhas do sistema operativo reinicializando automaticamente as máquinas virtuais quando uma falha no sistema operativo for detectada.

Aumentar a protecção ao longo da nossa infraestrutura

O vSphere HA pode ser configurado através da interface de cliente do vSphere que fornece protecção através de *failovers* sem ser necessária uma configuração complexa ou sequer configurações de soluções ao nível do sistema operativo. Sendo fácil de configurar e pouco exigente ao nível de requisitos mínimos, podemos:

- Proteger todas as aplicações contra falhas do servidor e do sistema operativo, independentemente do *hardware* do servidor ou do sistema operativo usado pela máquina virtual;
- Estabelecer uma primeira linha de defesa consistente para toda a infraestrutura TI;
- Proteger as aplicações sem quaisquer outras opções de *failover* e tornar a alta disponibilidade possível para as aplicações que poderiam, eventualmente, ter sido deixadas desprotegidas.

Por fim, resta referir que o vSphere HA é a mais utilizada em ambientes virtualizados, fornecendo a base para a criação de um ambiente altamente disponível, monitorizando as máquinas virtuais e os servidores em que elas correm.

O estudo desta ferramenta foi feito com base no *site* oficial do VMWare VSphere[1].

[1] http://www.vmware.com/products/datacenter-virtualization/vsphere/high-availability.html#glance
Consultado a 14/11/2012

C.2 Red Hat

A Red Hat dispõe de um pacote de aplicações designado **Red Hat Cluster Suite** para a implementação de *clusters* de alta disponibilidade e balanceamento de carga (utilizando o LVS - *Linux Virtual Server*).

Para além disso, permite a integração com o GFS para sistemas de armazenamento partilhados.

Este pacote inclui duas características de *clustering*. A principal é a alta disponibilidade, designada por *Cluster Manager*. Esta permite o funcionamento contínuo das aplicações no caso de uma falha num servidor. A segunda característica é o balanceamento de carga (originalmente chamado Piranha), fornecendo balanceamento ao nível da rede. Permite que um servidor *front-end* redireccione pacotes IP para um grupo de servidores de *backend* aumentando assim o desempenho global da rede.

O *Cluster Manager* fornece uma infraestrutura para aplicações que se enquadrem nas seguintes categorias:

- Aplicações que tolerem alguns segundos de *downtime*;
- Bases de Dados. O *Cluster Manager* é ideal para fornecer alta disponibilidade em bases de dados, incluindo Oracle 8i/9i, DB2, MySQL e a Red Hat *Database*;
- Sistemas de ficheiros heterogéneos como o NFS e SMB/CIFS (usando Samba);
- Aplicações comerciais *mainstream*. O *Cluster Manager* pode ser usado com aplicações como SAP, Oracle AS (*Application Server*) e Tuxedo;
- Internet e aplicações *OpenSource*. O *Cluster Manager* suporta a maioria das aplicações populares da Internet e *OpenSource* como por exemplo o Apache.
- Aplicações de mensagens como os servidores de mail Sendmail e o Domino da IBM.

O *Cluster Manager* fornece alta disponibilidade usando uma técnica amplamente usada por outros sistemas operativos que é a designada por *Application Failover*.

Como parte da implementação de um *cluster* moderno, o *Cluster Manager* foi desenvolvido especificamente para ser usado com o *hardware* comum, não requerendo componentes de *hardware* específicos ou diferentes dos que estamos habituados. No entanto,

em alguns casos pode ser adicionado *hardware* adicional para aumentar a disponibilidade como por exemplo uma UPS.

A configuração mais simples do *Cluster Manager* pode ser composto por apenas dois servidores e um subsistema externo de armazenamento SCSI ou *Fibre Channel*.

Podem ser configurados até oito servidores com o *Cluster Manager* ligados ao mesmo subsistema de armazenamento externo, permitindo-lhes aceder directamente aos discos partilhados. O *Cluster Manager* controla o acesso a partições de armazenamento de modo a que um servidor só possa aceder a uma partição num dado momento. Isto é necessário de forma a evitar acesso concorrente aos arquivos de dados.

Cada servidor passa a funcionar como se fosse um único sistema autónomo (*standalone*), correndo aplicações e acedendo aos dados alocados nas partições de armazenamento. Usando múltiplos servidores consegue-se escalar o *cluster*, adicionando poder de computação ao sistema.

Para além das ligações ao armazenamento partilhado, os servidores estão também ligados por rede conseguindo comunicar entre eles. Desta forma, caso um servidor falhe os outros servidores detectarão e automaticamente recomeçarão as aplicações que estavam a ser executadas pelo servidor que falhou. O servidor responsável por arrancar as aplicações e serviços do servidor que falhou pode ser seleccionado manualmente ou automaticamente pelo *Cluster Manager*. Uma vez que todos os servidores estão ligados ao armazenamento externo partilhado, os servidores operacionais podem aceder às partições do servidor que falhou e as suas aplicações podem continuar a ser executadas normalmente. Se for necessário, poderá haver um terceiro servidor que assume o IP do servidor que falhou de forma a que as operações de rede continuem sem interrupção. Em baixo mostra-se um exemplo deste tipo de configuração:

A característica fundamental de um *cluster Cluster Manager* é o facto de o armazenamento ser partilhado, permitindo que qualquer servidor possa ser hospedar uma aplicação e aceda directamente aos dados dessa aplicação. O *Cluster Manager* fornece alta disponibilidade para as aplicações gerindo redundâncias de serviços nos servidores. Contudo, de forma a tornar a configuração altamente disponível é necessário considerar outros componentes essenciais na configuração. Alguns *clusters* são configurados para não terem pontos únicos de falha (SPOF) pela inclusão de redundância a todos os níveis.

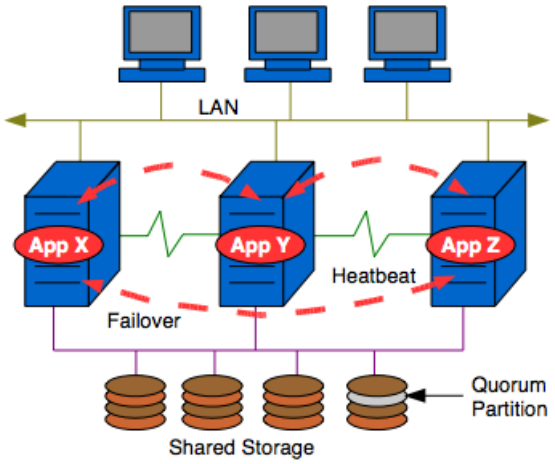

FIGURA C.1: Cluster Manager - Fonte: http://www.redhat.com/whitepapers/rha/RHA_ClusterSuiteWPPDF.pdf

Isto inclui geralmente UPSs e interfaces de rede redundantes. Também é importante saber que uma falha do armazenamento externo pode deitar o *cluster* a baixo, sendo vital usar também um sistema de armazenamento com alta disponibilidade. Isto inclui normalmente *dual controllers* para redundância e todo o armazenamento é configurado em RAID-1 (espelhamento) ou RAID-5.

O *Cluster Manager* e o balanceador de carga (Piranha) são duas tecnologias que se complementam e que podem ser usadas quer separadas, quer combinadas, dependendo dos requisitos de qualidade do *cluster*.

O balanceador de carga é baseado no projecto *OpenSource* LVS e fornece um conjunto de capacidades interessantes. Suporta a configuração de um balanceador de *backup* que controlará o balanceamento de carga no caso de o balanceador primário falhar. Adicionalmente, um balanceador de carga primário pode sondar os seus clientes IP de forma a assegurar que eles estão activos e rapidamente se ajustar a mudanças de estado de um cliente, sempre que entre ou saia do grupo de clientes em balanceamento de carga.

Na figura abaixo pode ser vista a configuração de um servidor Web num modelo *three tier*. A primeira camada dispõe de dois balanceadores de carga que distribuem pedidos Web para a segunda camada. Executando a aplicação Web dos clientes e usando principalmente dados estáticos, os servidores conseguem tratar de todos os pedidos Web. No entanto, para atender a pedidos que implicam a consulta da base de dados, a segunda camada reencaminhará os pedidos para a terceira camada do sistema.

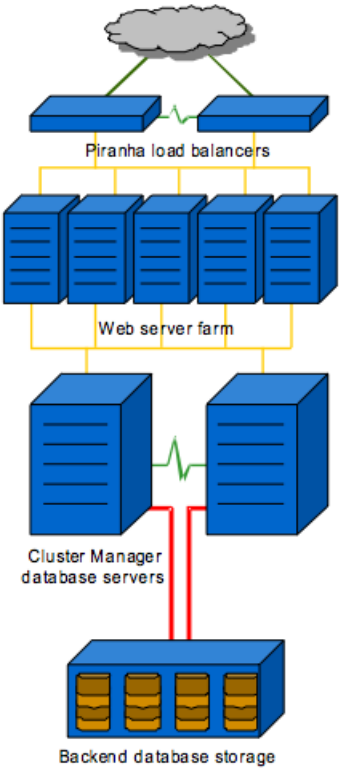

FIGURA C.2: Modelo *Three Tier* - Fonte: http://www.redhat.com/whitepapers/rha/RHA_ClusterSuiteWPPDF.pdf

O estudo desta ferramenta foi feito com base na documentação oficial da Red Hat[1].

[1] http://www.redhat.com/whitepapers/rha/RHA_ClusterSuiteWPPDF.pdf
Consultado a 14/11/2012

C.3 Oracle Solaris Cluster

A Oracle disponibiliza uma solução comercial de alta disponibilidade e recuperação de falhas para a *cloud*. Fornece uma infraestrutura que permite criar *clouds* privadas, públicas e híbridas bem como *datacenters*.

Características e Benefícios:

- Integração com o Kernel: Permite a detecção de falhas instantâneas do servidor e e recuperar aplicações de forma mais rápida e confiável, reduzindo o tempo de indisponibilidade ou *downtime*;
- *Disk Fencing*: Esta característica garante a integridade dos dados no caso de interrupções dos servidores, prevenindo que nós falhados tentem aceder ao armazenamento, corrompendo os dados;
- Quorum: Ajuda a prevenir a corrupção dos dados em cenários de falhas de servidores. Os dispositivos suportados incluem quorum baseado em disco, quorum baseado em *software* e servidor de quorum. Esta flexibilidade permite que os clientes criem a sua solução de quorum de acordo com a sua solução;
- Monitorização: A monitorização constante permite a prevenção de interrupções através da detecção antecipada de falhas. Todos os componentes do *Oracle Solaris Cluster* são monitorizados incluindo servidores, rede, discos, recursos de armazenamento e quorum. A disponibilidade dos sistemas de ficheiros usados pelas aplicações do *cluster* pode ser monitorizada e corrigida sempre que possível, através de diagnósticos de erro.
- Integração com o gestor de serviços da Oracle Solaris: Os administradores podem facilmente mover as suas aplicações de um ambiente de um único nó para um ambiente em *cluster* com múltiplos nós;
- Verificador de configuração: O verificador permite detectar configurações de *cluster* vulneráveis, de forma regular e rápida, prevenindo falhas no caso de existirem configurações erradas ao longo das máquinas do *cluster*.

O estudo desta ferramenta foi feito com base na documentação oficial da Oracle[1] [2].

[1] http://www.oracle.com/technetwork/server-storage/solaris-cluster/overview/features-cluster-166765.pdf?ssSourceSiteId=ocomen
Consultado a 14/11/2012

[2] http://www.oracle.com/us/products/servers-storage/solaris/cluster/overview/index.html
Consultado a 14/11/2012

C.4 Suse

A Suse dispõe de uma solução comercial de alta disponibilidade. Trata-se de uma extensão ao seu *Service Pack 2*.

Variedade de cenários de *clustering*

- Activo/Activo
- Activo/Passivo incluindo: N+1, N+M, N para 1 e N para M;
- *Clusters* híbridos, físicos e virtuais, permitindo que servidores virtuais se juntem a servidores físicos do *cluster*, melhorando a disponibilidade do serviço e a utilização de recursos;
- *Metro Clusters* ou locais;
- *Clusters* Multi-Site, geograficamente dispersos.

O *cluster* pode conter até 32 servidores Linux. Qualquer servidor pode reiniciar recursos (aplicações, serviços, endereços IP, e sistemas de ficheiros) a partir de um servidor que tenha falhado no *cluster*.

Flexibilidade

Esta extensão vem com o Corosync/OpenAIS (para suportar a comunicação entre os nós) e com o CRM Pacemaker. Usando o Pacemaker, os administradores podem monitorizar continuamente o estado dos recursos do *cluster*, gerir dependências e, automaticamente, iniciar e parar serviços com base em políticas e regras configuráveis. Permite ao utilizador a escolha das aplicações que deseja ter no seu *cluster* e decidir a infraestrutura de *hardware*. Com base em regras temporais, permite que os serviços migrem automaticamente para nós que tenham sido reparados, em momentos específicos.

Armazenamento e Replicação de Dados

Com esta extensão de alta disponibilidade o utilizador pode, dinamicamente, especificar um servidor de armazenamento sempre que for necessário. Suporta tecnologia Fibre Channel ou iSCSI. Os sistemas de partilha de disco também são suportados, mas não são um requisito. Esta extensão vem ainda com um sistema de ficheiros independente do

cluster, o OCFS2 e o gestor de volumes do *cluster*, o cLVM. Para a replicação dos dados pode ser usado o DRBD para espelhar os dados de um nó activo para o nó *standby*. Para além disso, esta extensão suporta o CTDB - *Clustered Trivial Database*, uma tecnologia para incorporar o Samba no *cluster*.

Suporte de ambientes virtualizados

Tanto suporta o *clustering* de servidores físicos como de servidores virtuais Linux. Este pacote vem com o Xen, que é um *software OpenSource* de supervisão da virtualização. Vem também com o KVM, um *software* de virtualização para Linux que é baseado em extensões de virtualização de *hardware*. O CRM é capaz de reconhecer, monitorizar e gerir serviços que estejam a correr tanto nos servidores virtuais, como nos servidores físicos.

Ferramentas de Administração *User Friendly*

Esta extensão vem com ferramentas poderosas que permitem não só a instalação e configuração do nosso *cluster* como também ferramentas eficientes de administração do mesmo:

- YaST - Trata-se de uma interface gráfica para a instalação e administração geral do sistema. Pode ser usada para instalar pacotes novos como também fornece módulos que ajudam a configurar o *cluster*.
- Pacemaker GUI - É uma interface gráfica para fácil configuração e administração do *cluster*. Guia o utilizador através da criação e configuração dos recursos e permite executar tarefas de inicialização, paragem e migração de recursos entre máquinas.
- HA Web Konsole (Hawk) - É uma interface *Web-based* com a qual podemos administrar o nosso *cluster* a partir de máquinas sem o sistema operativo Linux.
- *CRM Shell* - Permite, através de uma linha de comandos, configurar recursos e executar tarefas de monitorização e administração.

O estudo desta ferramenta foi feito com base na documentação oficial da Suse[1].

[1] https://www.suse.com/products/highavailability/
Consultado a 14/11/2012

C.5 Veritas Cluster Server

O Veritas Cluster Server é uma solução comercial que permite a construção de *clusters* Altamente Disponíveis sem qualquer intervenção manual, automatizando o processo de recuperação de uma falha num servidor eficientemente.

Apresenta como características:

- Automatização de *failover* no caso de falha em base de dados ou outra aplicação, quer num *datacenter* local ou em múltiplos *datacenters* remotos;
- Suporta vários sistemas operativos, quer físicos, quer virtuais;
- Permite ao administrador monitorizar, gerir e reportar eventos a partir de uma consola *web-based*;
- Permite um *failover* das aplicações sem a necessidade de reiniciar o sistema, de forma a garantir rapidez na recuperação dos serviços;
- Permite, através de uma *framework* de monitorização inteligente, a rápida deteção de falhas através da monitorização assíncrona dos nós. Desta forma, as falhas podem ser detectadas instantaneamente em vez de ter de esperar por uma resposta que não chega, de um recurso em falha;
- Permite efectuar testes de falha num *cluster* sem que para isso seja necessário desligá-lo.

O estudo desta ferramenta foi feito com base na documentação oficial do Veritas[1] [2].

[1] http://www.symantec.com/cluster-server
Consultado a 15/11/2012

[2] http://eval.symantec.com/mktginfo/enterprise/yellowbooks/b-cluster_file_system_tech_usage_20982271.en-us.pdf
Consultado a 15/11/2012

C.6 Microsoft's Cluster Server

É uma solução de *cluster* comercial que permite que vários servidores possam trabalhar em conjunto de forma a providenciar mecanismos de *failover*, aumentar a disponibilidade das aplicações e até ter processamento paralelo no caso de se querer ter um *cluster* de Alto Desempenho.

Este *software* permite a avaliação dos recursos dos servidores e proceder de forma a garantir a disponibilidade dos mesmos. Este serviço de *cluster* inclui componentes de *software* que permitem monitorizar e garantir a consistência e transferência de recursos de um nó para outro num *cluster*.

Algumas das características são:

- Gestor de Base de Dados que permite a configuração da base de dados do *cluster*;
- Gestão de nós, permitindo a comunicação entre os nós e do estado de cada um deles no *cluster*;
- *Failover* aquando a detecção de uma falha num nó, permitindo a transferência de serviços para outras máquinas;
- *Global Update Manager*, que permite que actualizações no *cluster* se façam atomicamente em todos os nós.

O estudo desta ferramenta foi feito com base na documentação oficial da Microsoft[1] [2].

[1] http://technet.microsoft.com/en-us/library/cc738051(v=ws.10).aspx
Consultado a 15/11/2012

[2] http://msdn.microsoft.com/en-us/library/ms952401.aspx
Consultado a 15/11/2012

C.7 HP Serviceguard

O HP Serviceguard é uma solução comercial para Linux orientada a aplicações críticas, que permite a continuidade dos serviços e a recuperação do sistema no caso de uma falha num servidor. Monitoriza as aplicações de forma a que, no caso da detecção de uma falha, possa automaticamente e de forma transparente recuperar dessas falhas sem comprometer a integridade dos dados e o desempenho do sistema.

Apresenta as seguintes características:

- Fornece um rápido *failover*, mesmo em grandes *clusters*, monitorizando a disponibilidade de recursos críticos e aplicações;
- Garante a integridade dos dados e o desempenho do *cluster*;
- Protege o *cluster* contra desastres e interrupções, permitindo-nos permanecer *online* mesmo depois de se perder um *datacenter*;
- Permite aos utilizadores visualizar as configurações, monitorizar, gerir e administrar um *cluster* e os seus componentes, nós e serviços;
- Reduz o tempo de *downtime* para zero no momento de fazer manutenção da infraestrutura do *cluster* através do *Live Application Detach*.

O estudo desta ferramenta foi feito com base na documentação oficial da HP[1].

[1] http://h20341.www2.hp.com/integrity/us/en/os/linux-serviceguard.html
Consultado a 16/11/2012

C.8 IBM HA/CMP

A IBM também apresenta uma solução comercial de alta disponibilidade, com o objectivo de fornecer a disponibilidade contínua das aplicações de um *cluster* quer no caso de interrupções planeadas, quer não. As aplicações críticas tipicamente envolvem, no mínimo, dois nós. O *cluster* é monitorizado de forma a perceber o estado dos nós e a perceber a consistência da configuração de todo o *cluster*. O *cluster* pode ser configurado para recuperar de falhas, possibilitando o uso de ferramentas como o PowerHA System-Mirror Standard Edition e o PowerHA System Mirror Enterprise Edition para a gestão do estado dos nós, partilha de sistemas de ficheiros entre nós, monitorização gráfica, alterações de configuração do *cluster*, entre outras funcionalidades, de forma a fornecer uma solução de alta disponibilidade robusta focada na facilidade de implementação e utilização. O estudo desta ferramenta foi feito com base na documentação oficial da IBM[1].

[1] http://www-03.ibm.com/systems/power/software/availability/aix/index.html
Consultado a 16/11/2012

C.9 Lifekeeper

O Lifekeeper é uma solução comercial que garante a alta disponibilidade em *clusters* com suporte para *failover*. Tem a vantagem de poder ser utilizado em sistemas Windows e em diferentes distribuições Linux.

Se ocorrer uma falha numa aplicação, ela será reiniciada ou transferida para um nó de *backup*. Todos os recursos necessários ao funcionamento dessa aplicação serão também transferidos para o nó de *backup*. Quaisquer outros serviços que se encontrem a funcionar correctamente permanecerão em execução na mesma máquina.

O ARK (*Application Recovery Kit*) é o componente que especifica qual a aplicação a iniciar, parar ou testar e que recursos são necessários para a aplicação executar correctamente. O ARK pode ser usado para aplicações de bases de dados, aplicações de mail, etc.

Num *cluster* Lifekeeper poderão ser executadas aplicações protegidas e não protegidas ao mesmo tempo. Depois de uma falha, as aplicações não protegidas deixarão de funcionar.

O Lifekeeper preocupa-se com a integridade dos dados. Desta forma, existem vários cenários que podem ser configurados para este efeito: replicação de dados, partilha de armazenamento e replicação com base em ligações WAN, tornando possível construir *clusters* de grandes dimensões com nós em diferentes localizações geográficas.

Os nós de um *cluster* Lifekeeper podem funcionar com *hardware* heterogéneo, não necessitando da mesma configuração.

O Lifekeeper usa prioridades de gestão de forma a permitir um conjunto vasto de cenários de *failover*:

- **N+1 *Failover***: O *cluster* inclui um nó de *backup* dedicado para dois ou mais nós primários. Na ocorrência de uma falha, uma ou mais aplicações poderão ser executadas neste nó.
- **N-Way *Failover***: Algumas aplicações protegidas num nó primário podem ser transferidas para nós de *backup* diferentes, depois da ocorrência de uma falha. Isto permite ter um bom desempenho depois de uma falha, ignorando eventuais conflitos entre as aplicações depois do *failover*.

- ***Failover* em Cascata**: Este cenário é útil quando queremos um elevado nível de disponibilidade. Isto garante que, na ocorrência de uma falha no servidor primário e no servidor de *backup*, exista um segundo *failover* para um segundo nó de *backup* impedindo as aplicações de deixar de funcionar. Esta opção pode ser usada para efectuar um *failover* para um *datacenter* remoto.

O estudo desta ferramenta foi feito com base na documentação oficial do Lifekeeper para Linux[1] [2].

[1] http://www.ha-cc.org/fileadmin/images/ComputerConcept/Downloads/PDF/LK_for_Linux/Lifekeeper_for_Linux.pdf
Consultado a 16/11/2012

[2] http://www.ha-cc.org/high_availability/components/application_availability/cluster/high_availability_cluster/steeleye_lifekeeper/structure_function/failover_scenarios/
Consultado a 16/11/2012

Anexo D

Soluções de Base de Dados

O FreeSWITCH suporta nativamente o SQLite e o PostgreSQL. No entanto, permite ligações a outras bases de dados via ODBC. Neste anexo, de forma a consubstanciar a escolha da empresa pelo PostgreSQL, são comparadas três soluções de base de dados *OpenSource* suportadas pelo *software* de telefonia IP FreeSWITCH.

D.1 SQLite

O SQLite é uma solução de base de dados diferente da maioria das soluções existentes e o seu objectivo principal é a simplicidade. Muitos utilizadores preferem-na por ser uma base de dados pequena e rápida. Para além disso, não necessita de qualquer instalação, configuração ou administração. Por ser simples, tem menos pontos onde pode falhar e, como consequência, é uma base de dados com um bom nível de confiabilidade e segurança.

Numa base de dados, a simplicidade de construção tanto pode trazer vantagens, como desvantagens. De forma a ser simples, o SQLite teve de se sacrificar noutros pontos que muitos consideram importantes para as suas soluções, como por exemplo a grande concorrência aos seus dados, acessos múltiplos, grandes quantidades de dados e escalabilidade. Trata-se de uma base de dados concebida para ser simples e que não foi desenhada para competir com bases de dados como o PostgreSQL ou o MySQL.

Apesar de não parecer uma boa solução, há algumas situações em que este sistema de base de dados se encaixa na perfeição:

- Dispositivos embebidos: uma vez que a base de dados SQLite não requer praticamente nenhuma administração, trata-se de uma boa escolha para dispositivos ou serviços que funcionem sem a intervenção humana, como por exemplo, telefones, PDAs, *set-top boxes*, entre outros.

- *Websites*: Para a grande maioria das necessidades dos *websites*, esta pode ser a melhor solução, já que consegue lidar com o tráfego gerado por estes. No entanto, dependerá de quantas requisições à base de dados um *website* fará;

- Em redes *ad hoc* são normalmente, no acesso a ficheiros do disco, usadas funções como fopen(), fread() e fwrite() para criar e gerir ficheiros de dados. Em substituição a estas funções, o SQLite funciona melhor, tendo um melhor desempenho;

- Também pode ser usada para bases de dados temporárias, aquando da necessidade de mover grandes quantidades de dados para fins de ordenamento, etc. Geralmente, usar SQLite é uma forma mais fácil para atingir este tipo de fins do que conseguir o mesmo efeito manualmente;

- Também é comum utilizar-se o SQLite para guardar registos estatísticos e análises métricas, uma vez que é muito fácil de configurar e de, caso se queira, possível de transferir via email, por exemplo.

Por outro lado, há situações onde não é aconselhável o uso do SQLite:

- Aplicações servidor-cliente: se tivermos vários clientes a acederem à mesma base de dados deveremos considerar outro tipo de solução, devido à latência associada à maioria dos sistemas de ficheiros, reduzindo em muito o desempenho da aplicação.

- *Websites*: o SQLite funciona muito bem como *backend* de um *website*. No entanto, se por necessidades de escalabilidade se tiver de separar a base de dados num outro servidor, deveremos optar por outro tipo de solução;

- Conjuntos de dados muito grandes: o SQLite está limitado a 2 Terabytes. Uma vez que esta base de dados é armazenada num ficheiro e como muitos sistemas de ficheiros limitam o tamanho máximo dos ficheiros, neste caso devemos optar por outra solução de base de dados;

- Alta concorrência: o SQLite usa *locks* de leitura e escrita em toda a base de dados armazenada num ficheiro. Isto significa que se um processo estiver a fazer uma leitura, todos os outros processos estão impedidos de executar qualquer escrita. Da mesma forma, se houver um processo a escrever na base de dados, nenhum dos outros processos poderá executar leituras. Normalmente, isto não é um problema porque o processo é rápido. No entanto, algumas aplicações requerem maiores níveis de concorrência e deverá ser escolhida outra ferramenta de base de dados.

O estudo desta ferramenta foi feito com base na documentação oficial do SQLite[1].

[1] http://www.sqlite.org/
Consultado a 12/11/2012

D.2 PostgreSQL e MySQL

O PostgreSQL e o MySQL são duas bases de dados parecidas. Ambas são relacionais e *OpenSource*. Contudo, a escolha de uma ou de outra depende dos requisitos da nossa solução e para que fins as vamos utilizar. Se pensarmos em vender a nossa aplicação e não quisermos ter custos associados ao licenciamento a que o MySQL obriga, então deveremos optar pelo PostgreSQL que não tem custos de licenciamento.

O PostgreSQL teve a sua origem em 1985 na Universidade da Califórnia em Berkeley. Trata-se de uma base de dados que é 100% *OpenSource* e desenvolvida e mantida por uma comunidade com milhares de contribuidores. Aparece-nos como uma versão final completa, ao contrário de outras como o MySQL em que temos várias versões de base de dados, algumas comerciais. O PostgreSQL apresenta uma licença livre que permite que as organizações a usem, copiem, modifiquem e distribuam apenas com o cuidado de deixar uma nota de *copyright*.

O PostgreSQL tem como alta prioridade a confiabilidade. Desta forma, foi concebida para ser uma base de dados sólida e bem construída, capaz de suportar aplicações críticas e com altas taxas de transacções. A documentação é também uma mais-valia, suportando múltiplos manuais livres *online*, como também dispondo do suporte oferecido pela comunidade. A consistência e integridade dos dados são também prioridades, detendo as propriedades ACID. Para além destas propriedades oferece uma forte segurança no controlo de acesso à base de dados, usando ferramentas de segurança como o Kerberos e o OpenSSL. Para além disso, podemos customizar a nossa base de dados de acordo com as regras que a nossa aplicação deverá comportar. Implementa características como PITR e alta disponibilidade, permitindo-nos criar um servidor *Warm Standby* para um rápido *failover*. Suporta *snapshots*, permitindo recuperações para pontos específicos no tempo. Disponibiliza ainda métodos de forma a conseguir alta disponibilidade, balanceamento de carga e replicação, permitindo-nos construir a solução que nos seja mais adequada.

O MySQL apareceu em 1994 e é conhecido como a base de dados *OpenSource* mais famosa do mundo. Foi originalmente concebida de forma a criar um *backend* rápido para servidores Web, através de um método de acesso rápido de indexação sequencial (ISAM) sem suporte às propriedades ACID. A partir daí, surgiram novas implementações como o suporte a um número adicional de *storage engines*. As propriedades ACID estão agora

disponíveis através da *engine* InnoDB, enquanto que outras capacidades como tabelas temporárias em memória estão disponíveis na *engine* MEMORY. Suporta também a *engine* MyISAM que oferece bastante rapidez da base de dados, no entanto perde na verificação da integridade dos dados. Isto é bom por exemplo para bases de dados de *sites* Web em que basicamente só suportam leituras de dados. Por outro lado, esta *engine* não é nada boa para bases de dados críticas, ficando as tabelas MyISAM corrompidas muito facilmente. Desta forma, a *engine* InnoDB, com as propriedades ACID, é a melhor escolha. O PostgreSQL tem apenas uma única *engine* que nos permite através de um ficheiro de configuração, alterar parâmetros que afectam a *performance* da base de dados.

Ambas as soluções podem ser configuradas de forma a optimizar a *performance* da base de dados, suportando extensões para outras funcionalidades.

A documentação do MySQL é bastante grande e conta com manuais livres, alguns livros, artigos *online* e também oferece suporte através de fornecedores como por exemplo a Oracle.

O MySQL foi vendido à Sun Microsystems que, por sua vez, foi comprada pela Oracle em 2010. A Oracle suporta várias edições como a *Standard*, a *Enterprise*, a *Classic*, a *Embedded* e a *Community*. Algumas destas são livres enquanto que outras são pagas. Actualmente existem mais escolhas para base de dados baseadas no código original do MySQL como a MariaDB, Drizzie, entre outros.

Tanto o PostgreSQL como o MySQL são ferramentas que podem ser executadas em múltiplos sistemas operativos como Linux, Unix, Mac OS X e Windows. Ambos são *software OpenSource*, gratuitos, flexíveis e escalam bem em sistemas de grandes dimensões. O PostgreSQL não suporta aplicações de SQL embebidas, ao contrário do MySQL. O MySQL é entendido como uma base de dados rápida de *backend* para *sites* Web e aplicações, conseguindo executar leituras rápidas e executar um grande número de *queries* de pequena dimensão. No entanto, oferece características menos sofisticadas e também menos verificações de integridade de dados. O PostgreSQL é conhecido por ser muito completo, respeitar fortemente as propriedades ACID e pelas verificações de integridade dos dados.

Tanto o PostgreSQL como o MySQL visam os *standards* SQL, apesar de o MySQL suportar as suas próprias extensões. O facto de seguirem os *standards* facilita o trabalho

aos administradores e utilizadores de bases de dados que apenas têm de aprender determinado *standard*, as suas características e os comandos associados, sendo o código portável, diminuindo algum tempo e esforço. Por outro lado, os *standards* levam o seu tempo a evoluir, limitando os utilizadores às regras desses *standards* e desencorajando os que desejam lançar novas características que dependem dos *standards*, adoptando soluções *non-compliant*.

Concluindo, uma vez que suporta vários modos SQL, como o ANSI, o MySQL revela-se mais complexo. No entanto é mais flexível, oferecendo mais opções para a adaptação da base de dados a diferentes cenários, através das *engines*. O PostgreSQL acaba por ser uma solução importante pela confiabilidade, integridade e protecção dos dados, sendo um projecto de comunidade, não dependendo das decisões de terceiros.

O estudo desta ferramenta foi feito com base na documentação oficial do PostgreSQL[1] e do MySQL [2].

[1] http://www.postgresql.org/
Consultado a 12/11/2012
[2] http://www.mysql.com/
Consultado a 12/11/2012

Anexo E

Requisitos Funcionais e Não-Funcionais

Este anexo pretende listar e familiarizar o leitor com todas os requisitos presentes na plataforma de telefonia IP, oferecendo-lhe uma perspectiva global e detalhada das funcionalidades existentes na mesma.

E.1 Requisitos Funcionais

O sistema deverá permitir o utilizador:

- Efectuar chamadas através da Internet quer via *softphones* ou *hardphones* IP;
- Terminar sessão;
- Editar os seus dados de conta, como o *username* e a *password*;
- Criar uma ligação IMAP de forma a ter uma conta de email associada, usada em aplicações como a de Fax;
- Dar permissões de visualização de determinadas aplicações a diferentes tipos de utilizadores, com base em grupos;
- Inicializar e parar os diferentes módulos do *FreeSWITCH*;

- Alterar os parâmetros de configuração do *event socket* que permite a ligação da Web ao *FreeSWITCH* e executar comandos como por exemplo fazer chamadas, transferir chamadas, etc.;
- Criar ou editar variáveis do *FreeSWITCH* sem ser necessário ter de editar os ficheiros de configuração do *FreeSWITCH*, como por exemplo o porto externo e o IP da máquina;
- Criar extensões e associá-las quer a *softphones*, quer a *hardphones* IP;
- Criar *gateways* PRI, BRI ou quaisquer outros *trunks* VoIP, registados ou não em operadoras;
- Visualizar que extensões é que estão criadas;
- Visualizar se as extensões estão ou não registadas;
- Atribuir números mais fáceis de decorar - *Alias*;
- Criar, editar e apagar utilizadores;
- Associar utilizadores a extensões;
- Adicionar várias extensões a um *call group*;
- Atribuir um DDI a uma extensão;
- Adicionar um caixa de entrada de email, na qual será entregue o *voicemail*;
- Criar *dialplans*, permitindo definir regras que o sistema deve executar quando chega uma chamada ou quando é efectuada uma chamada para um número específico. Exemplos: 1) Quando alguém liga para a extensão 665 a chamada deverá tocar em todos os telefones dos trabalhadores do departamento VoIP; 2) Quando um trabalhador recebe uma chamada mas não se encontra disponível para atender, podemos puxar a chamada dele para o nosso telefone marcando ** seguido do número da sua extensão;
- Atribuir um DDI a uma extensão;
- Definir rotas de entrada. Exemplo: 1) Se um cliente ligar para o 707999005 (número de suporte), a chamada bate no nosso sistema e segue uma regra de entrada que lhe diz para chamar todos os telefones do departamento VoIP; 2) Se

um cliente ligar para o 309700221 a chamada é atendida por um IVR que oferece duas opções: a primeira transfere a chamada para o departamento de suporte e a segunda transfere a chamada para a operadora;

- Associar um DDI a um *gateway*;
- Definir regras de saída das chamadas. Exemplo: o limite máximo de chamadas internacionais por dia é 40 e o número máximo de minutos por dia é 120;
- Definir rotas de saída. Exemplo: 1) Alguém de dentro liga para fora (telemóvel ou número fixo). De acordo com prioridades a chamada sai por um determinado *trunk* ou por outro, dependendo dos tarifários que a empresa tenha com as operadoras;
- Criar uma lista de *call broadcast*. Útil, por exemplo, quando queremos convidar pessoas para uma conferência;
- Criar uma fila de *call center* associada a uma extensão. Através de um *dialplan*, ao ligar-se para essa extensão, a chamada vai chegar às pessoas (agentes) que tiverem telefones registados nesta fila, de acordo com uma estratégia definida (tocar em todos os telefones, *round robin*, por ordem, etc.);
- Visualizar o registo e consultar os dados das chamadas efectuadas;
- Criar conferências;
- Criar uma conta para o envio e recepção de faxes;
- Criar um *huntgroup* associado a uma extensão. Ao chamar essa extensão, dependendo da ordem, a chamada irá tocar em vários telefones;
- Criar menus IVR;
- Visualizar o painel da operadora que permite efectuar, atender e transferir chamadas, para além de permitir visualizar as chamadas em tempo-real no sistema;
- Visualizar e carregar ficheiros de som para o sistema, a serem utilizados em IVRs, por exemplo;
- Criar ficheiros .wav *text-to-speech*;
- Criar *time conditions*. Quando alguém ligar para um determinado número, dependendo da hora a que a chamada é efectuada, esta é tratada da forma mais indicada,

ou com uma mensagem de boas vindas, ou uma mensagem a indicar que o período de actividade da empresa é das 9h às 18h e que estão fechados, por exemplo;

- Visualizar os *call centers* que estão activos;
- Visualizar as chamadas activas no sistema em tempo-real;
- Visualizar as conferências a decorrer em tempo-real;
- Visualizar as extensões activas no sistema;
- Visualizar as filas existentes no sistema para usar nos *call centers*;
- Listar todos os agentes pertencentes aos *call centers* e o seu estado no sistema (disponível ou indisponível);
- Visualizar os *logs* do serviço *FreeSWITCH*;
- Iniciar e parar serviços do sistema operativo usado no sistema de telefonia IP - linux;
- Visualizar o estado do módulo sofia do *FreeSWITCH*;
- Visualizar o tráfego a passar nas interfaces de rede especificadas;
- Abrir o gestor de base de dados do Postgres (aplicação adminer);
- Listar as aplicações existentes na plataforma WEB, as respectivas versões e descrições;
- Executar comandos *shell* via WEB e comandos PHP;
- Listar, adicionar e modificar as bases de dados do sistema e os respectivos dados de autenticação/configuração;
- Listar, adicionar e modificar as variáveis por omissão do sistema;
- Listar, adicionar e modificar domínios do sistema;
- Listar, adicionar e modificar grupos de utilizadores com permissões no sistema;
- Listar, adicionar e modificar os perfis SIP do sistema de telefonia IP;
- Executar *queries* à base de dados via WEB;

- Fazer *upgrade* ao *schema* da base de dados. Quando se alteram, eliminam ou adicionam dados à base de dados, esta ferramenta é útil para actualizar os dados na plataforma WEB;
- Listar ou modificar dados do domínio, como o número de salas de conferência disponíveis e o número de utilizadores máximo;
- Entrar no painel de conferências e visualizar que reuniões existem e o seu estado respectivo. No caso de ter permissões, poderá adicionar mais reuniões e/ou editar as existentes;
- Aprovisionar domínios, extensões e *gateways*.

E.2 Requisitos Não-Funcionais

- O sistema deverá garantir a integridade e consistência através da replicação da base de dados entre os dois servidores. Desta forma, quer um cliente seja tratado por um servidor ou por outro, verá os mesmos resultados;
- No caso de uma falha, o sistema deverá recuperar e disponibilizar todos os serviços da máquina que falhou (serviço Web FsCloud, base de dados e FreeSWITCH);
- O sistema deverá escalar. O número máximo de chamadas em simultâneo a definir não deverá implicar a degradação no desempenho das chamadas em curso nem dos servidores;
- Segurança/Confidencialidade: os servidores deverão usar encriptação nas suas ligações. A sessão SIP dos clientes deverá ser encriptada com TLS, enquanto que a *media* RTP deverá ser encriptada via SRTP.

Anexo F

Máquina Cliente - Configuração das Ferramentas de Monitorização e DNS

De forma a monitorizar os servidores do *cluster* de alta disponibilidade foram escolhidas as ferramentas Nagios, MRTG e VoipMonitor. O Nagios é uma ferramenta que permite a monitorização de serviços e portos nos servidores, notificando via email ou *sms* um administrador no caso da ocorrência de falhas. Disponibiliza uma interface WEB que facilita a monitorização ao administrador. Por sua vez, o MRTG permite a monitorização do tráfego nas interfaces de rede de um servidor, *router* ou *switch*, bem como de quaisquer outros dados que os dispositivos permitam aceder. Ambas as ferramentas são *software* livre e recolhem os seus dados via SNMP. O VoipMonitor foi a ferramenta utilizada para medir a qualidade de voz nos servidores-alvo quando sujeitos a testes de *stress*, através dos *scripts* lua desenvolvidos. Esta ferramenta utiliza o Wireshark como *sniffer* de rede e utiliza as capturas de tráfego para, com base em estatística e algoritmia, apresentar numa interface WEB resultados, gráficos e relatórios do tráfego gerado. Este servidor-cliente foi também utilizado para funcionar como DNS no cenário Activo/Activo em que houve necessidade de balancear a carga entre os dois servidores-alvo.

F.1 Configuração inicial da máquina

Configuração da interface de rede

```
vi /etc/sysconfig/network-scripts/ifcfg-eth0
        DEVICE="eth0"
        NM_CONTROLLED="yes"
        ONBOOT="yes"
        BOOTPROTO=dhcp

service network restart
```

Desactivar selinux e serviço iptables

```
service iptables stop
chkconfig iptables off
vi /etc/sysconfig/selinux
        SELINUX=disabled
setenforce 0
```

Instalação de alguns pacotes úteis à configuração da solução

```
yum install wget
yum install vim
yum install screen
yum install httpd
yum install php
chkconfig httpd on
```

F.2 Instalação do Nagios

Criação de um utilizador e grupo para o Nagios

```
adduser nagios
passwd nagios
mkdir /usr/local/nagios
chown nagios.nagios /usr/local/nagios
grep "\^User" /etc/httpd/conf/httpd.conf
groupadd nagcmd
usermod -G nagcmd nagios
usermod -G nagcmd apache
```

Instalação do Nagios

```
yum install gcc
yum install make
wget http://prdownloads.sourceforge.net/sourceforge/nagios/nagios-3.4.4.tar.gz
tar zxvf nagios-3.4.4.tar.gz
cd nagios
./configure --prefix=/usr/local/nagios --with-command-group=nagcmd
make all
make install
make install-init
make install-config
make install-commandmode
chkconfig nagios on
```

Configuração do caminho para o Nagios através do serviço Apache

```
# Na directoria de instalac o do nagios
make install-webconf
```

Criação de um utilizador para acesso ao Nagios

```
htpasswd -c /usr/local/nagios/etc/htpasswd.users nagiosadmin
Nota:  username: nagiosadmin password: wavecom
```

Aceder ao Nagios com pedido de autenticação

```
/etc/init.d/httpd start
/etc/init.d/nagios start
# apontar o browser para http://localhost/nagios/
```

Adicionar *hostnames* dos servidores-alvo

```
vi /etc/hosts/
        10.0.0.204              voipmonitor
        10.0.0.205              fs01-a
        10.0.0.206              fs01-b
```

Instalação do serviço SNMP (servidor-cliente e servidores-alvo)

```
yum install net-snmp
yum install net-snmp-perl
yum install net-snmp-utils
chkconfig snmpd on
```

Edição do ficheiro snmpd.conf (servidor-cliente e servidores-alvo)

```
vi /etc/snmp/snmpd.conf
        com2sec local   localhost       public
        com2sec mynetwork       10.0.0.0/24     public

        group MyRWGroup v1      local
        group MyRWGroup v2c     local
        group MyRWGroup usm     local
        group MyROGroup v1      mynetwork
        group MyROGroup v2c     mynetwork
        group MyROGroup usm     mynetwork

        view    systemview    included   .1.3.6.1.2.1.1
        view    systemview    included   .1.3.6.1.2.1.25.1.1

        access  notConfigGroup ""      any       noauth    exact  systemview none none

        view all    included  .1                               80

        access MyROGroup ""     any     noauth  exact   all     none    none
        access MyRWGroup ""     any     noauth  exact   all     all     none

        syslocation Wavecom
        syscontact dsanto@wavecom.pt

        dontLogTCPWrappersConnects yes
```

Testes ao SNMP - Verificar as MIBS disponíveis num determinado servidor

```
service snmpd restart
snmpwalk localhost -c public -v1
snmpwalk -v1 -c public voipmonitor IP-MIB::ipAdEntIfIndex
snmpwalk -v1 -c public fs01-a UCD-SNMP-MIB::ssCpuRawUser
snmpgetnext -v1 -c public fs01-a UCD-SNMP-MIB::ssCpuRawUser
snmpwalk -v1 -c public voipmonitor HOST-RESOURCES-MIB::hrStorageSize
snmpgetnext -v1 -c public voipmonitor HOST-RESOURCES-MIB::hrStorageSize
```

Testes ao SNMP (acesso remoto)

```
snmpwalk -v1 -c public fs01-a
snmpwalk -v1 -c public fs01-a IP-MIB::ipAdEntIfIndex
snmpwalk -v1 -c public fs01-a UCD-SNMP-MIB::ssCpuRawUser
snmpwalk -v1 -c public fs01-a HOST-RESOURCES-MIB::hrStorageSize
snmpgetnext -v1 -c public fs01-a HOST-RESOURCES-MIB::hrStorageSize
```

Nota: Para estes comandos funcionarem a *firewall* deverá estar desligada nos servidores remotos ou deverão ser criadas regras de *firewall* específicas.

F.3 Instalação do MRTG

Configuração do MRTG

```
yum install mrtg
mkdir -p /var/www/html/mrtg/
cfgmaker --output=/etc/mrtg/fs01-a.cfg --ifdesc=ip \
--ifref=descr --global 'WorkDir: /var/www/html/mrtg/' public@10.0.0.205

cd /usr/bin/
LANG=C mrtg /etc/mrtg/fs01-a.cfg (executar varias vezes se for preciso)
crontab -e
                */5 * * * * env LANG=C /usr/bin/mrtg /etc/mrtg/fs01-a.cfg \
                --logging  /var/log/mrtg.log

        indexmaker --output=/var/www/html/mrtg/index.html --title="Wavecom \
        - Alta Disponibilidade" --sort=name --enumerate /etc/mrtg/fs01-a.cfg

                mrtg /etc/mrtg/mrtg.cfg

                htpasswd -c /var/www/mrtg/htpasswd.users mrtgadmin (pass:wavecom)

                cp /var/www/html/mrtg/* /var/www/mrtg/
```

Configuração de acesso ao MRTG e Nagios

```
vi /etc/httpd/conf/httpd.conf

        ServerName localhost

        # Por quest es de seguranca pretende-se autenticacao basica para o
        acesso a informacao disponibilidade pelo Nagios e MRTG
        # Access MRTG Service
        <Location /mrtg>
            AuthName "Mrtg Access"
            AuthType Basic
            AuthUserFile /var/www/mrtg/htpasswd.users
            Require valid-user
            Allow from all
        </Location>

        # Access NAGIOS Service
        <Location /nagios>
            AuthName "Nagios Access"
            AuthType Basic
            AuthUserFile /usr/local/nagios/etc/htpasswd.users
            Require valid-user
            Allow from all
```

```
        </Location>

        # Access http://localhost/mymrtg

        service snmpd stop
        service httpd stop
```

MRTG - fs01-a.cfg

```
service snmpd stop
service httpd stop

vi /etc/mrtg/fs01-a.cfg

# Por baixo da directiva "WorkDir: /var/www/html/mymrtg/"
        LoadMIBs: /usr/share/snmp/mibs/UCD-SNMP-MIB.txt,/usr/share/snmp/mibs/ \
        HOST-RESOURCES-MIB.txt

### CPU USAGE FreeSWITCH 1
Target[linux_server_cpu_load1]: ( ssCpuRawUser.0&ssCpuRawUser.0:public@10.0.0.205 ) + \
( ssCpuRawSystem.0&ssCpuRawSystem.0:public@10.0.0.205  ) + \
( ssCpuRawNice.0&ssCpuRawNice.0:public@10.0.0.205  )
Title[linux_server_cpu_load1]: CPU Utilization for FreeSWITCH 1
PageTop[linux_server_cpu_load1]: <H1> CPU Utilization for FreeSWITCH 1 </H1>
MaxBytes[linux_server_cpu_load1]: 100
ShortLegend[linux_server_cpu_load1]: %
YLegend[linux_server_cpu_load1]: % of CPU Load
Legend1[linux_server_cpu_load1]: CPU Utilization
LegendI[linux_server_cpu_load1]: CPU Load
LegendO[linux_server_cpu_load1]:
Options[linux_server_cpu_load1]: nopercent
Unscaled[linux_server_cpu_load1]: ymwd

### MEMORY USAGE FreeSWITCH 1
Target[linux_server_mem_ram1]: ( memTotalReal.0&memTotalReal.0:public@10.0.0.205 - \
 memAvailReal.0&memAvailReal.0:public@10.0.0.205 )* 100 /  \
 ( memTotalReal.0&memTotalReal.0:public@10.0.0.205  )
Title[linux_server_mem_ram1]: Memory Utilization for FreeSWITCH 1
PageTop[linux_server_mem_ram1]: <H1> Memory Utilization for FreeSWITCH 1 </H1>
MaxBytes[linux_server_mem_ram1]: 100
ShortLegend[linux_server_mem_ram1]: %
YLegend[linux_server_mem_ram1]: % of Memory Load
Legend1[linux_server_mem_ram1]: Memory Utilization
LegendI[linux_server_mem_ram1]: Memory Load
LegendO[linux_server_mem_ram1]:
Options[linux_server_mem_ram1]: nopercent
Unscaled[linux_server_mem_ram1]: ymwd
```

```
### CPU USAGE FreeSWITCH 2
Target[linux_server_cpu_load2]: ( ssCpuRawUser.0&ssCpuRawUser.0:public@10.0.0.239 ) + \
( ssCpuRawSystem.0&ssCpuRawSystem.0:public@10.0.0.239  ) + \
( ssCpuRawNice.0&ssCpuRawNice.0:public@10.0.0.239  )
Title[linux_server_cpu_load2]: CPU Utilization for FreeSWITCH 2
PageTop[linux_server_cpu_load2]: <H1> CPU Utilization for FreeSWITCH 2 </H1>
MaxBytes[linux_server_cpu_load2]: 100
ShortLegend[linux_server_cpu_load2]: %
YLegend[linux_server_cpu_load2]: % of CPU Load
Legend1[linux_server_cpu_load2]: CPU Utilization
LegendI[linux_server_cpu_load2]: CPU Load
LegendO[linux_server_cpu_load2]:
Options[linux_server_cpu_load2]: nopercent
Unscaled[linux_server_cpu_load2]: ymwd

### MEMORY USAGE FreeSWITCH 2
Target[linux_server_mem_ram2]: ( memTotalReal.0&memTotalReal.0:public@10.0.0.239 - \
 memAvailReal.0&memAvailReal.0:public@10.0.0.239 )* 100 /  \
 ( memTotalReal.0&memTotalReal.0:public@10.0.0.239  )
Title[linux_server_mem_ram2]: Memory Utilization for FreeSWITCH 2
PageTop[linux_server_mem_ram2]: <H1> Memory Utilization for FreeSWITCH 2 </H1>
MaxBytes[linux_server_mem_ram2]: 100
ShortLegend[linux_server_mem_ram2]: %
YLegend[linux_server_mem_ram2]: % of Memory Load
Legend1[linux_server_mem_ram2]: Memory Utilization
LegendI[linux_server_mem_ram2]: Memory Load
LegendO[linux_server_mem_ram2]:
Options[linux_server_mem_ram2]: nopercent
Unscaled[linux_server_mem_ram2]: ymwd
```

Criação da página final com os dados a obter

```
cd /usr/bin
LANG=C mrtg /etc/mrtg/mrtg.cfg
indexmaker --output=/var/www/html/mrtg/index.html --title="Wavecom - Alta Disponibilidade" \
 --sort=name --enumerate /etc/mrtg/fs01-a.cfg
cp /var/www/html/mrtg/* /var/www/mrtg/
service snmpd start
service httpd start
```

Nota: verificar os novos ficheiros criados em /var/www/html/mrtg

Testar variação de valores nos gráficos gerados pelo MRTG

Para testar a oscilação de valores nos gráficos gerados pelo MRTG criou-se um *script* em *bash* e executou-se no servidor remoto (fs01-a). Este *script* permitiu ver que os dados obtidos por SNMP estão correctos e são fidedignos.

O *script* segue-se em baixo:

```
#!/bin/bash
while [ 1 ]; do
        tar -cvjf teste.tar.bz2 /var/log/
done
```

A página ficou acessível em:

- http://localhost/mymrtg

```
service httpd stop
service snmpd stop
service nagios stop
```

Configuração do Nagios - nagios.cfg

```
vim /usr/local/nagios/etc/nagios.cfg

##############################################################################
#
# NAGIOS.CFG - Sample Main Config File for Nagios 3.4.4
#
# Read the documentation for more information on this configuration
# file.  I've provided some comments here, but things may not be so
# clear without further explanation.
#
# Last Modified: 12-14-2008
#
##############################################################################



# LOG FILE
# This is the main log file where service and host events are logged
# for historical purposes.  This should be the first option specified
# in the config file!!!

log_file=/usr/local/nagios/var/nagios.log
```

```
# OBJECT CONFIGURATION FILE(S)
# These are the object configuration files in which you define hosts,
# host groups, contacts, contact groups, services, etc.
# You can split your object definitions across several config files
# if you wish (as shown below), or keep them all in a single config file.

# You can specify individual object config files as shown below:
cfg_file=/usr/local/nagios/etc/objects/commands.cfg
cfg_file=/usr/local/nagios/etc/objects/contacts.cfg
cfg_file=/usr/local/nagios/etc/objects/timeperiods.cfg
cfg_file=/usr/local/nagios/etc/objects/templates.cfg

# Definitions for monitoring the local (Linux) host
#cfg_file=/usr/local/nagios/etc/objects/localhost.cfg
cfg_file=/usr/local/nagios/etc/objects/fs01-a.cfg
cfg_file=/usr/local/nagios/etc/objects/fs01-b.cfg
```

Configuração do Nagios - fs01-a.cfg

```
##############################################################################
##############################################################################
#
# HOST DEFINITION
#
##############################################################################
##############################################################################

# Define a host for the local machine

define host{
        use                     linux-server            ; Name of host template to use
                                                        ; This host definition will inherit all
                                                        ; in (or inherited by) the linux-server
        host_name               fs01-a
        alias                   fs01-a
        address                 10.0.0.205
        }



##############################################################################
##############################################################################
#
# HOST GROUP DEFINITION
#
##############################################################################
##############################################################################
```

```
# Define an optional hostgroup for Linux machines

define hostgroup{
        hostgroup_name  instances ; The name of the hostgroup
        alias           instances ; Long name of the group
        members         fs01-a     ; Comma separated list of hosts that belong to this group
        }



##############################################################################
##############################################################################
#
# SERVICE DEFINITIONS
#
##############################################################################
##############################################################################


# Define a service to "ping" the local machine

define service{
        use                             generic-service         ; Name of service template to us
        host_name                       fs01-a
        service_description             PING
        check_command                   check_ping!100.0,20%!500.0,60%
        }


# Define a service to check the disk space of the root partition
# on the local machine.  Warning if < 20% free, critical if
# < 10% free space on partition.

define service{
        use                             generic-service         ; Name of service template to us
        host_name                       fs01-a
        service_description             RAM Usage
        check_command                   check_snmp!-C public -o HOST-RESOURCES- \
         MIB::hrStorageUsed.1 -m RFC1213-MIB -u "KB"
        }

define service{
        use                             generic-service         ; Name of service template to us
        host_name                       fs01-a
        service_description             Disk Usage
        check_command                   check_snmp!-C public -o HOST-RESOURCES- \
         MIB::hrStorageUsed.31 -m RFC1213-MIB -u "*4096B"
        }
```

```
define service{
        use                             generic-service         ; Name of service template to us
        host_name                       fs01-a
        service_description             CPU LOAD
        check_command                   check_snmp!-C public -o HOST-RESOURCES- \
         MIB::hrProcessorLoad.768 -m RFC1213-MIB -u "%"
        }

# Alternative
#define service{
#        use                             generic-service         ; Name of service template to u
#        host_name                       fs01-a
#        service_description             CPU LOAD Alternative
#        check_command                   check_load
#       }


define service{
        use                             generic-service         ; Name of service template to us
        host_name                       fs01-a
        service_description             SSH
        check_command                   check_ssh
        }

define service{
        use                             generic-service         ; Name of service template to us
        host_name                       fs01-a
        service_description             POP3S
        check_command                   check_tcp! 995
        }

define service{
        use                             generic-service         ; Name of service template to us
        host_name                       fs01-a
        service_description             IMAPS
        check_command                   check_tcp! 993
        }

define service{
        use                             generic-service         ; Name of service template to us
        host_name                       fs01-a
        service_description             PostgreSQL
        check_command                   check_tcp! 5432
        }

#define service{
#        use                             generic-service         ; Name of service template to u
```

```
#        host_name                       fs01-a
#        service_description             nrpe_check_pgsql
#        check_command                   check_nrpe2!check_pgsql
#        }



#define service{
#        use                             generic-service         ; Name of service template to u
#        host_name                       fs01-a
#        service_description             HTTPS
#       check_command                   check_tcp! 443
#        }

# HTTP
define service{
        use                             generic-service         ; Name of service template to us
        host_name                       fs01-a
        service_description             HTTP
        check_command                   check_http
        }


# HTTPS
define service{
        use                             generic-service         ; Name of service template to us
        host_name                       fs01-a
        service_description             HTTPS
        check_command                   check_https
        }
```

Configuração do Nagios - contacts.cfg

```
service httpd stop
service snmpd stop
service nagios stop
vi /usr/local/nagios/etc/contacts.cfg
        define contact{
        contact_name                    nagiosadmin             ; Short name of user
        use                             generic-contact         ; Inherit default values
         from generic-contact template (defined above)
        alias                           Nagios Admin            ; Full name of user

        email                           dsanto@wavecom.pt; <<***** CHANGE THIS TO YOUR
        EMAIL ADDRESS ******
        }

   define contact{
```

```
        contact_name                    nagiosgw             ; Short name of user
        use                             generic-contact         ; Inherit default values
         from generic-contact template (defined above)
        alias                           Nagios Admin            ; Full name of user

        email                           smsgw@wavecom.pt; <<***** CHANGE THIS TO YOUR
         EMAIL ADDRESS ******
        }

   define contactgroup{
        contactgroup_name       admins
        alias                   Nagios Administrators
        members                 nagiosadmin nagiosgw
        }
```

Configuração do Nagios - commands.cfg

```
##############################################################################
# COMMANDS.CFG - SAMPLE COMMAND DEFINITIONS FOR NAGIOS 3.4.4
#
# Last Modified: 05-31-2007
#
# NOTES: This config file provides you with some example command definitions
#        that you can reference in host, service, and contact definitions.
#
#        You don't need to keep commands in a separate file from your other
#        object definitions.  This has been done just to make things easier to
#        understand.
#
##############################################################################


##############################################################################
#
# SAMPLE NOTIFICATION COMMANDS
#
# These are some example notification commands.  They may or may not work on
# your system without modification.  As an example, some systems will require
# you to use "/usr/bin/mailx" instead of "/usr/bin/mail" in the commands below.
#
##############################################################################


# 'notify-host-by-email' command definition
define command{
        command_name    notify-host-by-email
        command_line    /usr/bin/printf "%b" "***** Nagios *****\n\nNotification \
        Type: $NOTIFICATIONTYPE$\nHost: $HOSTNAME$\nState: \
```

```
         $HOSTSTATE$\nAddress: $HOSTADDRESS$\nInfo: \
          $HOSTOUTPUT$\n\nDate/Time: $LONGDATETIME$\n" | \
          /bin/mail -s "** $NOTIFICATIONTYPE$ Host Alert: $HOSTNAME$ is \
           $HOSTSTATE$ **" $CONTACTEMAIL$
        }

# 'notify-service-by-email' command definition
define command{
        command_name    notify-service-by-email
        command_line    /usr/bin/printf "%b" "***** Nagios *****\n\nNotification \
        Type: $NOTIFICATIONTYPE$\n\nService: $SERVICEDESC$\nHost: \
         $HOSTALIAS$\nAddress: $HOSTADDRESS$\nState: \
          $SERVICESTATE$\n\nDate/Time: $LONGDATETIME$\n\nAdditional Info: \
          \n\n$SERVICEOUTPUT$\n" | /bin/mail -s "** $NOTIFICATIONTYPE$ Service \
           Alert: $HOSTALIAS$/$SERVICEDESC$ is $SERVICESTATE$ **" $CONTACTEMAIL$
        }




##############################################################################
#
# SAMPLE HOST CHECK COMMANDS
#
##############################################################################


# This command checks to see if a host is "alive" by pinging it
# The check must result in a 100% packet loss or 5 second (5000ms) round trip
# average time to produce a critical error.
# Note: Five ICMP echo packets are sent (determined by the '-p 5' argument)

# 'check-host-alive' command definition
define command{
        command_name    check-host-alive
        command_line    $USER1$/check_ping -H $HOSTADDRESS$ -w 3000.0,80% -c 5000.0,100% -p 5
        }




##############################################################################
#
# SAMPLE SERVICE CHECK COMMANDS
#
# These are some example service check commands.  They may or may not work on
# your system, as they must be modified for your plugins.  See the HTML
```

```
# documentation on the plugins for examples of how to configure command definitions.
#
# NOTE:  The following 'check_local_...' functions are designed to monitor
#        various metrics on the host that Nagios is running on (i.e. this one).
################################################################################

# 'check_local_disk' command definition
define command{
        command_name    check_local_disk
        command_line    $USER1$/check_disk -w $ARG1$ -c $ARG2$ -p $ARG3$
        }


# 'check_local_load' command definition
define command{
        command_name    check_local_load
        command_line    $USER1$/check_load -w $ARG1$ -c $ARG2$
        }


# 'check_local_procs' command definition
define command{
        command_name    check_local_procs
        command_line    $USER1$/check_procs -w $ARG1$ -c $ARG2$ -s $ARG3$
        }


# 'check_local_users' command definition
define command{
        command_name    check_local_users
        command_line    $USER1$/check_users -w $ARG1$ -c $ARG2$
        }


# 'check_local_swap' command definition
define command{
        command_name    check_local_swap
        command_line    $USER1$/check_swap -w $ARG1$ -c $ARG2$
        }


# 'check_local_mrtgtraf' command definition
define command{
        command_name    check_local_mrtgtraf
        command_line    $USER1$/check_mrtgtraf -F $ARG1$ -a $ARG2$ -w $ARG3$ -c \
         $ARG4$ -e $ARG5$
        }
```

```
###############################################################################
# NOTE:  The following 'check_...' commands are used to monitor services on
#        both local and remote hosts.
###############################################################################

# 'check_ftp' command definition
define command{
        command_name    check_ftp
        command_line    $USER1$/check_ftp -H $HOSTADDRESS$ $ARG1$
        }


# 'check_hpjd' command definition
define command{
        command_name    check_hpjd
        command_line    $USER1$/check_hpjd -H $HOSTADDRESS$ $ARG1$
        }


# 'check_snmp' command definition
define command{
        command_name    check_snmp
        command_line    $USER1$/check_snmp -H $HOSTADDRESS$ $ARG1$
        }


# 'check_http' command definition
define command{
        command_name    check_http
        command_line    $USER1$/check_http -I $HOSTADDRESS$ $ARG1$
        }

define command{
        command_name    check_https
        command_line    $USER1$/check_http -I $HOSTADDRESS$ $ARG1$ -S
        }



# 'check_ssh' command definition
define command{
        command_name    check_ssh
        command_line    $USER1$/check_ssh $ARG1$ $HOSTADDRESS$
        }


# 'check_dhcp' command definition
```

```
define command{
        command_name    check_dhcp
        command_line    $USER1$/check_dhcp $ARG1$
        }


# 'check_ping' command definition
define command{
        command_name    check_ping
        command_line    $USER1$/check_ping -H $HOSTADDRESS$ -w $ARG1$ -c $ARG2$ -p 5
        }


# 'check_pop' command definition
define command{
        command_name    check_pop
        command_line    $USER1$/check_pop -H $HOSTADDRESS$ $ARG1$
        }


# 'check_imap' command definition
define command{
        command_name    check_imap
        command_line    $USER1$/check_imap -H $HOSTADDRESS$ $ARG1$
        }


# 'check_smtp' command definition
define command{
        command_name    check_smtp
        command_line    $USER1$/check_smtp -H $HOSTADDRESS$ $ARG1$
        }


# 'check_tcp' command definition
define command{
        command_name    check_tcp
        command_line    $USER1$/check_tcp -H $HOSTADDRESS$ -p $ARG1$ $ARG2$
        }


# 'check_udp' command definition
define command{
        command_name    check_udp
        command_line    $USER1$/check_udp -H $HOSTADDRESS$ -p $ARG1$ $ARG2$
        }
```

```
# 'check_nt' command definition
define command{
        command_name    check_nt
        command_line    $USER1$/check_nt -H $HOSTADDRESS$ -p 12489 -v $ARG1$ $ARG2$
        }



##############################################################################
#
# SAMPLE PERFORMANCE DATA COMMANDS
#
# These are sample performance data commands that can be used to send performance
# data output to two text files (one for hosts, another for services).  If you
# plan on simply writing performance data out to a file, consider using the
# host_perfdata_file and service_perfdata_file options in the main config file.
#
##############################################################################


# 'process-host-perfdata' command definition
define command{
        command_name    process-host-perfdata
        command_line    /usr/bin/printf "%b" "$LASTHOSTCHECK$\t$HOSTNAME$\t$HOSTSTATE$ \
\t$HOSTATTEMPT$\t$HOSTSTATETYPE$\t$HOSTEXECUTIONTIME$ \
\t$HOSTOUTPUT$\t$HOSTPERFDATA$\n" >> /usr/local/nagios/var/host-perfdata.out
        }


# 'process-service-perfdata' command definition
define command{
        command_name    process-service-perfdata
        command_line    /usr/bin/printf "%b" "$LASTSERVICECHECK$\t$HOSTNAME$ \
        \t$SERVICEDESC$\t$SERVICESTATE$\t$SERVICEATTEMPT$\t$SERVICESTATETYPE$ \
        \t$SERVICEEXECUTIONTIME$\t$SERVICELATENCY$\t$SERVICEOUTPUT$ \
        \t$SERVICEPERFDATA$\n" >> /usr/local/nagios/var/service-perfdata.out
        }
```

Instalação do sendmail e do mailx para envio de notificações via email

```
yum install sendmail
yum install mailx
```

Testes de envio de email

Teste 1:

```
mail -s teste_monit dsanto@wavecom.pt
teste
.
```

Teste 2:

```
/bin/mail -s teste dsanto@wavecom.pt
teste
.
```

Instalação dos *Plugins* do Nagios

```
wget http://prdownloads.sourceforge.net/sourceforge/nagiosplug/ \
nagios-plugins-1.4.15.tar.gz
tar zxvf nagios-plugins-1.4.15.tar.gz
cd nagios-plugins-1.4.15
./configure --prefix=/usr/local/nagios/ --with-nagios- \
user=nagios --with-nagios-group=nagios
make
make all
make install
```

Os plugins instalados deverão ficar visíveis na seguinte directoria:

/usr/local/nagios/libexec

Visualizar *Plugins*

```
cd /usr/local/nagios/libexec/
./check_pop -H monitserver
```

Se, na directoria dos *plugins*, faltar o check_snmp é porque os *plugins* foram instalados na ausência do pacote net-snmp ou net-snmp-utils. Nesse caso deverá proceder-se à reinstalação dos mesmos.

Para o Nagios não mostrar na página WEB o serviço HTTP como "Warning", é necessário adicionar uma página index.html na directoria /var/www/html:

```
cd /var/www/html
touch index.html
```

Posteriormente, reiniciou-se o serviço httpd e o snmpd na máquina remota. Na máquina que monitoriza reiniciaram-se os serviços snmpd e httpd e foi possível verificar na página do Nagios que o serviço httpd se apresentava agora como "OK".

A título de exemplo, seguem-se as figuras como resultado da configuração das ferramentas Nagios e MRTG.

FIGURA F.1: Notificação por email -Nagios

O Nagios, como tinha dito, apresenta-se como uma ferramenta de monitorização bastante completa permitindo, inclusivamente, a configuração da recepção de notificações via email e sms. A figura em cima mostra um exemplo de uma notificação via email. Neste caso, é mostrado um alerta de que a máquina fs01-a, no IP 10.0.0.239, não está a aceitar pedidos seguros no protocolo POP3.

Por sua vez, a imagem que se segue exemplifica a interface de gestão do Nagios. Nela podemos consultar todos os serviços activos numa máquina e o seu estado actual. É possível também consultar a percentagem de utilização dos consumos de processador e memória RAM da máquina.

FIGURA F.2: Interface do Nagios

A figura abaixo pretende mostrar o aspecto do MRTG. Podemos verificar que, com base nos resultados obtidos por SNMP, o MRTG começa a representá-los sob a forma de um gráfico, como é o caso do consumo dos recursos do processador.

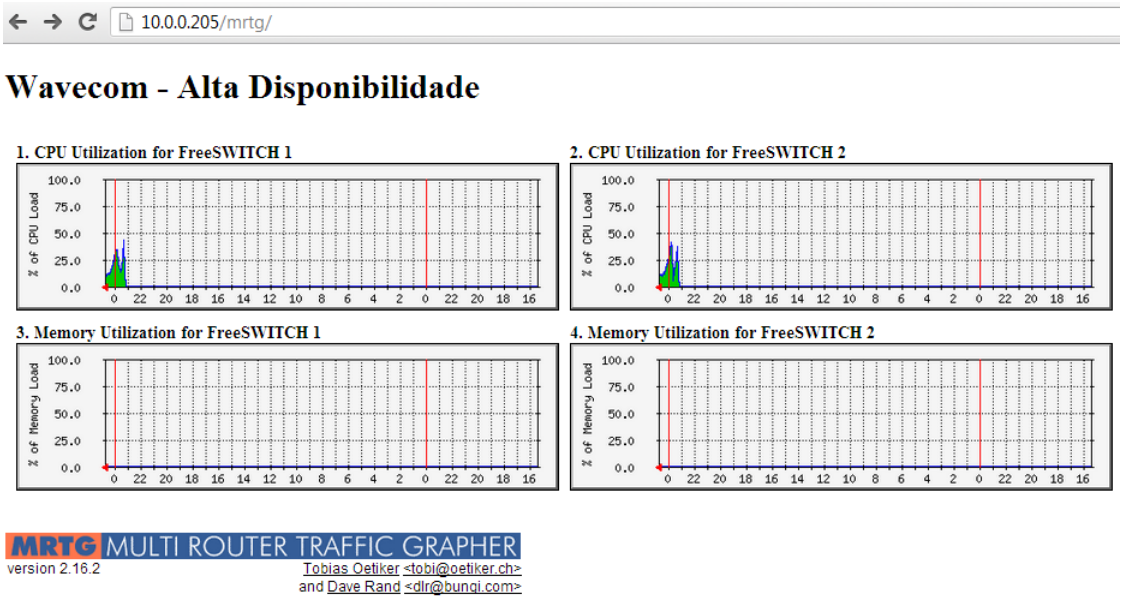

FIGURA F.3: Interface do MRTG

F.4 Instalação do VoIPMonitor

Instalação do VoipMonitor

Instalou-se o FreeSWITCH nesta máquina, uma vez que é necessário para realizar chamadas com base nos *scripts* de teste desenvolvidos.

```
yum install git-core
git clone https://desanto@github.com/finesource/ipcentrex.git
cd ipcentrex
git checkout -b experiments origin/experiments
cd fscloud
./fscloud-install-server.sh
reboot
```

Instalação de alguns pacotes necessários ao VoIPMonitor

```
yum -y install httpd wireshark php php-gd php-mysql php-mbstring mtr php-process \
mysql-server librsvg2
chkconfig --add httpd
chkconfig httpd on
```

```
/etc/init.d/httpd start
chkconfig --add mysqld
chkconfig mysqld on
/etc/init.d/mysqld start
```

Instalação do *sniffer*

```
cd /usr/src/
wget http://sourceforge.net/projects/voipmonitor/files/6.5/ \
voipmonitor-amd64-6.5.4-static.tar.gz
tar xzf voipmonitor*.tar.gz
cd voipmonitor*
./install-script.sh
mkdir /var/spool/voipmonitor/
chown apache /var/spool/voipmonitor/
```

Criação da base de dados

```
mysqladmin create voipmonitor
```

Instalação da GUI WEB

```
mkdir /var/www/html/voipmonitor
cd /var/www/html/voipmonitor
wget "http://www.voipmonitor.org/download-gui?version=latest&major=5&allowed" -O w.tar.gz
tar xzf w.tar.gz
mv voipmonitor-gui*/* ./
rm index.html
chown -R apache /var/www/html
```

Instalação de dependências necessárias à geração dos gráficos e relatórios da GUI WEB

```
wget http://sourceforge.net/projects/voipmonitor/files/wkhtml/0.10.0_rc2/ \
wkhtmltoimage-x86_64 -O "/var/www/html/voipmonitor/bin/wkhtmltoimage-x86_64"
chmod +x "/var/www/html/bin/wkhtmltoimage-x86_64"
wget http://sourceforge.net/projects/voipmonitor/files/wkhtml/0.10.0_rc2/ \
wkhtmltopdf-x86_64 -O "/var/www/html/voipmonitor/bin/wkhtmltopdf-x86_64"
chmod +x "/var/www/html/bin/wkhtmltopdf-x86_64"
```

Instalação do IonCUBE necessário à execução dos *scripts* PHP da GUI do VoIPMonitor

```
wget http://voipmonitor.org/ioncube/x86_64/ioncube_loader_lin_5.3.so -O \
 /usr/lib64/php/modules/ioncube_loader_lin_5.3.so
echo "zend_extension = /usr/lib64/php/modules/ioncube_loader_lin_5.3.so" > \
 /etc/php.d/ioncube.ini
```

Configuração de acesso ao VoIPMonitor

```
vi /etc/http/conf/httpd.conf

        # Access VoipMonitor
        <Location /voipmonitor>
            AuthName "VoIPMonitor Access"
            AuthType Basic
            AuthUserFile /usr/local/voipmonitor/htpasswd.users
            Require valid-user
            Allow from all
        </Location>

        htpasswd -c /usr/local/voipmonitor/htpasswd.users voipmonitoradmin
(pass:wavecom)
```

Reiniciar o serviço Apache e iniciar o serviço voipmonitor

```
sed -i 's/SELINUX=enforcing/disabled/' /etc/selinux/config
setenforce 0
/etc/init.d/httpd restart
/etc/init.d/voipmonitor start
```

Aceder à página WEB do VoipMonitor

http://localhost/voipmonitor

F.5 Configuração do DNS

Tal como referi anteriormente, foi necessário a criação de um servidor de DNS que servisse de balanceador de carga dos pedidos HTTP e SIP efectuados pelos clientes.

```
yum -y install bind bind-libs bind-utils
```

A configuração segue-se em baixo: **named.conf**

```
//
// named.conf
//
// Provided by Red Hat bind package to configure the ISC BIND named(8) DNS
// server as a caching only nameserver (as a localhost DNS resolver only).
//
// See /usr/share/doc/bind*/sample/ for example named configuration files.
//

options {
```

```
#       listen-on port 53 { 127.0.0.1; };
#       listen-on-v6 port 53 { ::1; };
        directory       "/etc/named";
        dump-file       "/var/named/data/cache_dump.db";
        statistics-file "/var/named/data/named_stats.txt";
        memstatistics-file "/var/named/data/named_mem_stats.txt";
        allow-query     { localhost; 10.0.0.0/24; 192.168.0.0/16; 172.16.0.0/16; };
#       recursion yes;
#
#       dnssec-enable yes;
#       dnssec-validation yes;
#       dnssec-lookaside auto;
        forwarders { 8.8.8.8 ; };

        /* Path to ISC DLV key */
        # bindkeys-file "/etc/named.iscdlv.key";
};

logging {
        channel default_debug {
                file "data/named.run";
                severity dynamic;
        };
};

acl internals {
    10.0.0.0/24;
    127.0.0.0/8;
    192.168.201.0/24;
    192.168.200.0/24;
    192.168.101.0/24;
    10.0.0.0/8;
};

acl externals {
};


view "internal" {
    match-clients { internals; };
        zone "aactivo.wavecom.pt" IN {
        type master;
        file "aactivo.wavecom.pt";
        allow-update { none; };
    };
};

view "external" {
```

```
     match-clients { externals; };

};



#include "/etc/named.rfc1912.zones";
```

aactivo.wavecom.pt. Assumindo que o 10.0.0.189 é o servidor de DNS:

```
;
@       IN      SOA     ns.aactivo.wavecom.pt.  support.aactivo.wavecom.pt.     (
                        2008050604      ;serial no
                        28800   ;refresh time
                        7200    ;retry time
                        604800  ;expire time
                        86400 ) ; min time to live
                IN      NS      ns.aactivo.wavecom.pt.

$ORIGIN aactivo.wavecom.pt.
_sip._udp       SRV     10 50 5060 sip1.aactivo.wavecom.pt.
_sip._udp       SRV     10 50 5060 sip2.aactivo.wavecom.pt.
_sip._tcp       SRV     10 50 5060 sip1.aactivo.wavecom.pt.
_sip._tcp       SRV     10 50 5060 sip2.aactivo.wavecom.pt.
_sips._tcp      SRV     10 50 5061 sip1.aactivo.wavecom.pt.
_sips._tcp      SRV     10 50 5061 sip2.aactivo.wavecom.pt.

@               IN      A       10.0.0.239
                IN      A       10.0.0.205
ns              IN      A       10.0.0.189
sip1.aactivo.wavecom.pt.        IN      A       10.0.0.205
sip2.aactivo.wavecom.pt.        IN      A       10.0.0.239
```

Este DNS tem uma zona para o domínio aactivo.wavecom.pt. Esta zona comporta as seguintes características:

- Os pedidos de registo SIP poderão ser feitos quer via UDP, TCP ou TLS (normalmente acontece no caso da utilização de *softphones*);
- Os pedidos de resolução do domínio aactivo.wavecom.pt são resolvidos por duas máquinas, tal como pretendido.

Anexo G

Mecanismos de detecção de falhas

De forma a controlar os serviços que estão a ser executados nos servidores e, uma vez que os mecanismos de detecção e recuperação de falhas por parte do Corosync e Pacemaker são realizados apenas quando os serviços efectivamente falham, surgiu a necessidade de criar *scripts* que permitissem monitorizar activamente estes serviços, tomando precauções no caso de serem detectados comportamentos anómalos no sistema, mais especificamente:

- Quando o serviço Apache retorna HTTP *responses* diferentes das esperadas;
- Quando os serviços postgres, freeswitch e pgpoool apresentam um consumo dos recursos do processador superior ao que seria de esperar.

G.1 *Script* de verificação do estado do serviço Apache

```
# !/bin/bash
#
site=http://10.0.0.239/
email="dsanto@wavecom.pt"
send=-1
subject=""
mailbody=""

curl -s --head $site | head -n 1 | grep "HTTP/1.[01] [23]" > /dev/null
if [ $? -ne 0 ];
then
        echo "ALERT: HTTP BAD RESPONSE"
```

```
        subject="ALERT: HTTP BAD RESPONSE"
        mailbody="ALERT: your website is returning bad responses"
        send=1

else
        echo "HTTP OK"
fi

if [ "$send" == 1 ]; then
        echo $mailbody | mail -s "$subject" "$email"
fi

# Check if the Apache service is getting too much load
top -b -n 1 | grep httpd | awk '{ totuse = totuse + $9 } END { print totuse }'
if [ $? -eq 0 ];
then
        echo "The httpd is running without consuming too much resources"
        mailbody="The postgresql is running without consuming too much resources"
        subject="ALERT: httpd is working properly!"
        send=0
fi
if [ $? -ge 50 ];
then
        mailbody="The httpd is running over 50% CPU Usage"
        subject="ALERT: httpd over 50% CPU usage"
        send=0
fi

if [ $? -ge 70 ];
then
        mailbody="The httpd is running over 70% CPU Usage"
        subject="ALERT: httpd over 70% CPU usage"
        send=2
fi

if [ "$send" -eq 0 ]; then
        echo $mailbody | mail -s "$subject" "$email"
fi

if [ "$send" == 2 ]; then
        shutdown -h now
fi
```

Este *script* detecta se os pedidos de resolução da plataforma WEB retornam códigos HTTP diferentes de 200 e também se o consumo dos recursos do processador é superior a 50% (caso em que envia um email para o administrador) ou se é superior a 70% (caso

em que notifica o administrador por email e desliga o servidor). Neste caso, os pedidos serão servidos pelo segundo servidor até que o primeiro seja reposto.

G.2 *Script* de verificação do estado do serviço Postgres

```
# !/bin/bash
#
email="dsanto@wavecom.pt"
send=-1
mailbody=""
subject=""

# get the sum of the % of cpu of all the postgre processes - $9 represents the column number 9
top -b -n 1 | grep postgres | awk '{ totuse = totuse + $9 } END { print totuse }' > /dev/null

if [ $? -eq 0 ];
then
        echo "Postgres is not consuming too much resources"
        mailbody="Postgres is not consuming too much resources"
        subject="ALERT: nothing to do... everything is working fine!"
        #send=0
fi
if [ $? -ge 50 ];
then
        mailbody="The postgresql is running over 50% CPU Usage"
        subject="ALERT: Postgresql over 50% CPU usage"
        send=1
fi

if [ $? -ge 70 ];
then
        mailbody="Postgresql is running over 70% CPU Usage"
        subject="ALERT: Postgresql over 70% CPU usage. Performing shutdown!"
        send=2
fi

if [ "$send" -eq 0 ]; then
        echo $mailbody | mail -s "$subject" "$email"
fi

if [ "$send" == 2 ]; then
        shutdown -h now
fi
```

G.3 *Script* de verificação do estado do serviço FreeSWITCH

```
# !/bin/bash
#
email="dsanto@wavecom.pt"
send=-1
mailbody=""
subject=""

# get the sum of the % of cpu of all the freeswitch processes - $9 represents the column number
top -b -n 1 | grep freeswitch | awk '{ totuse = totuse + $9 } END { print totuse }' > /dev/null

if [ $? -eq 0 ];
then
        echo "FreeSWITCH is not consuming too much resources"
        mailbody="FreeSWITCH is not consuming too much resources"
        subject="ALERT: nothing to do... everything is working properly!"
        #send=0
fi
if [ $? -ge 50 ];
then
        mailbody="FreeSWITCH is running over 50% CPU Usage"
        subject="ALERT: FreeSWITCH over 50% CPU usage"
        send=1
fi

if [ $? -ge 70 ];
then
        mailbody="FreeSWITCH is running over 70% CPU Usage"
        subject="ALERT: FreeSWITCH over 70% CPU usage. Performing shutdown!"
        send=2
fi

if [ "$send" -eq 0 ]; then
        echo $mailbody | mail -s "$subject" "$email"
fi

if [ "$send" == 2 ]; then
        shutdown -h now
fi
```

G.4 *Script* de verificação do estado do serviço Pgpool

```
# !/bin/bash
#
email="dsanto@wavecom.pt"
send=-1
mailbody=""
subject=""

# get the sum of the % of cpu of all the freeswitch processes - $9 represents the column number
top -b -n 1 | grep pgpool | awk '{ totuse = totuse + $9 } END { print totuse }' > /dev/null

if [ $? -eq 0 ];
then
        echo "Pgpool-II is not consuming too much resources"
        mailbody="Pgpool-II is not consuming too much resources"
        subject="ALERT: nothing to do... everything is working properly!"
        #send=0
fi
if [ $? -ge 50 ];
then
        mailbody="Pgpool-II is running over 50% CPU Usage"
        subject="ALERT: Pgpool-II over 50% CPU usage"
        send=1
fi

if [ $? -ge 70 ];
then
        mailbody="Pgpool-II is running over 70% CPU Usage"
        subject="ALERT: Pgpool-II over 70% CPU usage. Performing shutdown!"
        send=2
fi

if [ "$send" -eq 0 ]; then
        echo $mailbody | mail -s "$subject" "$email"
fi

if [ "$send" == 2 ]; then
        shutdown -h now
fi
```

No caso destes três últimos *scripts* o processo de verificação do seu estado é semelhante para todos. Quando o valor do CPU para o serviço em questão for superior a 50% é enviada uma mensagem ao administrador a notificar esta ocorrência. No caso de ultrapassar os 70% o administrador é notificado e imediatamente é feito um *shutdown*

ao servidor. Num cenário Activo/Passivo ou Activo/Activo, os pedidos serão servidos unicamente pelo segundo servidor até que o primeiro seja reposto.

Importa ainda referir que estes *scripts* são executados de tempos a tempos nos servidores através de uma *cronjob*.

Referências

[1] SiChoon Noh. Active-active high availability of information infraestructure system for effective network security. International Conference on Information Science and Security (ICISS 2008), IEEE. Hyderabad, India, December 2008.

[2] R.M. Gasca P. Neira, Laurent Lefevre. High availability support for the design of stateful networking equipments. Proceedings of the International Conference on Availability, Reliability and Security (ARES'06), IEEE. Vienna, Austria, April 2006.

[3] Ranjith Vasireddy Kishor S. Trivedi. Modeling high availability systems. 12th Pacific Rim International Symposium on Dependable Computing (PRDC'06), IEEE. University of California, Riverside, USA, December 2006.

[4] High Availability Linux Project. http://www.linux-ha.org/wiki/Main_Page, . Consultado a 16/10/2012.

[5] Heartbeat Project History. http://clusterlabs.org/wiki/Pacemaker#Project_History, . Consultado a 16/10/2012.

[6] Werner Vogels, Dan Dumitriu, Ken Birman, Rod Gamache, Mike Massa, Rob Short, John Vert, Joe Barrera, Jim Gray. The design and architecture of the microsoft cluster service - a practical approach to high-availability and scalability. Published in the Proceedings of FTCS'98, IEEE. Munich, Germany, June 1998.

[7] Andrew Beekhof Steven C. Dake, Christiane Caulfield. The corosync cluster engine. Proceedings of the Linux Symposium, IEEE. Ottawa, Canada, July 2008.

[8] OpenAIS Project. http://www.openais.org/doku.php, . Consultado a 17/10/2012.

[9] Corosync and OpenAIS Relationship. http://www.corosync.org/, . Consultado a 17/10/2012.

[10] Pacemaker Supported Cluster Stacks. http://clusterlabs.org/wiki/Pacemaker#Supported_Cluster_Stacks, . Consultado a 17/10/2012.

[11] Hong Tang, Rui She, Chen He and Yunsheng Dou. Construction and application of linux virtual server cluster for scientific computing. NPC '08 Proceedings of the 2008 IFIP International Conference on Network and Parallel Computing, IEEE. Shanghai, China, October 2008.

[12] Yandong Che Yanping Gao, Xiangjun Li. New architecture and algorithm for web-server cluster based on linux virtual server. International Symposium on Information Processing (ISIP), IEEE. Moscow, Russia, May 2008.

[13] Linux Virtual Server - High Availability. http://www.LinuxVirtualServer.org/HighAvailability.html, . Consultado a 23/10/2012.

[14] Anand Gorti. A fault tolerant voip implementation based on open standards. Proceedings of the Sixth European Dependable Computing Conference (EDCC'06), IEEE. Coimbra, Portugal, October 2006.

[15] Hongbo Jiang, Arun Iyengar, Erich Nahum, Wolfgang Segmuller, Asser Tantawi and Charles P. Wright. Load balancing for sip server clusters. IEEE INFOCOM 2009 proceedings. Rio de Janeiro, Brasil, April 2009.

[16] Wei-Ming Wu, Kuochen Wang, Rong-Hong Jan, Chia-Yuan Huang. A fast failure detection and failover scheme for sip high availability networks. 13th IEEE International Symposium on Pacific Rim Dependable Computing. Melbourne, Victoria, Australia. December 2008.

[17] FreeSWITCH - Enterprise Deployment. http://wiki.freeswitch.org/wiki/Enterprise_Deployment, . Consultado a 24/10/2012.

[18] FreeSWITCH - Enterprise Deployment. http://www.anacom.pt/render.jsp?contentId=1162118, . Consultado a 14/6/2013.

[19] ITU - International Telecommunication Union. Performance and quality of service requirements for internation mobile telecommunications. https://www.itu.int/dms_pubrec/itu-r/rec/m/R-REC-M.1079-2-200306-I!!PDF-E.pdf. 2003.