Universidade do Minho
Escola de Engenharia

Projeto de Infraestruturas de Centro de Dados
1º/4º ano MEI/MIEI

# Escalabilidade de Infraestruturas Telemetria de Infraestruturas

Relatório de Desenvolvimento


Isaac Paulo Betuel Mabiala
(pg41074@alunos.uminho.pt)

João Paulo Oliveira de Andrade Marques
(a81826@alunos.uminho.pt)

José André Martins Pereira
(a82880@alunos.uminho.pt)

Ricardo André Gomes Petronilho
(a81744@alunos.uminho.pt)


28 de Dezembro de 2019

# Conteúdo

# Capítulo 1

# Introdução

No âmbito da Unidade Curricular de Infraestruturas de Centro de Dados, foi-nos proposto o desenvolvimento de uma arquitetura para um sistema da aplicação **Tucano**.

Ao longo deste relatório abordam-se as fases de desenho e desenvolvimento da arquitetura, explicação e disponibilidade da mesma, quais os possíveis pontos de falha e testes de carga realizados sobre a mesma, bem como análise dos resultados obtidos em cada fase.

Inicialmente, faz-se uma contextualização do problema, bem como uma análise da aplicação **Tucano**, para se entender a arquitetura da aplicação, bem como as suas dependências.

De seguida, descreve-se a arquitetura implementada, decisões realizadas para se atigirem os objetivos referidos acima.

A identificação de **SPOFS**, ou seja, pontos críticos da arquitetura implementada, e as soluções encontradas, para reduzir o impacto dos mesmos.

Por fim, e não menos importante, apresenta-se uma avaliação do desempenho dos sitema, para uma melhor definição da arquitetura, a nível da disponibilidade dos diferentes componentes, bem como testes de carga.

# Capítulo 2

# Descrição/perceção do Problema

Neste capítulo faz-se uma descrição/contextualização do problema em análise, abordando-se os objetivos pretendidos.

## 2.1 Objetivos

Os principais objetivos deste projeto concentram-se no desenvolvimento de uma arquitetura, com infraestruturas que promovam alta disponibilidade e desempenho, sendo necessário planeamento e deployment para a concretização do mesmo. Após a construção desta arquitetura, tem-se que perceber, os pontos críticos da mesma, e soluções para reduzir o impacto desses mesmos pontos.

## 2.2 Análise da aplicação

A análise da arquitetura da aplicação torna-se importante para melhor perceção da divisão dos diferentes componentes. Deste modo, com uma análise do relatório elaborado pelos criadores da aplicação, conseguiu-se perceber que a aplicação está divida em dois subsistemas, **frontend** e **backend**.

O frontend, utiliza uma biblioteca base do ReactJS em conjunto com componentes na sua maioria do Semantic UI, tendo um servidor em constante execução.

O backend, é composto por uma API utilizando a framework Phoenix escrita em Elixir que por sua vez compila para a BEAM (Erlang virtual machine) garantindo assim de forma fácil a escalabilidade da plataforma. Do mesmo modo, que o frontend, o backend consiste num servidor aplicacional, que interage com o servidor de base de dados de PostgreSQL.

## 2.3 Instalação e Configuração

Com a finalidade de complementar a análise feita anteriormente, e utilizando a documentação disponibilizada pela equipa que desenvolveu o **Tucano**, procedeu-se à instalação da mesma, seguindo o guião de instalação.

Deste modo, o grupo decidu instalar inicialmente todos os componentes na máquina *localhost*, para entender as dependências, e configurações necessárias para o bom funcionamento da aplicação. Os processos seguidos nesta instalação, serão realizados novamente, no deployment do **Tucano**, para a arquitetura apresentada no decorrer do relatório.

Assim, conclui-se que o projeto foi importante para uma melhor perceção e aprofundamento dos conhecimentos obtidos nas aulas teóricas, bem como entender como as aplicações de hoje em dia estão preparadas para falhas que acontecem diariamente.

# Capítulo 3

# Desenvolvimento

Nesta secção serão abordadas as fases de implementação da arquitetura desenvolvida, problemas e respetivas soluções para a implementação, alguns dos pontos críticos que foram encontrados e possíveis soluções, e ainda os resultados obtidos através de testes de carga.

## 3.1 Arquitetura implementada

A arquitetura implementada divide-se em 3 partes: os LVS, o Cluster e o DRBD.

### 3.1.1 LVS

Inicialmente começou-se por configurar duas máquinas LVSs - **Linux Virtual Server** - juntamente com três máquinas WS - Web Server.

Os pedidos dos clientes são recebidos por um dos LVS na placa **NAT** com o endereço IP **192.168.64.168**, sendo de seguida reencaminhados para um dos WS existindo assim **balanceamento e caso um dos WS não esteja funcional o pedido é servido por outro WS**. No caso dos LVSs estão todos ligados mas **apenas um se encontra ativo em cada momento**, ou seja, os pedidos são sempre redirecionados para um LVS e caso esse falhe um dos outros é ativo para continuar a responder aos pedidos que chegam.

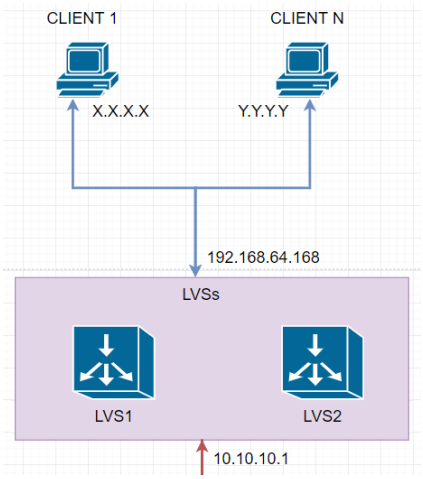


Figura 3.1: Conexões dos Clientes aos LVs.

Desta forma o serviço de FrontEnd é hóspedado pelos respetivos WSs.

A conectividade entre os LVSs e os WSs é efetuado através de uma rede **isolada** na gama de endereços IP **10.10.10.0/24**. Os LVSs são identificados pelo endereço **10.10.10.1** uma vez que é o **gateway** desta rede, isto é, as conexões que todas as máquinas nesta rede efetuarem são mapeadas pelo LVS, isto apenas é possível porque **em cada momento só um dos LVS é que se encontra ativo**.

Assim foi definido no ficheiro **/etc/keepalived/keepalived.conf** do LVS o mapeamento das conexões. No caso da nossa arquitetura foram definidas rotas para a porta **80** (FrontEnd) e para a **4000** (BackEnd).

A necessidade de ter definido a porta 4000 para acesso ao BackEnd deve-se ao facto de alguns pedidos internos do FrontEnd necessitarem de acesso ao BackEnd.

Idealmente os nodos do cluster (3.1.2) deveriam estar numa rede **diferente** da rede isolada criada para a comunicação entre os LVSs e os WSs, mas devido a dificuldades de implementação estes encontram-se na mesma rede, existindo assim conetividade. Note-se que os WSs **não acedem diretamente aos nodos do cluster**, apenas comunicam com os LVSs, e os mesmos redirecionam os pacotes para os respetivos nodos.

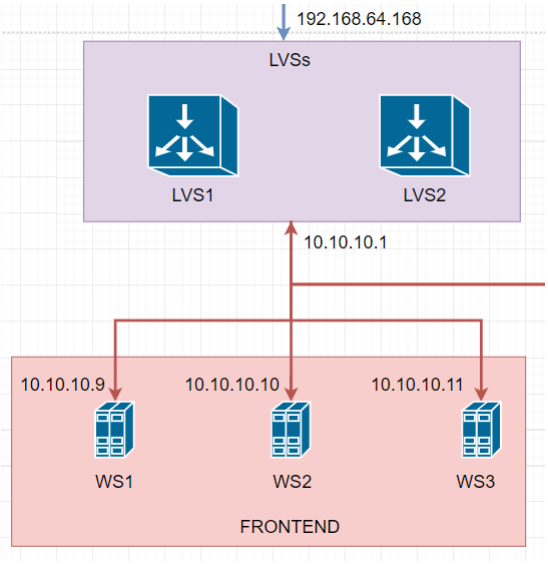


Figura 3.2: Ligações entre os LVSs e os WSs.

### 3.1.2 Cluster

O cluster permite que os serviços expostos pelo mesmo tenham **alta disponibilidade**. No âmbito da nossa arquitetura o cluster é constítuido por dois nodos: **cl1** e **cl2**; com os respetivos endereços IP: **10.10.10.8** e **10.10.10.9**; estando conectados à rede isolada **10.10.10.0/24** para que os LVSs tenham conectividade ao cluster tal como referido na secção anterior.

Com a ajuda das ferramentas: **pcs**, **pacemaker** e **fence-agents-all**; fornecidas pela equipa Red Hat foi configurado o cluster sendo definidos dois serviços: o serviço **tupi** (backend) e o serviço **postgresql** (base de dados). Para cada um dos serviços foram criados três **recursos**: **endereço de IP virtual**, **filesystem** e o **serviço efetivamente a hospedar**. O endereço de IP virtual é utilizado para que as máquinas LVSs externas ao cluster consigam conectar-se ao respetivo serviço. O endereço é virtual uma vez que quando é efetuado um pedido a um serviço, o software do cluster faz o encaminhamento do pedido para o nodo em que se encontra disponível, desta forma **não existe uma conexão direta a um nodo do cluster**.

O serviço postgresql foi construído com três recursos, já referidos acima, sendo que o endereço de IP virtual é o **10.10.10.21**. O filesystem monta a partição **/dev/mapper/mpatha2** (partição 2 do DRBD)

na pasta **/mnt/db**. Por fim para a configuração do servidor foi necessário alterar a pasta onde o serviço postgresql inicia a base dados por padrão, para isso, alteramos a variável **PGDATA** no ficheiro **/var/-lib/pgsql/data/postgres.conf** para a pasta onde foi montado o filesystem da base de dados - **/mnt/db**. Desta forma quando o serviço postgresql é inicializado, **automaticamente inicia o servidor com os ficheiros corretos**. Note-se que o software do cluster escolhe automaticamente qual o nodo a hospedar um determinado serviço, desta forma, os serviços tupi (backend) e postgresql **podem estar em nodos diferentes**, por exemplo, o nodo cl1 pode hospedar o serviço tupi e o nodo cl2 o serviço postgresql, desta forma para que o serviço tupi consiga aceder ao postgresql, o mesmo tem de **permitir conexões externas**. Assim alteramos a variável **listen_addresses** no ficheiro **/mnt/db/postgresql.conf** para - * - e no ficheiro **/mnt/db/pg_hba.conf** introduzimos a linha - **host all all 10.10.10.1/24 md5** - desta forma permite-se conexões ao servidor da base de dados provenientes da rede isolada.

O serviço tupi (backend) foi construído do mesmo modo que o serviço do postgresql, sendo que o endereço de IP virtual é o **10.10.10.20**. O filesystem monta a partição **/dev/mapper/mpatha1** (partição 1 do DRBD) na pasta **/mnt/backend**. Por fim para a configuração do servidor foi necessário criar o ficheiro **/etc/systemd/system/tupi.service** que especifica na variável **ExecStart** o script a correr quando é inicializado o serviço tupi com o programa **systemd**. Neste caso o script que é corrido é o ficheiro **/mnt/-backend/tupi/tupi.sh**.

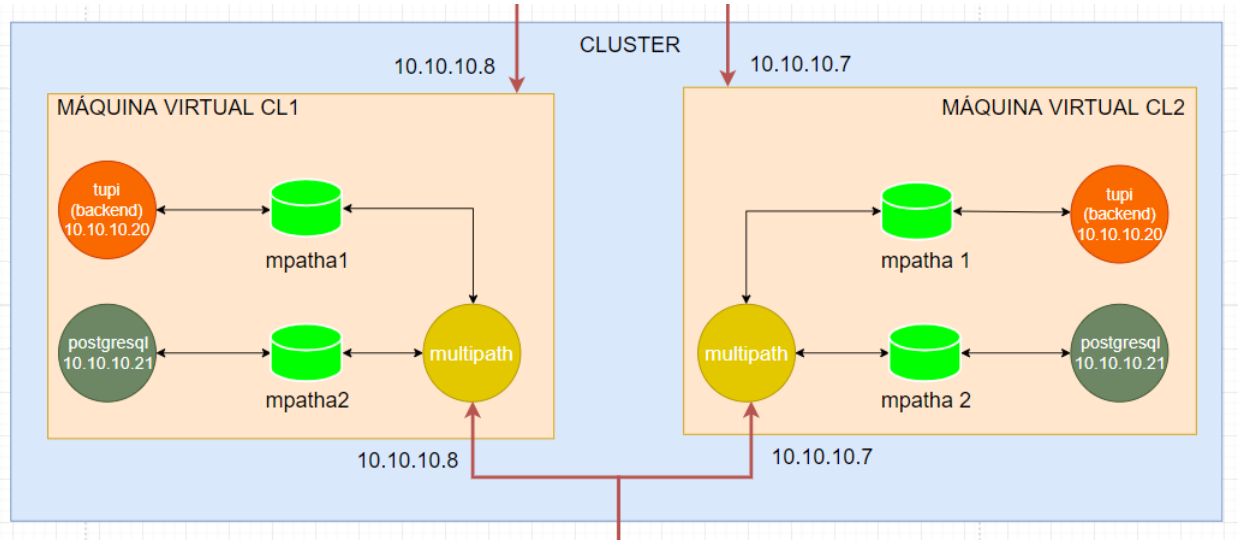


Figura 3.3: Cluster com os respetivos nodos e serviços.

### 3.1.3 DRBD

Na arquitetura desenvolvida neste projeto foi implementado um **DRDB** - Distributed Replicated Block Device.

No processo de configuração do DRBD foi utilizado o protocolo **iSCSI** que é um protocolo IP para conectar entre si instalações de armazenamento de dados. Permite o controlo de dispositivos de armazenamento através de comandos **SCSI** - Small Computer Systems Interface – pela camada TCP/IP (pela rede). Desta forma quem utilizar o protocolo iSCSI conecta-se a um dispositivo remoto (na rede) da mesma maneira que a um dispositivo periférico local (ex: pen usb) tendo total **abstração da conexão remota**. A arquitetura deste protocolo consiste em iSCSI clients (nodos do cluster) e iSCSI targets (máquinas drbd1 e drbd2).

Foi também utilizado o programa **multipath** que configura múltiplas rotas entre um servidor e seus dispositivos de armazenamento remotos de forma a **agregar todas as rotas num único caminho** para

o dispositivo.

Para existir comunicação entre as máquinas DRBDs e dos nodos do cluster foi utilizada a rede isolada 10.10.10.0/24, sendo que a máquina drbd1 está a usar o endereço IP 10.10.10.41 e a drbd2 utiliza o endereço IP 10.10.10.40.

No caso de **failover** de um dos DRBDs o sistema **continua a funcionar pois ambas as máquinas DRBD são DRBD primários**. Quando a máquina DRBD que falhou voltar ao ativo novamente faz sincronização de dados com a DRBD que continuou a sua execução enquanto esta falhou.

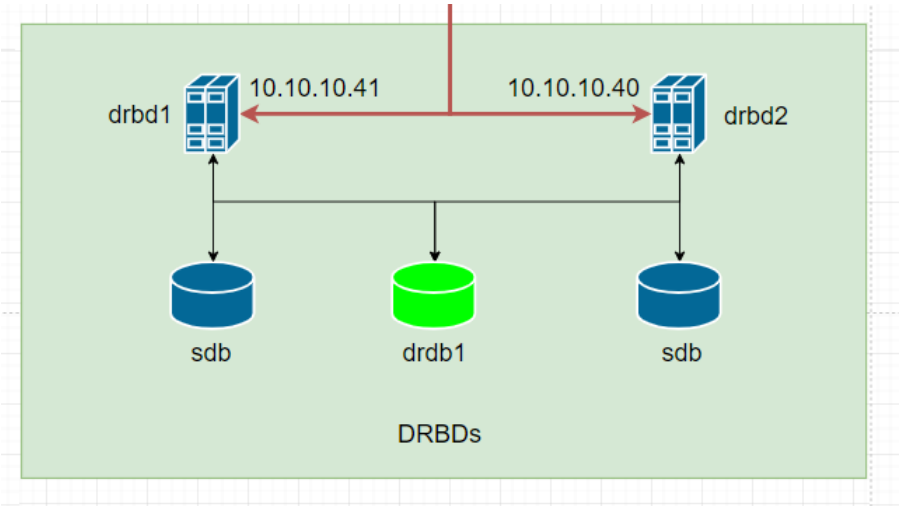


Figura 3.4: Arquitetura do DRBD.

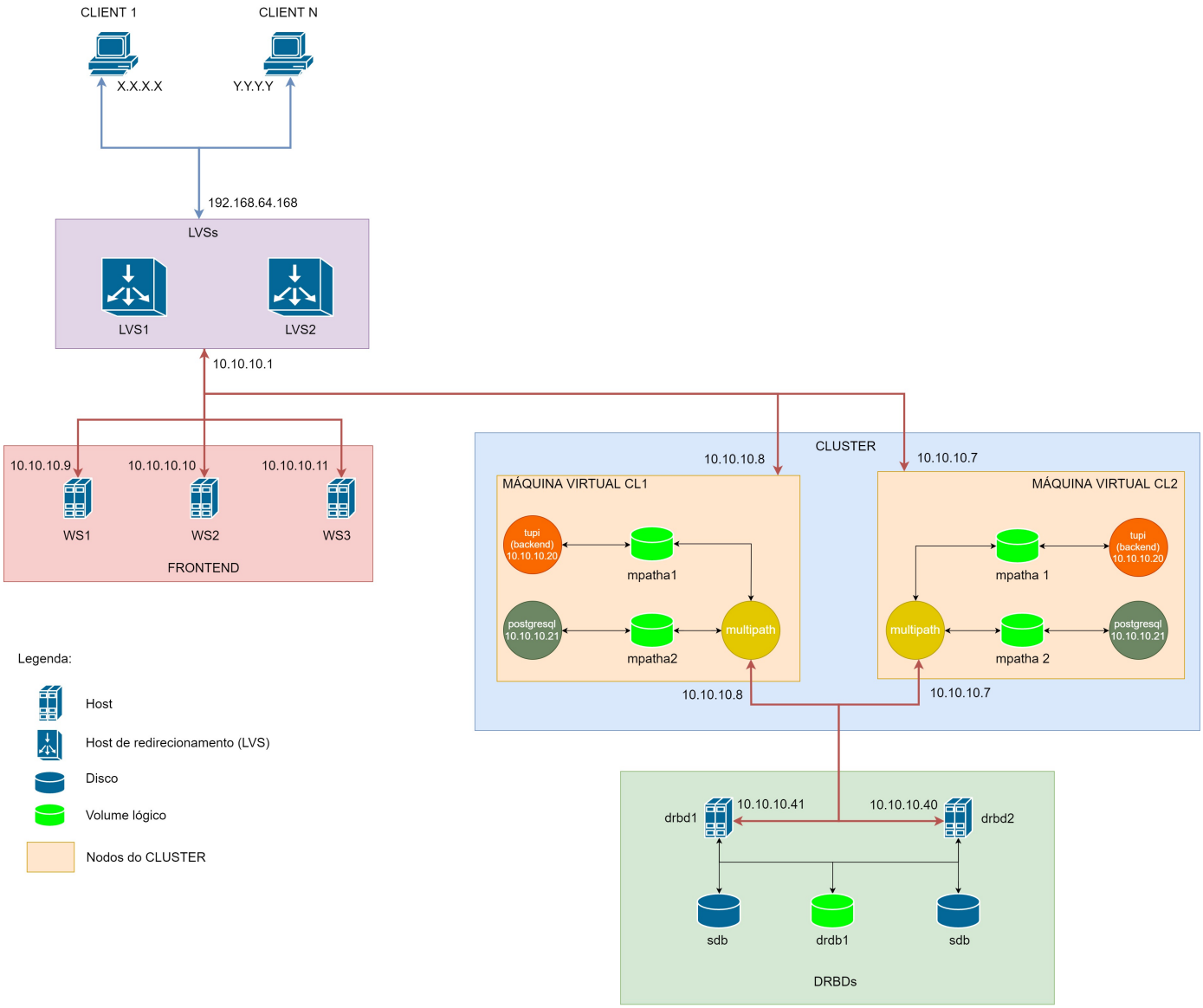


Figura 3.5: Arquitetura da infraestrutura.

## 3.2 Identificação de SPOFs

Os possíveis pontos de rutura da arquitetura são: **LVS**, *web services* , **DRBD** e **Cluster**. Importante referir, que existirão sempre estes pontos de rutura, pois não se consegue evitar a 100%, visto que os recursos não são ilimitados, mas sim reduzir a probabilidade das ruturas acontecerem. O **LVS** pode provocar a quebra do sistema, pois consiste no ponto de entrada no mesmo. Deste modo, a sua falta, torna todo o sistema inutilizável. Como solução para este ponto crítico, a equipa decidiu, como já referido anteriormente, ter uma segunda máquina **LVS**, pronta para entrar em funcionamento em caso de falha.

Do mesmo modo, os *web services*, máquinas com o servidor frontend, são um ponto de rutura, pois com apenas um servidor, em caso da sua falha, o sistema fica inutilizável, pois necessita-se da interface gráfica para utilização da aplicação. Assim, o grupo decidiu colocar três máquinas com os *web services*, ou seja, podem falhar até duas máquinas, e o sistema mantém-se em funcionamento. Com a adição destas três

máquinas, também garante-se a distribuição da carga, reduzindo assim, a possibilidade de falha das mesmas.

Em relação ao cluster, este também se torna um ponto crítico, pois agrega os serviços de backend, e base de dados, cruciais ao funcionamento do sistema. Deste modo, com a finalidade de garantir a disponibilidade destes mesmos serviços, decidiu-se conter dois nodos no cluster, tal como já foi referido, sendo que, os serviços em caso de falha de um servidor, automaticamente, se ativam em outro nodo do mesmo, logo, consegue-se ter até a um máximo de uma falha no cluster, e continuar-se em funcionamento. O ideal seria adicionar mais nodos ao cluster, para reduzir a probabilidade de rutura, no entanto, face aos recursos, apenas foi possível colocar dois nodos.

Por fim, e não menos importante, têm-se o ponto de rutura do **DRBD**, responsável pelos dados dos serviços do **Cluster**, logo percebe-se a importância e os problemas, que a falha do mesmo, causa em todo o sistema. Assim, a própria arquitetura do **DRBD** está preparada para falhas, garantindo-se que na falta de uma das máquinas do mesmo, os dados dos serviços estão salvaguardados e mantidos em funcionamento.

## 3.3 Avaliação do desempenho do sistema

Para a avaliação de desempenho do sistema foram realizados vários testes para garantir a alta disponibilidade e fiabilidade do mesmo. Para tal foram realizados testes de carga ao FrontEnd e ao BackEnd, foram ainda realizados testes ao nível de disponibilidade do sistema, caso de failover do LVS ou dos nodos do cluster.

Para a realização dos testes de carga recorreu-se à ferramenta **Jmeter**, a qual serve para simular um grupo de clientes (threads) a realizar um certo pedido à apliacação **Tucano**, sendo que cada cliente faz o mesmo pedido uma quantidade específica de vezes.

### 3.3.1 Teste de carga FrontEnd

Para o teste de carga ao FrontEnd foram utilizados diferentes quantidades de clientes e de números de pedidos realizados pelos mesmos.

Para uma maior variedade de testes foi simulado o failover de um ou dois dos WSs, pelo que foram realizados testes com 1, 2 e 3 WSs ativos.

Os resultados obtidos para os diferentes conjuntos de valores (número de clientes, número de pedidos e número de WSs ativos) foram os seguinte:

| #C + #P / #WS | 1 WS | 2 WS | 3 WS |
|---|---|---|---|
| 10 C + 10 P | 37500 | 45801 | 57692 |
| 10 C + 100 P | 134228 | 124740 | 95693 |
| 10 C + 1000 P | 153335 | 151745 | 120821 |
| 50 C + 10 P | 80213 | 84507 | 63694 |
| 50 C + 100 P | 167130 | 156168 | 122399 |
| 50 C + 1000 P | 209731 | 241682 | 204178 |
| 100 C + 10 P | 52401 | 64724 | 87719 |
| 100 C + 100 P | 147275 | 206043 | 178041 |
| 100 C + 1000 P | 230937 | 249750 | 231151 |

Tabela 3.1: Throughput por minuto.

C - Clientes, P - Pedidos, WS - Web Server

A funcionalidade do *Tucano* utilizada para este teste foi o *login*, que consiste em carregar apenas a página de frontend do login.

### 3.3.2 Teste de carga BackEnd

Para este teste foram utilizados os mesmos conjuntos de clientes e de pedidos tendo obtido os seguintes resultados:

| #C + #P | Throughput |
|---|---|
| 10 C + 10 P | 226 |
| 10 C + 100 P | 225 |
| 50 C + 10 P | 224 |
| 50 C + 100 P | 227 |
| 100 C + 10 P | 229 |
| 100 C + 100 P | 225 |

Tabela 3.2: Throughput por minuto.

C - Clientes, P - Pedidos

A funcionalidade do *Tucano* utilizada para este teste foi a *autenticação*, que consiste em aceder ao backend para este realizar a autenticação do cliente.

### 3.3.3 Discussão dos resultados

Após a conclusão dos testes, percebeu-se que com o aumento dos *web services* obtêm-se melhores resultados, no entanto, devido à falta de recursos, não foi possível colocar mais máquinas.

Por outro lado, nota-se um aumento de desempenho em relação à arquitetura monolítica realizada inicialmente na máquina *localhost*, devido à distribuição de carga e aumento de disponibilidade, bem como uma melhoria de tolerância à faltas. Com apenas uma máquina, uma falha, p todo o sistema estava compremetido, o que não acontece na arquitetura implementada.

### 3.3.4 Disponibilidade LVS

Para averiguar a disponibilidade das máquinas LVS e das máquinas WS foram todas estas colocadas a correr e foram realizados pedidos ao FrontEnd verificando que se obtinha resposta.

Para verificar se a implementação resistia a failovers foi simulado a falha de ambos os LVS (um de cada vez), a falha de ambos os WS (um de cada vez e dois ao mesmo tempo) e a falha de um LVS e de um ou dois WS. Para todos estes casos a implementação resistiu à falha, pois o LVS apesar de estarem os 2 a correr apenas 1 está ativo, pelo que a falha de um dos LVS faz com que o outro esteja ativo na mesma. No caso dos WSs ambos estão a correr e ativos pelo que a falha de um ou dois apenas faz com que em vez de estarem os 3 ativos está apenas 1 ou 2. Quando ambos falham (um LVS e um ou dois WS) a situação é igual à de quando falha apenas um ou dois WS, porque os LVSs tem sempre apenas um ativo.

Com esta implementação garantimos o failover de 1 LVS, o failover de 2 WS, o failover de 1 LVS e 1 WS, e o failover de 1 LVS e 2 WS.

Se fosse necessário garantir uma maior resistência a falhas seria necessário aumentar o número de recursos, ou seja, se fosse necessário resistir à falha de 2 LVSs tinhamos de ter no mínimo 3 LVSs e o mesmo acontece para os WSs.

### 3.3.5 Disponibilidade Cluster

O teste de disponibilidade do cluster é idêntico ao teste de disponibilidade realizado ao LVS, ou seja, com a falha de um dos nodos do cluster este continua a responder aos pedidos que lhe são efetuados.

Para testar foi simulada a falha de um dos nodos do cluster e com a utilização do RedHat para configuração de serviços nos cluster foi possível verificar os serviços a passarem de um nodo para o outro sem o cliente reparar que existiu falha.

Nesta implementação foram utilizados 2 nodos do cluster pelo que é garantido o failover de um nodo do cluster, para garantir um maior número de failovers são necessários mais recursos. No caso de ser necessário garantir o failover de 3 nodos do cluster este tem de ter no mínimo 4 nodos.

# Capítulo 4

# Conclusão

Assim conclui-se que os objetivos inicialmente propostos foram atingidos. O processo inicial de contextualização, perceção e instalação em *localhost* da aplicação foi importante, para um melhor entendimento da arquitetura da mesma, bem como das suas dependências e configurações necessárias.

A definição da arquitetura e infraestrutura apresentada para os componentes da aplicação **Tucano**, permite elevada disponibilidade, performance, face aos recursos disponíveis das máquinas onde foram testadas.

Do mesmo modo, conseguiu-se aumentar a resistência a falhas dos pontos críticos com a replicação de alguns componentes, tais como, **LVS** e *web services* e aumento de nodos no cluster.

Ocorreram algumas dificuldades no encaminhamento do frontend para o backend, pois inicialmente os nodos do cluster estavam numa rede diferente do **LVS**. No entanto, para resolver este problema de comunicação entre os componentes foi necessário colocá-los na mesma rede, sendo que o **LVS** passou a reencaminhar pedidos para os nodos do cluster que tenham o serviço do **backend**.

A concretização do patricionamento do disco presente nos nodos do cluster, tornou-se difícil. O patricionamento é necessário, para garantir a independência dos serviços **backend** e **base de dados**, ou seja, estes podem estar em nodos diferentes do cluster, logo os seus dados, também terão que estar em partições diferentes.

A falta de recursos dificultou a realização dos testes, como a limitação da placa de rede, bem como o número de máquinas ligadas em simultâneo, devido à falta de memória.

A criação do serviço do servidor backend, também dificultou e impediu o progresso, pois a equipa desconhecia a forma de criação de serviços **systemd**, mas com o auxílio da equipa docente, conseguiu-se desenvolver o *script*.

Após o estabelecimento da infraestrutura, não se conseguiu executar a funcionalidade **login**, o qual se devia ao problema de os nodos do cluster conterem placas **NAT** ativas, e assim o backend "respondia"para esta mesma placa, ou seja, erradamente, pois deveria ser para o **IP** do **LVS**. Deste modo, a solução foi remover esta placa, no entanto, a conexão entre os nodos do cluster e as máquinas **DRBD**, usava a placa NAT. Para evitar problemas de conexão entre os nodos do **Cluster**, com as máquinas do **DRBD**, colocou-se todas as máquinas na mesma rede.