

Universidade de Aveiro

Departamento de Eletrónica, Telecomunicações e Informática

Mestrado em Engenharia Informática

Ano Letivo 2020/2021

# Arquiteturas de Software

## Trabalho Prático 3

Aveiro, 20 de junho de 2021

Luís Laranjeira, 81526

Rafael Sá, 104552

# Índice

1. Introdução.....	4
2. Implementação .....	5
2.1. Desempenho .....	6
2.2. Disponibilidade .....	8
2.3. Escalabilidade .....	10
2.4. Usabilidade.....	10
2.4.1. Interfaces do Load Balancer.....	10
2.4.2. Interfaces do Monitor .....	11
2.4.3. Interfaces dos Servidores.....	13
2.4.4. Interfaces dos Clientes.....	15
2.4.5. Interface do Control Panel.....	16
3. Requisitos Implementados e Distribuição de Tarefas.....	17
4. Conclusões .....	18

# Índice de Figuras

Figura 1 - Arquitetura da Solução.....	5
Figura 2 - Função para Criação de Processos .....	5
Figura 3 - Variáveis da Classe Serializável para as Mensagens .....	6
Figura 4 - Função para obter o Servidor mais livre.....	7
Figura 5 - Verificação se a fila de espera do Servidor está cheia .....	7
Figura 6 - Ciclo de vida das threads de processamento de pedidos.....	8
Figura 7 - Ciclo de vida da thread de heartbeat dos servidores.....	8
Figura 8 - Ciclo de vida da thread de monitorização de servidores .....	8
Figura 9 - Processamento da mensagem de Servidor Down no Load Balancer .....	9
Figura 10 - Mensagens recebidas pelo monitor para manter o estado do cluster atualizado. 9	
Figura 11 - Registo de novo servidor no monitor.....	10
Figura 12 - Interface de configuração do load balancer.....	10
Figura 13 - Interface de apresentação de pedidos a ser tratados pelo load balancer .....	11
Figura 14 - Interface de configuração do monitor .....	11
Figura 15 - Interface com listagem de servidores do cluster.....	12
Figura 16 - Interface com listagem de pedidos a serem geridos pelo load balancer.....	12
Figura 17 - Interface com o estado dos pedidos a serem processados por um determinado servidor .....	13
Figura 18 - Interface de configuração de um servidor .....	13
Figura 19 - Interface com listagem dos pedidos recebidos num servidor .....	14
Figura 20 - Interface com listagem dos pedidos processados num servidor.....	14
Figura 21 - Interface de configuração de um cliente.....	15
Figura 22 - Interface com listagem dos pedidos pendentes e criação de novos pedidos.....	15
Figura 23 - Interface com listagem dos pedidos executados e criação de novos pedidos ...	16
Figura 24 - Interface do control panel.....	16

# 1. Introdução

Este trabalho foi realizado no âmbito da unidade curricular de Arquiteturas de Software, a fim de avaliar e colocar em prática os conhecimentos obtidos na componente teórica. Este trabalho foca essencialmente nos atributos de qualidade e nas táticas arquiteturais que são necessárias para que estes sejam implementados com sucesso na solução desenvolvida.

O objetivo deste trabalho é desenhar e desenvolver uma infraestrutura, mais concretamente um *cluster* de servidores, que forneça a clientes um serviço computacional matemático, neste caso o cálculo da constante de Avogadro.

A arquitetura da solução deve basear-se em quatro atributos de qualidade: desempenho, disponibilidade, escalabilidade e usabilidade. De acordo com estes atributos de qualidade, foram selecionadas táticas arquiteturais a serem implementadas na solução.

Em relação ao desempenho, os pedidos dos clientes devem ser distribuídos de forma justa pelos servidores ativos (réplicas de computação). Para além disso, os servidores devem processar pedidos de forma concorrente, com um máximo de 3 processamentos simultâneos e 2 pedidos na fila de espera.

No que toca à disponibilidade deve existir redundância, ou seja, no caso de um servidor falhar, os pedidos que esse servidor estivesse a processar devem ser delegados a outros servidores. Adicionalmente, deve existir um monitor que supervisione o estado do *cluster*.

Relativamente à escalabilidade, deve ser possível escalar horizontalmente, adicionando novos servidores quando for necessário.

Por último, em relação à usabilidade, as interfaces gráficas devem permitir avaliar a solução desenvolvida (utilização, rastreabilidade, verificação, validação, entre outros).

A solução deve conter um *load balancer*, que é responsável pela distribuição dos pedidos dos clientes de forma justa pelos vários servidores, um monitor, responsável por manter uma vista atualizada do estado do *cluster*, um ou mais servidores, responsáveis por processar os pedidos dos clientes, e um ou mais clientes responsáveis por efetuar os pedidos. Todas as entidades devem possuir uma interface gráfica própria para configuração e monitorização.

## 2. Implementação

Neste capítulo será feita uma apresentação da implementação da solução. A figura 1 apresenta um esquema representativo da arquitetura da solução desenvolvida.

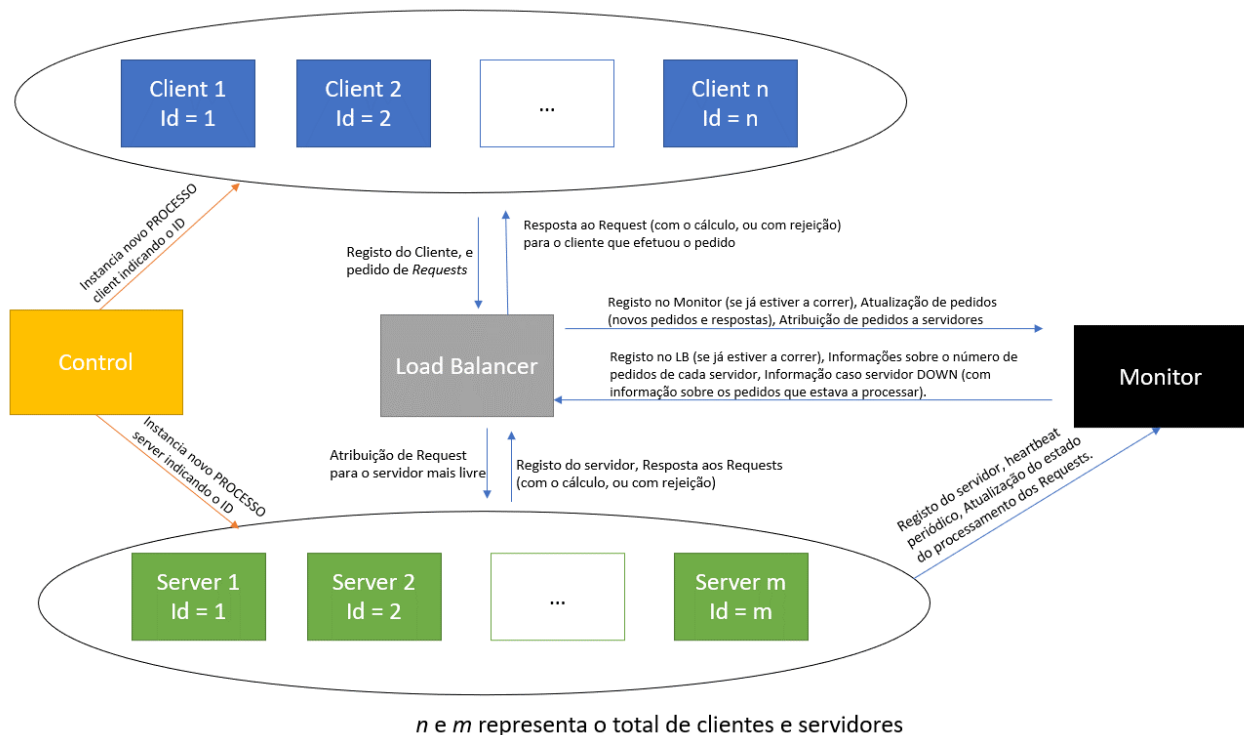


Figura 1 - Arquitetura da Solução

A entidade Control é responsável por fazer a gestão das entidades, podendo lançar novos processos de servidores e clientes com o respetivo ID, bem como do monitor e o load balancer. A criação dos processos é feita recorrendo ao *ProcessBuilder* do Java, como é apresentado na figura 2.

```
private ProcessBuilder getProcessBuilder(Class cls, List<String> args) {
    String javaHome = System.getProperty("java.home");
    String javaBin = javaHome + File.separator + "bin" + File.separator + "java";
    String classpath = System.getProperty("java.class.path");
    String className = cls.getName();

    List<String> command = new ArrayList<>();
    command.add(javaBin);
    command.add("-cp");
    command.add(classpath);
    command.add(className);
    command.addAll(args);
    return new ProcessBuilder(command);
}
```

Figura 2 - Função para Criação de Processos

Apenas o *load balancer* e o monitor possuem um *ServerSocket* para aceitar novos clientes, pelo que são os clientes e os servidores responsáveis por efetuar o registo e estabelecer um canal de comunicação. No caso do canal de comunicação entre o *load balancer* e o monitor, quando uma destas entidades é iniciada, é verificado se a outra entidade já está a correr e, se estiver, é estabelecida a ligação. Caso contrário, essa entidade aguarda que seja a outra a estabelecer a ligação quando iniciar.

A troca de mensagens entre as várias entidades é feita utilizando uma única classe serializável, que possui os vários construtores necessários para os vários tipos de mensagens. A figura 3 apresenta as variáveis utilizadas nas mensagens.

```
public class Message implements Serializable{

    /** Client ID. */
    private int clientId;
    /** Request ID. */
    private int requestId;
    /** Server ID. */
    private int serverId;
    /** Message code. */
    private int messageCode;
    /** Number of iterations. */
    private int iterations;
    /** Value of NA, Avogadro Constant. */
    private String valueNa;
    /** Requests being processed by a server. */
    private List<Message> serverRequests;
    /** Number of requests being processed by each server. */
    private List<ServerCounter> serverCounters;
```

Figura 3 - Variáveis da Classe Serializável para as Mensagens

De seguida será feita uma apresentação detalhada de como foram implementadas as táticas arquiteturais de cada um dos requisitos.

## 2.1. Desempenho

Sempre que o *load balancer* recebe um novo pedido de um cliente, o Monitor é informado e responde com a informação acerca do número de pedidos que cada servidor está a processar. Ao receber esta informação, o *load balancer* verifica qual o servidor que está a processar menos pedidos e atribui-lhe esse pedido. A função que verifica o servidor mais livre, de forma a que o pedido seja atribuído de forma justa, é apresentado na figura 4.

```
private int getFreestServer(List<ServerCounter> serversCounters) {
    int serverId = -1, minCounter = Integer.MAX_VALUE;
    boolean isFirst = true;
    for (ServerCounter serverCounter : serversCounters) {
        if(isFirst){
            isFirst = false;
            minCounter = serverCounter.getCounter();
            serverId = serverCounter.getServerId();
        } else if(serverCounter.getCounter() < minCounter){
            minCounter = serverCounter.getCounter();
            serverId = serverCounter.getServerId();
        }
    }
    return serverId;
}
```

Figura 4 - Função para obter o Servidor mais livre

Quando o servidor recebe um novo pedido, é verificado se a fila de espera está cheia. Se estiver cheia o pedido é rejeitado, caso contrário esse pedido é adicionado à fila de espera. A figura 5 apresenta o código que faz essa verificação.

```
while((msg = cLB.receiveMessage()) != null){
    if(queueIsFull()){
        msg.setServerId(serverId);
        msg.setMessageCode(MessageCodes.REJECTION);
        cLB.sendMessage(msg);
        serverGUI.addRequestReceived(msg, "Rejected");
    } else {
        addRequestToQueue(msg);
        serverGUI.addRequestReceived(msg, "In Queue");
    }
}
```

Figura 5 - Verificação se a fila de espera do Servidor está cheia

Para processar os pedidos nos servidores, são instanciadas *threads* que irão alternar entre o estado de processamento de pedido, e a espera de pedidos dentro de um FIFO. O número de *threads* corresponde ao número máximo de pedidos que podem ser executados em simultâneo.

Quando um pedido é adicionado à fila de espera, é dada indicação para que a próxima *thread* no FIFO seja acordada para processar esse pedido. Caso não exista nenhuma *thread* no FIFO, o pedido continua na fila de espera até uma das *threads* estar livre. A figura 6 apresenta o ciclo de vida das *threads* de processamento de pedidos.

```

@Override
public void run() {
    Message msg;
    while(fifo.in()) {
        msg = getNextRequestOnQueue();
        if(msg != null){
            Message reply = getReplyMessage(msg);
            cLB.sendMessage(reply);
        }
    }
}

```

Figura 6 - Ciclo de vida das threads de processamento de pedidos

## 2.2. Disponibilidade

Para manter um estado atualizado do *cluster*, o servidor é responsável por enviar uma mensagem de *heartbeat* periódica a indicar que ainda está ativo. O ciclo de vida da *thread* responsável pelo *heartbeat* dos servidores é apresentada na figura 7.

```

@Override
public void run() {
    Message heartbeatMessage = new Message(true, serverId, MessageCodes.HEARTBEAT);
    while(cMonitor.sendMessage(heartbeatMessage)) {
        try {
            Thread.sleep(heartbeatTimeout);
        } catch (InterruptedException ex) {
            System.out.println(ex.toString());
        }
    }
}

```

Figura 7 - Ciclo de vida da thread de heartbeat dos servidores

O monitor possui uma *thread* para cada servidor que é responsável por verificar que o servidor não excede um determinado *threshold* entre mensagens de *heartbeat*. Sempre que é recebida uma mensagem de *heartbeat*, a contagem do tempo para o *threshold* é reiniciada. O ciclo de vida da *thread* responsável pela monitorização do *heartbeat* dos servidores é apresentada na figura 8.

```

@Override
public void run() {
    while(!isEnd()){
        try {
            Thread.sleep(heartbeatThreshold);
            break;
        } catch (InterruptedException ex) {}
    }
    if(!isEnd())
        serverDown(serverId);
}

```

Figura 8 - Ciclo de vida da thread de monitorização de servidores



Ao ser dada a indicação que um servidor está em baixo, o monitor envia a indicação ao *load balancer*, já com a informação relativa aos pedidos que esse servidor estava a processar. Ao receber essa informação, o *load balancer* repete o processo de atribuição de servidor para cada um dos pedidos. Como já foi referido, esse processo consiste em informar o monitor acerca do novo pedido, o monitor responder com o estado de cada servidor, e o *load balancer* atribuir o pedido ao servidor mais livre. A figura 9 apresenta o processamento da mensagem no *load balancer* quando um servidor está em baixo.

```
@Override
public synchronized void serverDown(int serverId, List<Message> messages) {
    cServers.remove(serverId);
    messages.forEach(msg -> {
        lbGUI.removeRequestFromTable(msg);
        newRequest(msg);
    });
}
```

Figura 9 - Processamento da mensagem de Servidor Down no Load Balancer

O monitor é responsável por manter o estado do *cluster* atualizado, pelo que informações acerca de toda as atualizações no sistema. A figura 10 apresenta todo o tipo de mensagens que o monitor recebe.

```
while((msg = cc.receiveMessage()) != null){
    switch(msg.getMessageCode()){
        case MessageCodes.REG_LB_M:
            loadBalancerUp(cc);
            break;
        case MessageCodes.REG_SERVER:
            newServer(msg.getServerId(), cc);
            break;
        case MessageCodes.REQUEST:
            newLBRequest(msg);
            break;
        case MessageCodes.ASSIGNMENT:
            requestAssigned(msg.getRequestId(), msg.getServerId());
            break;
        case MessageCodes.CUR_ITER:
            serverIterationsUpdate(msg.getRequestId(), msg.getIterations(), msg.getServerId());
            break;
        case MessageCodes.REPLY:
        case MessageCodes.REJECTION:
            newLBReply(msg);
            break;
        case MessageCodes.HEARTBEAT:
            serverHeartbeat(msg.getServerId());
            break;
        case MessageCodes.REJECTION_NO_SERVERS:
            requestRejectionByNoServers(msg.getRequestId());
            break;
        case MessageCodes.TEST_MESSAGE:
            cc.closeConnection();
            return;
    }
}
```

Figura 10 - Mensagens recebidas pelo monitor para manter o estado do cluster atualizado

## 2.3. Escalabilidade

Para garantir a escalabilidade da solução, é possível lançar novos servidores sempre que necessário, sendo que estes registam-se no *load balancer* e no monitor ao serem iniciados, ficando logo disponíveis para receber novos pedidos. A figura 11 apresenta o processo de registo de um novo servidor no monitor.

```
@Override
public synchronized void newServer(int serverId, CClient cc) {
    servers.put(serverId, new ServerInfo(cc));
    serversCounters.put(serverId, new ServerCounter(serverId, 0));
    monitorGUI.addServerToTable(serverId);
    heartbeatThreads.put(serverId, new ServerHeartbeatThread(serverId));
    heartbeatThreads.get(serverId).start();
}
```

Figura 11 - Registo de novo servidor no monitor

## 2.4. Usabilidade

Todas as entidades possuem as respetivas interfaces gráficas, tendo sido desenhadas de forma a poderem ser utilizadas no processo de avaliação. Todas as interfaces de configuração possuem os campos preenchidos com os valores por defeito. Para além disso, todos os campos numéricos e obrigatórios são verificados.

### 2.4.1. Interfaces do Load Balancer

A interface gráfica do *load balancer* é composta pela interface de configuração, apresentada na figura 12, e pela interface que apresenta os pedidos que o load balancer tem pendentes para serem atribuídos ou que estão a ser processados pelos servidores, apresentada na figura 13.

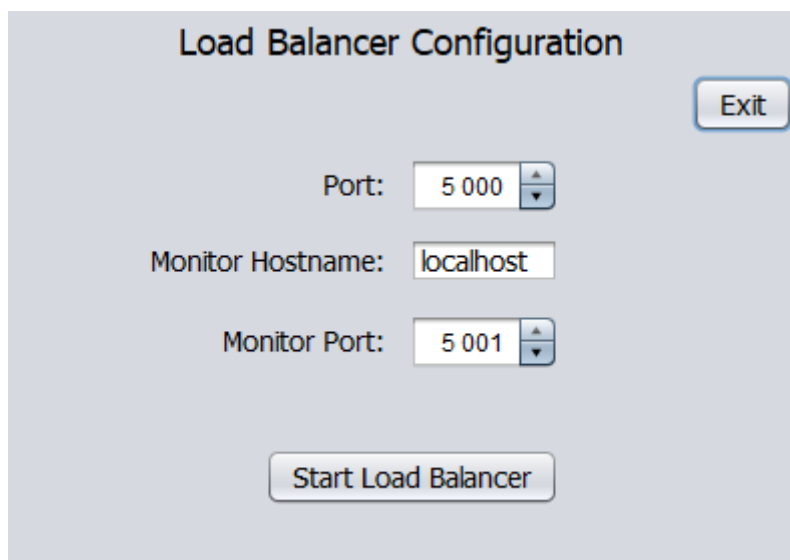
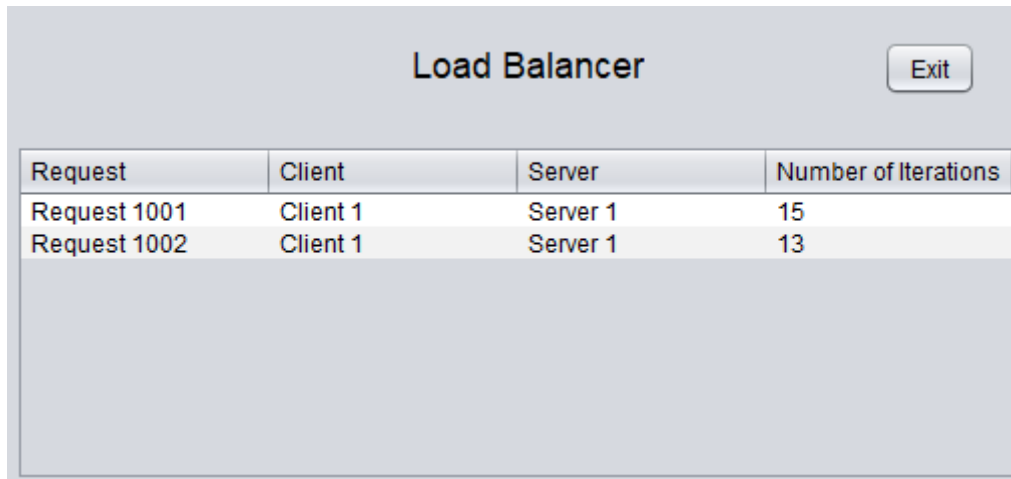
A screenshot of a Java Swing window titled "Load Balancer Configuration". The window has a light gray background. In the top right corner, there is a blue "Exit" button. The main area contains three configuration fields: "Port:" with a text box containing "5 000" and a spin button; "Monitor Hostname:" with a text box containing "localhost"; and "Monitor Port:" with a text box containing "5 001" and a spin button. At the bottom center, there is a large "Start Load Balancer" button.

Figura 12 - Interface de configuração do load balancer

Na interface de configuração é possível configurar o *port* do load balancer, e ainda o *host name* e o *port* do monitor.



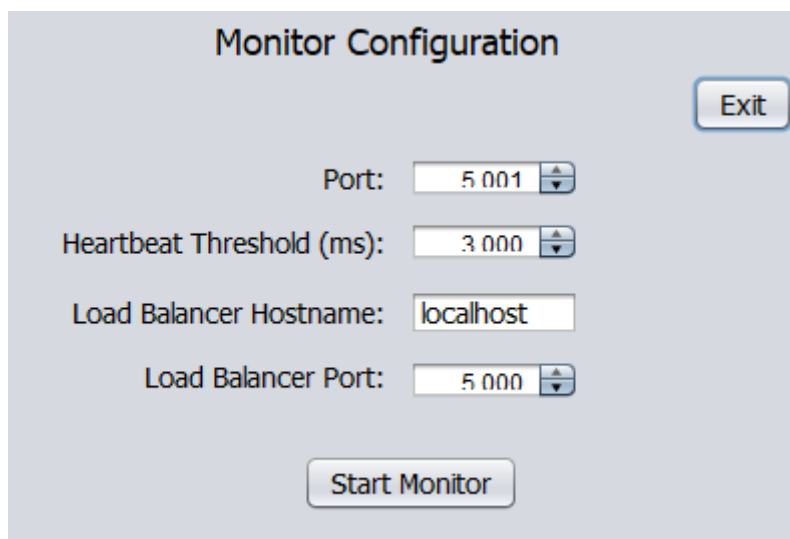
Request	Client	Server	Number of Iterations
Request 1001	Client 1	Server 1	15
Request 1002	Client 1	Server 1	13

Figura 13 - Interface de apresentação de pedidos a ser tratados pelo load balancer

Na interface de listagem dos pedidos, é apresentado para cada pedido o seu ID, o ID do cliente que efetuou o pedido, o servidor para onde o pedido foi atribuído, e ainda o número de iterações.

#### 2.4.2. Interfaces do Monitor

A interface gráfica do monitor é composta pela interface de configuração, apresentada na figura 14, pela interface que apresenta a lista de servidores, apresentada na figura 15, pela interface que apresenta os pedidos que estão a ser geridos pelo *load balancer*, apresentada na figura 16, e pela interface que mostra o estado dos pedidos que um dado servidor está a processar, apresentada na figura 17.



Monitor Configuration

Exit

Port: 5 001

Heartbeat Threshold (ms): 3 000

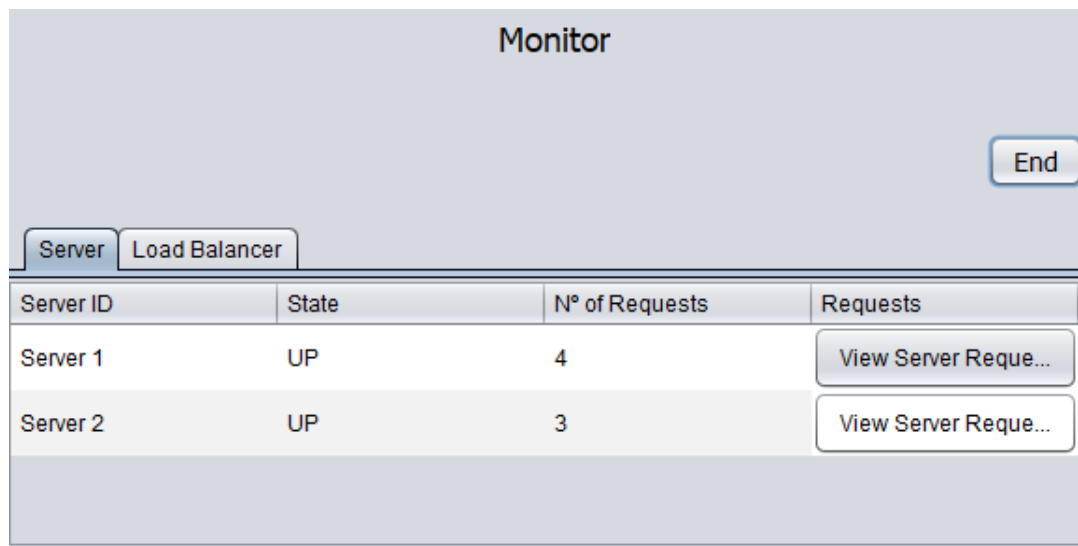
Load Balancer Hostname: localhost

Load Balancer Port: 5 000

Start Monitor

Figura 14 - Interface de configuração do monitor

Na interface de configuração é possível configurar o *port* do monitor, o *host name* e o *port* do load balancer, e ainda o *threshold* para indicar que um servidor está em baixo, em milissegundos.

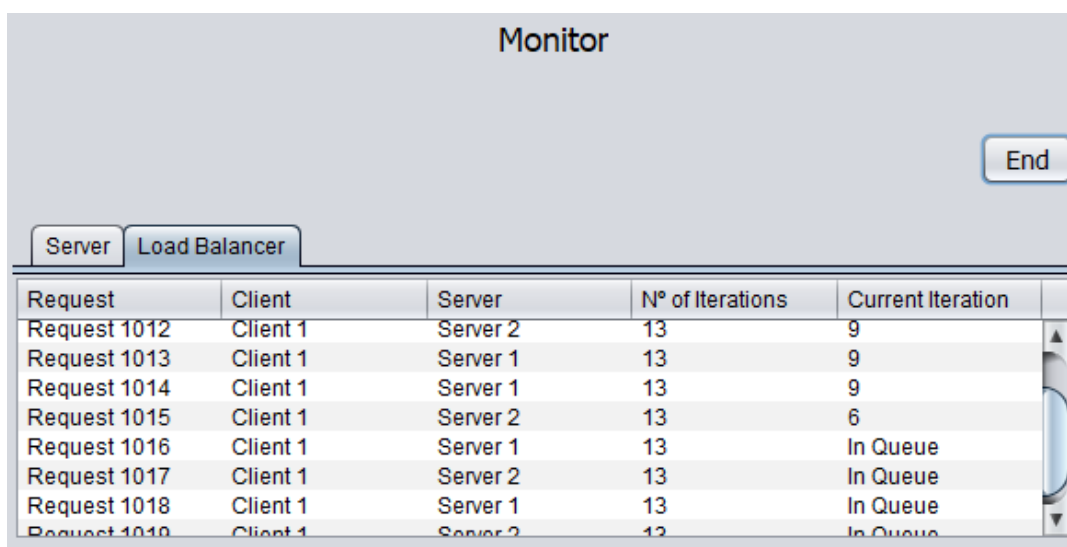


The screenshot shows a web interface titled "Monitor". At the top right is an "End" button. Below the title are two tabs: "Server" (selected) and "Load Balancer". The main content is a table with the following data:

Server ID	State	Nº of Requests	Requests
Server 1	UP	4	<button>View Server Reque...</button>
Server 2	UP	3	<button>View Server Reque...</button>

Figura 15 - Interface com listagem de servidores do cluster

Na interface de listagem de servidores do monitor, para cada servidor é apresentado o seu ID, o seu estado, e o número de pedidos que está a processar no momento.



The screenshot shows the same "Monitor" interface, but with the "Load Balancer" tab selected. The table displays the following data:

Request	Client	Server	Nº of Iterations	Current Iteration
Request 1012	Client 1	Server 2	13	9
Request 1013	Client 1	Server 1	13	9
Request 1014	Client 1	Server 1	13	9
Request 1015	Client 1	Server 2	13	6
Request 1016	Client 1	Server 1	13	In Queue
Request 1017	Client 1	Server 2	13	In Queue
Request 1018	Client 1	Server 1	13	In Queue
Request 1019	Client 1	Server 2	13	In Queue

Figura 16 - Interface com listagem de pedidos a serem geridos pelo load balancer

Na interface de listagem dos pedidos que o load balancer está a processar, é apresentado para cada pedido o seu ID, o cliente que o efetuou, o servidor que está a processar o pedido, o número de iterações, e ainda a iteração atual do processamento.

Monitor			
<input type="button" value="Back"/>			
Server 1 Requests			
Request	Client	Nº of Iterations	Current Iteration
Request 1026	Client 1	13	5
Request 1027	Client 1	13	5
Request 1029	Client 1	13	4
Request 1031	Client 1	13	In Queue
Request 1033	Client 1	13	In Queue

Figura 17 - Interface com o estado dos pedidos a serem processados por um determinado servidor

Na interface de listagem dos pedidos que um servidor está a processar, é apresentado para cada pedido o seu ID, o cliente que o efetuou, o número de iterações, e ainda a iteração atual do processamento.

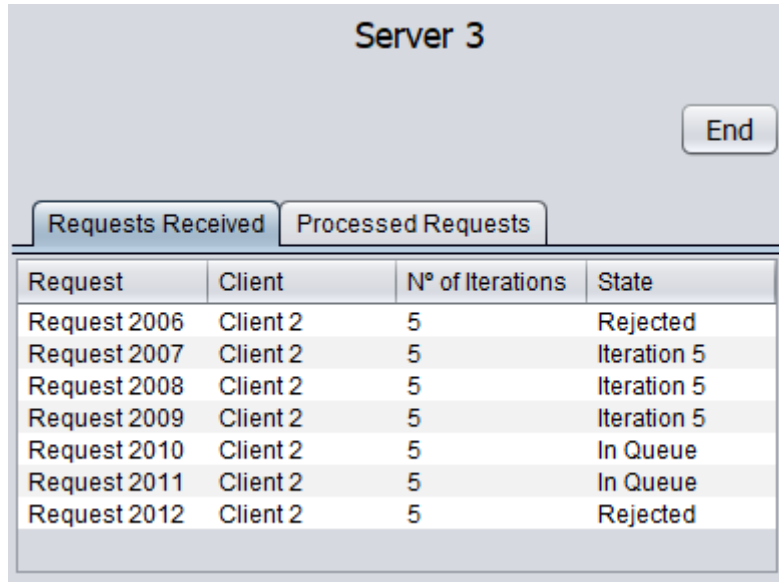
### 2.4.3. Interfaces dos Servidores

A interface gráfica de um servidor é composta pela interface de configuração, apresentada na figura 18, pela interface que apresenta os detalhes dos pedidos recebidos, apresentada na figura 19, e pela interface que apresenta os detalhes dos pedidos processados, apresentada na figura 20.

Server 4 Configuration			
<input type="button" value="Exit"/>			
Queue Size:	<input type="text" value="2"/>	Load Balancer Hostname:	<input type="text" value="localhost"/>
Max Nº of Requests:	<input type="text" value="3"/>	Load Balancer Port:	<input type="text" value="5 000"/>
Iteration Timeout (ms):	<input type="text" value="1 000"/>	Monitor Hostname:	<input type="text" value="localhost"/>
Heartbeat Timeout (ms):	<input type="text" value="1 000"/>	Monitor Port:	<input type="text" value="5 001"/>
<input type="button" value="Start Server"/>			

Figura 18 - Interface de configuração de um servidor

Na interface de configuração de um servidor é possível configurar o *host name* e o *port* do load balancer e do monitor, o tamanho da fila de espera, o número máximo de pedidos que podem ser executados em simultâneo, o tempo de cada iteração, e ainda a periodicidade do *heartbeat*.



**Server 3**

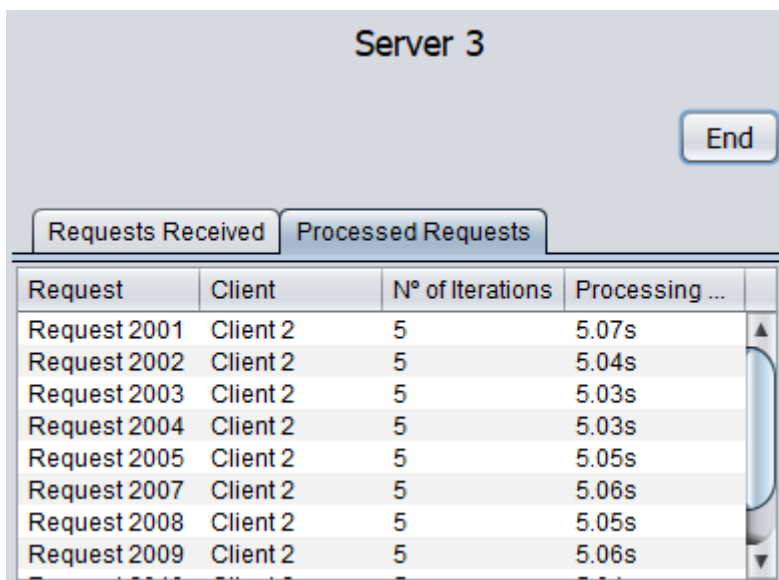
Requests Received    Processed Requests

Request	Client	Nº of Iterations	State
Request 2006	Client 2	5	Rejected
Request 2007	Client 2	5	Iteration 5
Request 2008	Client 2	5	Iteration 5
Request 2009	Client 2	5	Iteration 5
Request 2010	Client 2	5	In Queue
Request 2011	Client 2	5	In Queue
Request 2012	Client 2	5	Rejected

Figura 19 - Interface com listagem dos pedidos recebidos num servidor

Na interface de listagem dos pedidos que um servidor recebeu, é apresentado para cada pedido o seu ID, o cliente que o efetuou, o número de iterações, e ainda o estado do pedido.

Nesta interface apenas são apresentados os pedidos rejeitados, ou os que ainda estão em processamento. Quando um pedido acaba de ser processado, é transitado para a interface que apresenta os pedidos já processados, apresentada na figura 20.



**Server 3**

Requests Received    Processed Requests

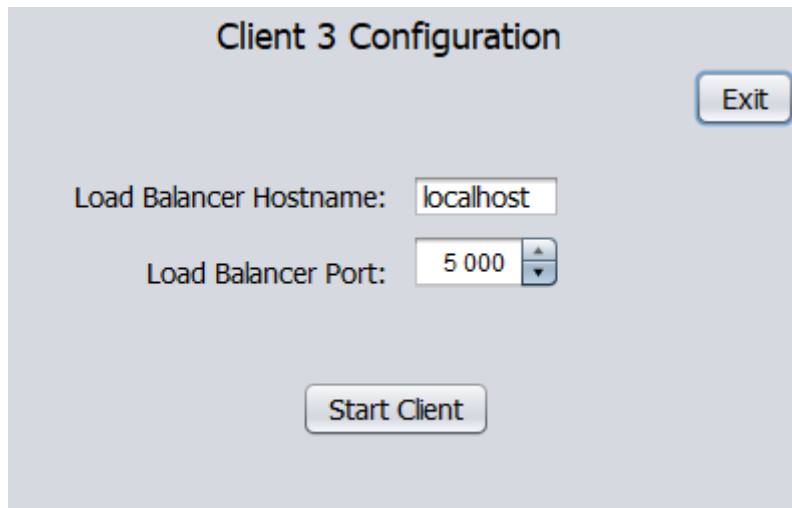
Request	Client	Nº of Iterations	Processing ...
Request 2001	Client 2	5	5.07s
Request 2002	Client 2	5	5.04s
Request 2003	Client 2	5	5.03s
Request 2004	Client 2	5	5.03s
Request 2005	Client 2	5	5.05s
Request 2007	Client 2	5	5.06s
Request 2008	Client 2	5	5.05s
Request 2009	Client 2	5	5.06s

Figura 20 - Interface com listagem dos pedidos processados num servidor

Na interface de listagem dos pedidos que um servidor processou, é apresentado para cada pedido o seu ID, o cliente que o efetuou, o número de iterações, e ainda o tempo de processamento do pedido, em segundos.

#### 2.4.4. Interfaces dos Clientes

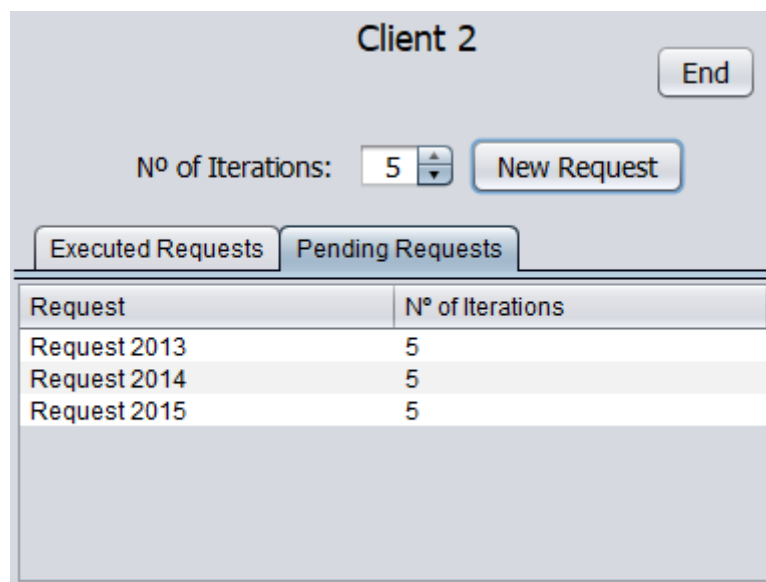
A interface gráfica de um cliente é composta pela interface de configuração, apresentada na figura 21, pela interface que apresenta os pedidos pendentes e que inclui a criação de novos pedidos, apresentada na figura 22, e ainda pela interface que apresenta os pedidos processados e que também inclui a criação de novos pedidos, apresentada na figura 23.



The image shows a window titled "Client 3 Configuration". It has a light gray background. In the top right corner, there is a button labeled "Exit". Below the title, there are two labels: "Load Balancer Hostname:" followed by a text input field containing "localhost", and "Load Balancer Port:" followed by a spin button showing "5 000". At the bottom center, there is a button labeled "Start Client".

Figura 21 - Interface de configuração de um cliente

Na interface de configuração de um cliente é possível configurar o *host name* e o *port* do load balancer.



The image shows a window titled "Client 2". It has a light gray background. In the top right corner, there is a button labeled "End". Below the title, there is a label "Nº of Iterations:" followed by a spin button showing "5". To the right of the spin button is a button labeled "New Request". Below these elements, there are two tabs: "Executed Requests" and "Pending Requests". The "Pending Requests" tab is selected. Below the tabs is a table with two columns: "Request" and "Nº of Iterations". The table contains three rows of data:

Request	Nº of Iterations
Request 2013	5
Request 2014	5
Request 2015	5

Figura 22 - Interface com listagem dos pedidos pendentes e criação de novos pedidos

Na interface de listagem dos pedidos pendentes de um cliente, é apresentado para cada pedido o seu ID e o número de iterações. A interface inclui ainda a criação de novos pedidos, onde é possível configurar o número de iterações do pedido.

**Client 2**

Nº of Iterations:

Request	Nº of Iterations	Reply
Request 2006	5	Rejected
Request 2001	5	$6.02214 \times 10^{23}$
Request 2002	5	$6.02214 \times 10^{23}$
Request 2003	5	$6.02214 \times 10^{23}$
Request 2004	5	$6.02214 \times 10^{23}$
Request 2005	5	$6.02214 \times 10^{23}$
Request 2012	5	Rejected
Request 2007	5	$6.02214 \times 10^{23}$

Figura 23 - Interface com listagem dos pedidos executados e criação de novos pedidos

Na interface de listagem dos pedidos executados de um cliente, é apresentado para cada pedido o seu ID e o número de iterações e a resposta obtida. A interface inclui ainda a criação de novos pedidos, onde é possível configurar o número de iterações do pedido.

#### 2.4.5. Interface do Control Panel

Apesar de não ser obrigatório, foi ainda criada uma interface para gerir as entidades do cluster. Na interface é possível lançar o load balancer, monitor, servidores e clientes, e é possível matar servidores e clientes. A interface é apresentada na figura 24.

**Control Panel**

Server ID	Shutdown
Server 3	<input type="button" value="Shutdown"/>
Server 4	<input type="button" value="Shutdown"/>

Figura 24 - Interface do control panel



### 3. Requisitos Implementados e Distribuição de Tarefas

Todas as tarefas e requisitos presentes no enunciado foram cumpridas na totalidade.

As tarefas foram distribuídas de forma igualitária entre os elementos do grupo. Em relação à implementação das táticas arquiteturais, a distribuição foi:

- **Rafael Sá:** implementação das táticas relativas ao desempenho (*replicas*, *concurrency* e *bound queue size*) e disponibilidade (*redundancy* e *monitor*).
- **Luís Laranjeira:** implementação das táticas relativas à escalabilidade (*horizontal scalability*) e usabilidade (GUIs).

Apesar desta distribuição, existiu uma ajuda mútua de ambas as partes em eventuais problemas que surgiram nas tarefas de cada elemento.

A percentagem de contribuição para o projeto é de 50% para ambos os elementos do grupo.

## 4. Conclusões

É possível verificar que recorrendo às táticas arquiteturais que foram apresentadas, foi possível cumprir com quatro dos mais relevantes atributos de qualidade, nomeadamente o desempenho, a disponibilidade, a escalabilidade e a usabilidade.

Consideramos que a solução desenvolvida vai de encontro às expectativas e cumpre com todos os objetivos propostos para este trabalho.