Sistemas Distribuídos

eVoting: Voto Eletrónico na UC

Projeto Meta 1
Relatório

1. Arquitetura de software detalhadamente descrita

O nosso software consiste em 4 componentes principais:

- Admin Console
- RMI Server
- Multicast Server (Mesa de Voto)
- Multicast Client (Terminal de Voto)

A **Admin Console** conecta-se ao RMI Server por RMI de modo a realizar uma variedade de funções. Para tal, a Admin Console está num ciclo infinito sempre à espera de comandos para saber o que fazer. Este ciclo está numa função que se chama a si própria para facilitar o processo de a admin console se ligar ao RMI Server secundário caso o primário vá abaixo.

O **Multicast Server** corresponde a uma mesa de voto. Como só existe uma mesa de voto em cada departamento, para identificá-la damos-lhe o nome do departamento onde está localizada (ex: Mesa de Voto dei). O Multicast Server tem duas classes que são threads e portanto executam em simultâneo: uma delas usa sockets para receber pacotes que vêm por UDP do Multicast Client e transmite informação pertinente para o RMI Server, outra usa sockets para enviar pacotes por UDP ao Multicast Client sempre que necessário.

O **Multicast Client** corresponde a um terminal de voto. Para identificarmos o terminal de voto associamos-lhe um número aleatório. É a partir de um terminal de voto que um user/utilizador pode votar numa mesa de voto. Tal como no Multicast Server, o Multicast Client tem duas classes que são threads: uma usa sockets para enviar pacotes por UDP ao Multicast Server e outra usa sockets receber pacotes via UDP vindos do Multicast Server, imprimindo toda a informação pertinente ao user/utilizador que se encontra presente na terminal de voto.

O **RMI Server** conecta-se diretamente ao Multicast Server e à Admin Console por RMI. É no RMI Server que se realizam muitas das funções deste projeto, pois é o único servidor que tem acesso à storage, isto é, aos nossos ficheiros de texto, onde guardamos todos os dados e informação relativa a registos de utilizadores, eleições, candidatos, etc. Nós atribuímos o porto 7000 ao nosso RMI Server primário e o porto 6999 ao RMI Server secundário. Sempre que o RMI Server primário vai abaixo, o software tenta ligar-se ao secundário e quando este também falha, o software verifica se o primário já está de volta para se poder ligar a ele caso tal seja possível.

2. Detalhes sobre o funcionamento do servidor Multicast

O Multicast Server, bem como o Multicast Client, comunicam através do protocolo UDP. Este protocolo consiste no uso de sockets para o envio e recepção de mensagens através de pacotes.

Quando queremos enviar uma mensagem passamos essa mensagem para bytes. Um pacote a enviar é da classe DatagramPacket, na sua criação especificamos o endereço Multicast (MULTICAST_ADDRESS) do grupo de endereços IP para onde vamos enviar o pacote, a mensagem a enviar em bytes bem como o seu comprimento e o porto para onde vamos enviar o pacote. Depois, usamos um socket para efetuar o envio, pois este comunica com o porto especificado no pacote. A recepção consiste no mesmo processo mas inverso, isto é, primeiro recebemos o pacote e depois decodificamos a mensagem que vem nele, vendo os dados nele presentes e passando-os depois para uma string.

Para a comunicação entre Multicast Server e Multicast Client usamos dois portos: 4320 e 4321. O porto 4320 dedica-se à transmissão de pacotes do Client para o Server e o 4321 dedica-se à transmissão de pacotes do Server para o Client. Usar dois portos diferentes é necessário, porque se tal não for feito quando, por exemplo, o Server quiser enviar um pacote para o Client ele irá receber também esse pacote, uma vez que a receção e envio de sockets acontece no mesmo porto.

O Multicast Server também comunica por RMI com o RMI Server. Isto acontece sempre quando é necessário passar informação do Multicast Server ou Multicast Client para a storage (ex: adicionar um voto) ou quando é necessário usar informação presente na storage para a realização de alguma função (ex: login).

3. Detalhes sobre o funcionamento do servidor RMI

O RMI é uma interface de programação que permite a execução de chamadas remotas no estilo RPC em aplicações desenvolvidas em Java. O servidor instância objetos remotos e espera por clientes que invoquem os seus métodos, já o cliente referência um ou mais métodos remotos de um objeto remoto.

Relativamente ao nosso projeto, decidimos criar uma classe Java à parte com o nome RMI.java, que contém todos os métodos que serão necessários fazer pela parte da Admin Console e do Multicast Server. Ao criarmos uma variável RMI, tanto no Admin Console como no Multicast Server, estamos a procurar uma conexão com o servidor RMI. Ao lermos uma mensagem, basta chamar os métodos da classe RMI e enviarmos essa mensagem como argumento desse método. As operações são feitas no RMI Server, no qual iremos retornar o resultado dessa operação e enviar esse resultado de volta para a Admin Console ou Multicast Server, como se fossem os clientes do servidor RMI.

Quando um servidor RMI primário se desliga, o Client (Admin Console / Multicast Server) irá ligar-se ao RMI secundário. Este procedimento acontece numa função recursiva, de modo a que quando um se desliga ele volta a executar a mesma função mas com um servidor RMI diferente do que já estava ligado, ficando assim sempre à procura de um servidor disponível.

No Multicast Server, quando ambos os servidores estão desligados, incrementamos numa variável “tempo” os segundos que passam desde que ambos os servidores se desligaram. Até à variável “tempo” chegar aos 30 segundos, estamos sempre a tentar realizar a conexão a um dos servidores.

4. Distribuição de tarefas pelos elementos do grupo

O trabalho foi realizado em conjunto pelos membros do grupo, e portanto, ambos os membros trabalharam em todas as tarefas. Apenas a realização de poucas funções no RMIServer.java foi exclusiva a cada um dos membros.

5. Descrição dos testes feitos à plataforma

Descrição	Pass/Fail	Testes Realizados
Registar novo utilizador (estudante, docente, ou funcionário)	Pass	-O utilizador é registado. -Verificar se o utilizador já existe.
Criar eleição	Pass	-A eleição é criada. -Verificar se já existe uma eleição com o mesmo ID.
Gerir listas de candidatos a uma eleição	Pass	-Registar um usuário como candidato. -Verificar se já é um candidato dessa eleição para não haver repetidos. -Verificar se o usuário cumpre os requisitos. -Verificar se o candidato já está registado.
Criar mesas de voto	Pass	-Registar mesa de voto. -Verificar se essa mesa de voto já está a ser usada para eleição.
Gestão automática de terminais de voto, por Multicast	Pass	-Conectar vários terminais de voto a uma mesa de voto.
Identificar eleitor na mesa de voto e desbloquear um terminal de voto	Pass	-Verificar que apenas um Terminal de voto recebe a mensagem de “desbloqueado”, e não é repetida por outros terminais.
Login de eleitor no terminal de voto	Pass	-Verificar se o utilizador já fez login. -Verificar se o utilizador está registado. -Verificar se a palavra passe está correta. -Verificar se o utilizador já iniciou a sessão antes de selecionar a eleição.
Votar (escolher, uma só vez, uma lista no terminal de voto)	Pass	-Verificar se o utilizador já selecionou a eleição antes de votar. -Verificar se o utilizador cumpre os requisitos antes de votar na eleição. -Verificar se o utilizador já votou.

Editar propriedades de uma eleição	Pass	-Verificar se a eleição existe. -Altera as propriedades da eleição. -Verificar se a eleição já começou.
Saber em que local votou cada eleitor	Pass	-Mostrar qual a mesa de voto que o eleitor votou.
Mostra mesas de voto on/off e votantes	Pass	-Mostra as mesas de voto que estão abertas e os terminais de voto abertos.
Console de administração mostra em tempo real.	Fail	-Mostrar em tempo real que uma mesa de voto está on/off.
Mostra o número de Pessoas que votaram numa mesa de voto até um certo momento.	Pass	-Verificar se a mesa de voto existe. -Mostra o número de pessoas que votaram.
Consola de administração mostra a informação em tempo real.	Fail	-A consola de administração mostra o número de pessoas que votaram.
Eleição termina corretamente na data, hora e minuto marcados	Pass	-Verificar se a eleição já terminou, antes de votarmos.
Consultar resultados detalhados de todas as eleições passadas	Pass	-Verificar se a eleição já terminou. -Mostra resultados.
Avaria de um servidor não tem efeito sobre os clientes.	Pass	-Desligamos o servidor primário, e o multicast server liga-se automaticamente ao secundário. -Os clientes podem continuar a usar o terminal de voto.
Perda de votos não acontece.	Fail	-Envia uma mensagem a dizer “Try again, please” de modo a tentar evitar perdas de voto
Duplica votos não acontece.	Pass	-Nunca aconteceu, pois é registado no fim de um voto ter sido realizado.
Avarias temporárias < 30s são invisíveis.	Pass	-O Terminal de voto continua a ser usado.
Durante esses 30 segundos, se um dos servidores ligar, o terminal liga-se a ele.	Pass	-O Terminal de voto volta a reconectar.
Envio de notificação a dizer que os servidores não ligaram após 30 segundos.	Pass	-O Terminal de voto recebe a notificação.
Os dados são os mesmos para ambos os servers.	Pass	-Está tudo registado em ficheiros, e cada um tem acesso de igual forma aos ficheiros.
O failover é invisível para clientes/eleitores.	Pass	-Os clientes não perdem a sessão.
O servidor principal quando recupera torna-se secundário	Pass	-Ao ligar o terminal ao secundário, ligamos o primário, desligamos o secundário, e ele volta a reconectar-se ao primário apenas depois de o secundário se desligar.

Terminal de voto bloqueado após 120 segundos sem uso.	Fail	-O Terminal de voto não é bloqueado após 120 segundos.
O RMI primário envia pings para o RMI secundário.	Fail	-Não achamos necessário segundo a nossa implementação.

Linhas a verde = Requisitos Funcionais
Linhas a amarelo = Exceções e Failover

Trabalho realizado por:

- Hugo Jordão Nº2018285733
- Luís Loureiro Nº2018297934