eVoting: Voto Eletrónico na UC

Sistemas Distribuídos 2020/21 — Meta 1 — 1 de abril de 2021 (23:59)

Resumo

O *voto eletrónico* consiste na obtenção, armazenamento e contagem de votos, por via eletrónica, relativos a um processo eleitoral. Numa eleição apresentam-se candidatos e a escolha de cada votante deverá ser secreta, sendo que cada eleitor poderá votar apenas uma vez, e a contagem de votos deverá estar de acordo com as escolhas realmente feitas pelos votantes. Este projeto consiste em criar um sistema de voto eletrónico para eleger as direções de organismos da Universidade de Coimbra.

1 Objetivos do projecto

No final do projeto cada estudante deverá ter:

- programado um sistema de voto eletrónico, com arquitetura cliente-servidor;
- seguido um modelo multithreaded para desenhar os servidores;
- aplicado sockets para comunicação entre clientes e servidores;
- criado uma camada de persistência de dados usando Java RMI;
- garantido a disponibilidade da aplicação através de redundância com failover.

2 Visão geral

As eleições universitárias têm como objetivo eleger as direções de organismos da universidade. Uma eleição é iniciada por uma comissão eleitoral, que administra todo o processo eleitoral e decide o momento em que a eleição vai terminar (data, hora e minuto). Os eleitores votam nas listas candidatas até ao término da eleição. No final faz-se o apuramento de resultados.

Grupos de 2 alunos devem considerar eleições de formato simples, nas quais existam listas candidatas compostas por membros da comunidade académica. Por exemplo, eleições para núcleos de estudantes compostas por listas de estudantes e nas quais possam votar apenas estudantes.

Neste projeto considera-se o voto eletrónico como meio de realização de eleições. No momento da eleição, um eleitor dirige-se a uma **mesa de voto** onde é identificado e autorizado a votar. Seguidamente, é-lhe indicado um **terminal de voto** onde poderá fazer a sua escolha secreta. No final da eleição realiza-se a contagem de votos no **servidor central replicado**. São estes os 3 componentes fundamentais da arquitetura.

3 Funcionalidades a desenvolver

Os utilizadores da aplicação são os *eleitores* que interagem apenas com os terminais de voto, os *administradores* (comissão eleitoral) que usam a consola de administração, e os *membros das mesas* que usam a máquina instalada em cada mesa de voto. Deverá ser possível realizar as seguintes operações:

1. **Registar pessoas.** Deve poder-se registar estudantes, docentes e funcionários. Deverá guardar toda a informação pessoal que considere necessária, bem como uma password (código de acesso) e o departamento/faculdade ao qual a pessoa pertence. A informação pessoal deverá incluir também dados de contacto telefónico, morada, número e validade do cartão de cidadão.

2. **Criar eleição.** Uma eleição tem um momento (data, hora e minuto) em que começa e outro em que termina. Tem igualmente um título e uma descrição sucinta. Deve ser possível, de uma forma simples, restringir o grupo de pessoas que pode votar na eleição. Por exemplo, eleições para núcleo de estudantes decorrem num único departamento e podem votar apenas os estudantes desse departamento.

3. **Gerir listas de candidatos a uma eleição.** Uma candidatura a uma dada eleição é um conjunto ordenado de pessoas, designado por lista. Existem listas separadas de estudantes, docentes e funcionários. Uma eleição para núcleo de estudantes tem apenas como candidatas listas de estudantes.

4. **Gerir mesas de voto.** Deve ser possível adicionar e remover mesas de voto associadas a uma dada eleição. Uma mesa de voto é composta por uma máquina onde se identifica eleitores antes de votarem, e um ou mais terminais de voto nos quais os eleitores escolhem a lista em que querem votar. Uma mesa de voto necessita apenas da indicação do departamento no qual está localizada (no máximo existirá uma mesa em cada departamento). Não é obrigatório que todos os departamentos tenham mesa de voto (pode haver uma mesa no DEI para servir todo o pólo 2, por exemplo). Cada mesa de voto terá o(s) seu(s) grupo(s) de multicast próprio(s).

5. **Gerir terminais de voto.** Cada mesa de voto pode ter diversos terminais de voto associados. Tal como se descreve mais à frente, a comunicação e gestão destes terminais será feita inteiramente por Multicast. Esta gestão deverá ser automática, bastando configurar o grupo de Multicast para que as máquinas se descubram e identifiquem. Uma alternativa é utilizar dois grupos de multicast distintos: um para descoberta de máquinas e outro para comunicação de votos dos terminais para o servidor da mesa.

6. **Identificar eleitor na mesa de voto.** O primeiro passo para que uma pessoa possa votar consiste em pesquisar por qualquer campo por forma a identificar esse eleitor. Ao identificar um eleitor, um dos terminais de voto fica desbloqueado para que essa pessoa possa ir votar secretamente (um terminal que esteja livre deve ser escolhido usando as funcionalidades do Multicast). Deve poder-se listar e escolher uma eleição, caso haja mais do que uma eleição em simultâneo. O terminal de voto volta a ficar bloqueado após 120 segundos sem uso.

7. **Autenticação de eleitor no terminal de voto.** Após a identificação, um eleitor dirige-se ao terminal de voto que tiver sido desbloqueado e apresenta o seu username/password para obter acesso ao "boletim de voto", ou seja, à enumeração das listas candidatas à eleição em que se irá votar.

8. **Votar.** Cada eleitor pode votar no máximo uma vez por eleição, escolhendo no terminal de voto uma das seguintes opções: uma das listas candidatas, voto em branco ou voto nulo. É fundamental que cada voto seja secreto, que cada eleitor vote apenas uma vez nas eleições em que estiver autorizado, e que todos os votos sejam contados corretamente no final. Qualquer pessoa pode votar em qualquer mesa, mesmo que seja noutro departamento que não o seu.

9. **Alterar propriedades de uma eleição.** As propriedades textuais de cada eleição devem poder ser editadas, e os instantes de início e de fim da eleição devem poder ser alterados. As alterações devem apenas poder ser feitas antes de começarem as eleições.

10. **Saber em que local votou cada eleitor.** Por uma questão de auditoria, deve ser possível saber em que mesa de voto e em que momento votou cada eleitor.

11. **Consolas de administração mostram o estado das mesas de voto.** As consolas de administração que estejam ligadas à aplicação recebem imediatamente (em tempo real) notificações relativamente às mesas de voto (e respetivos terminais) que estejam a funcionar corretamente ou não estejam em funcionamento.

12. **Consolas de administração mostram eleitores em tempo real.** As consolas de administração que estejam ligadas à aplicação recebem, em tempo real, o número de eleitores que votaram até ao momento em cada mesa de voto (numa dada eleição).

13. **Término da eleição na data, hora e minuto marcados.** No momento indicado nos detalhes de uma eleição (data, hora e minuto) o processo eleitoral termina au-

tomaticamente. Realiza-se aí o apuramento do número de votos por lista e fecha-se a possibilidade de efetuar mais votos. O resultado de uma eleição é o número de votos obtidos por cada lista candidata, os votos brancos e os votos nulos. Os detalhes dessa eleição são atualizados e podem ser consultados posteriormente.

14. **Consultar resultados detalhados de eleições passadas.** Os resultados finais de todas as eleições (que já tenham terminado) devem poder ser consultados. Estes resultados incluem o número (absoluto e em percentagem) de votos de cada lista candidata, bem como o número e percentagem de votos em branco.

4 Arquitetura

A Figura 1 mostra a arquitetura global do projeto. Cada grupo deverá programar os *servidores RMI*, que são idênticos embora um seja inicialmente primário e outro secundário. Cada grupo deverá igualmente programar os *servidores Multicast*, que são idênticos salvo questões de configuração, assim como os terminais de voto (clientes Multicast). Cada grupo deverá também programar uma *consola de administração*, à parte, que se liga aos servidores RMI e permite à comissão eleitoral gerir eleições.

O sistema deverá aceitar qualquer número de terminais de voto, que são os clientes Multicast. Será disponibilizado um cliente Multicast em Java que poderá ser adaptado pelos alunos. Pretende-se com isto que os terminais de voto sejam muito simples, ainda que na prática se aceite outras soluções. Assim, deverão ser criados quatro classes:

- RMI Server – É o servidor central (replicado) que armazena todos os dados da aplicação, suportando por essa razão todas as operações necessárias através de métodos remotos usando Java RMI.

- Multicast Server – Existe um Multicast Server por cada mesa de voto que gere localmente os terminais de voto que lhe estão associados. Permite aos membros

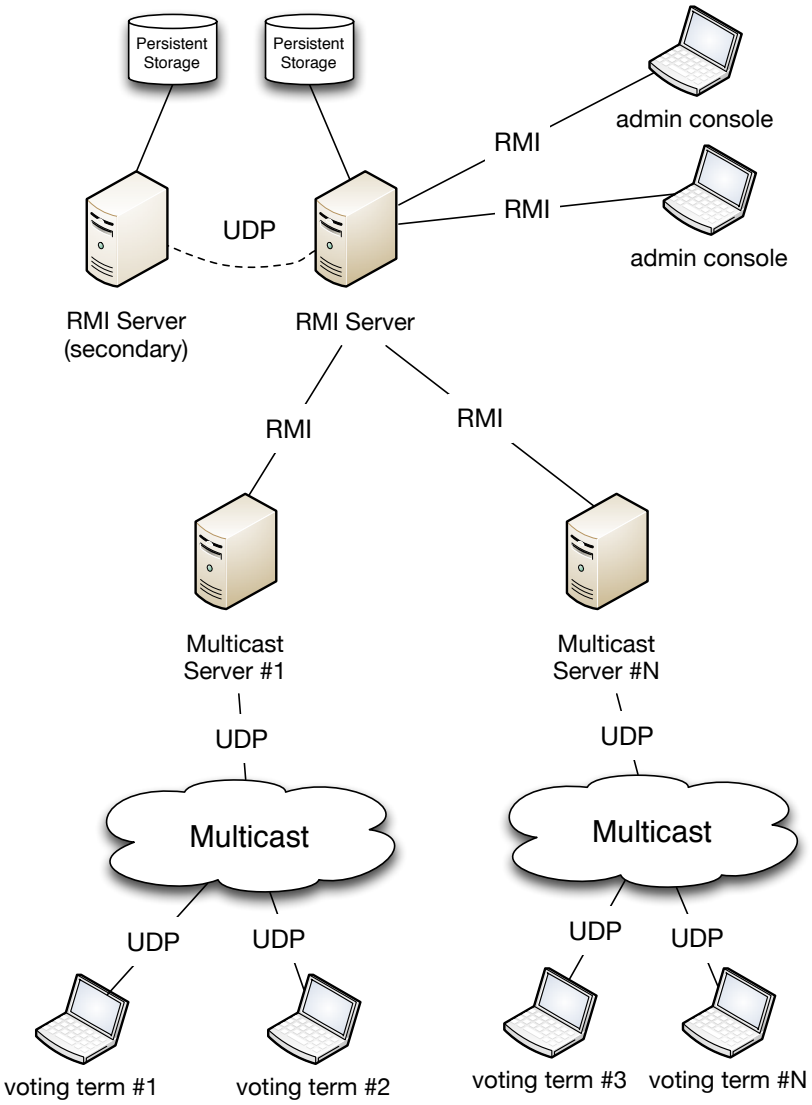

Fig. 1: Arquitetura do projeto.

da mesa realizar a funcionalidade 6 e realiza automaticamente a funcionalidade 5.

- Voting Terminal – São os clientes Multicast que estão associados a cada mesa de voto, e que permitem realizar as funcionalidades 7 e 8. Recomenda-se que o terminal de voto utilize dois grupos de Multicast distintos: um para descobrir e ser descoberto pelo servidor (por exemplo, quando o servidor precisa de um terminal livre) e outro grupo apenas para comunicar a intenção de voto do eleitor.
- Admin Console – Podem existir uma ou mais consolas de administração, que permitem realizar as funcionalidades 1–4 e 9–18. Estas consolas comunicam apenas através de Java RMI.

5 Protocolo Multicast

De forma a permitir a implementação de clientes eVoting para qualquer plataforma (seja Windows, Linux, macOS, Android, iOS), o protocolo deverá ser especificado em detalhe. Segue-se uma recomendação da base de um protocolo a ser completado pelos estudantes. A estrutura principal deste protocolo é a mensagem que consiste num conjunto não ordenado de pares chave-valor, semelhante a um HashMap em Java ou a um dicionário em Python, finalizados com uma mudança de linha. Um exemplo seria:

```
chave1 | valor1; chave2 | valor dois
```

Como tal, não é permitido ter o caracter ponto e vírgula (;), nem o caracter mudar de linha, nem o caracter "pipe" (|) no nome da chave nem no valor.

Todas as mensagens têm um campo obrigatório chamado "type". Este valor permite distinguir o tipo de operação que o cliente pretende fazer no servidor, ou o tipo de resposta que o servidor está a dar. Um exemplo mais realista é portanto:

```
type | login; username | pierre; password | omidyar
```

E a resposta respetiva, sendo que o campo msg é opcional, mas aceite.

```
type | status; logged | on; msg | Welcome to eVoting
```

Finalmente, para representar listas de elementos é usado o tamanho e o contador de elementos. O tamanho é descrito num campo com o sufixo x_count, onde x é o campo com a lista, e cada campo do elemento tem o formato x_i_campo, onde i é o índice da lista, a começar em 0. Um exemplo encontra-se de seguida:

```
type | item_list; item_count | 2; item_0_name | A Candidate
   List; item_1_name | Another Candidate List
```

Este protocolo deverá ser completado e especificado pelos alunos no decorrer do projeto.

6 Atributos de qualidade

A aplicação deverá lidar corretamente com quaisquer exceções que estejam previstas. Por forma a garantir que uma eleição pode sempre decorrer com normalidade, deverá usar uma solução de failover para garantir que a aplicação continua a funcionar mesmo que um servidor avarie.

6.1 Tratamento de exceções

Como o hardware pode falhar, é necessário que os utilizadores não notem nenhuma falha de serviço. Como tal, no caso do servidor RMI falhar, é preciso garantir que os votos efetuados não se percam, nem apareçam duplicados. No caso das avarias temporárias (inferiores a 30 segundos), os clientes não se devem aperceber desta falha.

Também do lado dos clientes é possível que a ligação se perca a meio. É necessário garantir que nenhuma falha do lado do cliente deixe nenhuma operação a meio.

Os terminais de voto podem crashar e deve bastar reiniciá-los para que se possa depois continuar a votar.

6.2 Failover

De modo a que quando o servidor RMI falhar, o servidor secundário o substitua, é necessário ter alguns cuidados. Em primeiro lugar o servidor secundário deve trocar periodicamente, via UDP, pings ou heartbeats com o servidor primário. Se algumas destas mensagens se perderem (por exemplo cinco pings seguidos) o servidor secundário assume que o primário avariou e liga-se para o substituir. Outros mecanismos equivalentes serão aceites.

Os servidores Multicast também devem detetar quando existe uma falha permanente do servidor RMI e devem tentar ligar-se ao secundário. Dado o uso de persistência do lado do RMI, os dados visíveis pelos servidores Multicast devem ser exatamente os mesmos. Do lado dos clientes, todo este processo deve ser transparente. Para eles esta falha nunca deverá ter qualquer efeito visível.

Finalmente, se o servidor RMI original recuperar, deverá tomar o papel de secundário e não de primário.

6.3 Relatório

Devem reservar tempo para a escrita do relatório no final do projeto, tendo em conta os passos anteriores. Devem escrever o relatório de modo a que um novo colega que se junte ao grupo possa perceber a solução criada, as decisões técnicas efetuadas e possa adicionar novos componentes ou modificar os que já existem. **O relatório pode ser inteiramente escrito em Javadoc no código-fonte apresentado pelos estudantes.** O relatório deve incluir:

- Arquitetura de software detalhadamente descrita. Deverá ser focada a estrutura de threads e sockets usadas, bem como a organização do código.
- Detalhes sobre o funcionamento do servidor Multicast. Deve especificar detalhadamente o protocolo usado para comunicação Multicast.
- Detalhes sobre o funcionamento do servidor RMI. Deverá explicar detalhadamente o funcionamento dos métodos remotos disponibilizados e eventuais callbacks usados, bem como a solução usada para failover.
- Distribuição de tarefas pelos elementos do grupo.
- Descrição dos testes feitos à plataforma (tabela com descrição e pass/fail de cada teste).

6.4 Distribuição de tarefas

De modo a que a avaliação seja justa num trabalho de grupo, é fundamental uma divisão de trabalho justa. Dado que a nota resultante da defesa será individual, são propostas as duas possíveis divisões de trabalho:

- Elemento 1 será responsável pelo RMI Server e o elemento 2 pelo Multicast Server e pela Admin Console. Esta divisão assume que a interface RMI é inteiramente especificada inicialmente e as suas alterações serão daí em diante mínimas. No caso de existir um terceiro elemento, este ficaria responsável pelo Multicast Server ou pela Admin Console (isto é, divide trabalho com o elemento 2).
- Cada um dos elementos ficará com igual número de funcionalidades a implementar.

Finalmente, poderão ser aceites outras distribuições que sejam justas.

7 Planos futuros para o projeto

Na segunda meta do projeto irão expandir a presente solução, adicionando uma interface Web usando Spring/Struts2/JSP e irão integrar a aplicação com uma API REST de um serviço externo. Nessa fase, o servidor Web irá usar a API do servidor RMI aqui criado.

8 Entrega do projeto

O projeto deverá ser entregue num arquivo ZIP. Esse arquivo deverá conter um ficheiro README.TXT com toda a informação necessária para instalar e executar o projeto sem a presença dos alunos. Projetos sem este ficheiro, sem informações suficientes, que não compilem ou não executem corretamente **não serão avaliados**.

Dentro do ficheiro ZIP deverá também estar um PDF com o relatório. O relatório deve seguir a estrutura fornecida, dado que a avaliação irá incidir sobre cada um dos pontos.

Também no ficheiro ZIP deverão existir quatro ficheiros JAR: server.jar (servidor Multicast), terminal.jar (terminal de voto), rmiserver.jar (servidor RMI) e console.jar (consola de administração).

Finalmente, o ficheiro ZIP deverá ter também uma pasta com o código fonte completo do projeto. A ausência deste elemento levará à **não avaliação do projeto**.

O ficheiro ZIP deverá ser entregue na plataforma inforestudante até ao dia 1 de abril de 2021 (23:59), via `http://inforestudante.uc.pt`.