Fundamentos de Sistemas Distribuídos

Trabalho Prático

Relatório de Desenvolvimento

Miguel Oliveira	Pedro Moura	César Silva
pg41088	pg41094	pg41842

Universidade do Minho,

29 de Dezembro de 2019

Resumo

Este relatório descreve o desenvolvimento de um projeto no âmbito da UC de Fundamentos de Sistemas Distribuídos, onde, através de ferramentas e bibliotecas apresentadas nas aulas, são aplicadas técnicas para um bom e, sobretudo, fiável funcionamento de um sistema distribuído.

Conteúdo

Capítulo 1

Introdução

Com o objetivo de aplicar os conhecimentos adquiridos nas aulas relativas à UC de Fundamentos de Sistemas Distribuídos, foi-nos proposto o desenvolvimento de um sistema de troca de mensagens com persistência e ordenação, inspirado na rede social *Twitter*. O enunciado apresenta alguns requisitos que devem ser respeitados:

1. O sistema deve incluir um conjunto de servidores, que se conhecem todos entre si. Admite-se a possibilidade de um destes servidores ser reiniciado, devendo garantir que o sistema continua a funcionar depois de todos os servidores estarem novamente operacionais. O servidor não deve ter qualquer interação direta com o utilizador.

2. O sistema deve incluir clientes que se ligam a qualquer um dos servidores. Admite-se que o cliente pode ser reiniciado e ligado a um novo servidor. O cliente deve incluir uma interface rudimentar para interagir com o sistema, que deve ter algumas funcionalidades.

3. Admite-se que tanto os clientes como os servidores podem fazer uso da memória persistente.

4. O conjunto de mensagens obtido por cada cliente em cada operação deve refletir uma visão causalmente coerente das operações realizadas em todo o sistema, por esse ou outros utilizadores.

Neste primeiro capítulo foi feita a contextualização e foram apresentados os objetivos deste projeto. De seguida, apresentamos uma proposta de solução, onde são descritas tanto as abordagens seguidas bem como as decisões tomadas para atingirmos os objetivos. No terceiro capítulo, descrevemos como chegamos à proposta de solução apresentada, com detalhes mais técnicos. Posto isto, no quarto capítulo, apresentamos o resultado final da aplicação acompanhados com alguns testes realizados. Por fim, no quinto e último capítulo, sumarizamos o que foi escrito no relatório, e a nossa satisfação global com o projeto.

Capítulo 2

Proposta de Solução

2.1 Sistema

Muito abstratamente, o sistema é composto por um conjunto de servidores que comunicam entre si através de mensagens (principalmente de estado), e por clientes que se ligam a estes servidores.

Um bom e consistente funcionamento do sistema depende de uma igualmente boa capacidade de organização e coordenação por parte dos servidores, o que requer a utilização de algumas técnicas e propriedades, nomeadamente:

- *Leader election* - é o processo de eleger um servidor como o organizador das tarefas que são distribuídas por todos os servidores. É essencial para o sistema adquirir a capacidade de organização desejada.
- *Causal delivery* - é uma abstração usada para garantir que as mensagens são enviadas numa ordem que respeita a relação *happened-before*, que diz que se um evento deve acontecer antes de outro, o resultado destes eventos também deve refletir isso. Esta propriedade é o que permite o sistema ter informação consistente e ser coordenado.
- *Two phase commit* - **ainda não sei se é para meter**

Os servidores comunicam apenas por mensagens, logo é com estas que o sistema tem de conseguir implementar as técnicas referidas. De modo a que seja possível conseguir diferentes fins, é necessária a existência de diferentes tipos de mensagens, como iremos ver a seguir.

Outro aspeto fundamental no sistema é a relação servidor-cliente. Um servidor tem de ser capaz de hospedar diversos clientes, comunicar com eles, e alterar o seu estado consoante os pedidos efetuados por eles.

2.1.1 Mensagens

As mensagens enviadas entre servidores têm 2 campos.

- *Vector Clock* - é um vetor de N relógios lógicos (um relógio por servidor). Com este vetor é possível implementar o algoritmo dos *vector clocks* que por sua vez assegura a *causal delivery* referida acima.
- *Conteúdo* - o conteúdo da mensagem em si.

Como foi referido anteriormente, as mensagens têm diferentes funcionalidades e, consoante estas, estão divididas em diferentes tipos.

2.1.1.1 Mensagens de estado

Sempre que um servidor efetua uma alteração no estado, é suposto notificar os outros servidores de modo a que estes se atualizem, e assim se garanta a consistência de informação. As mensagens de estado são as responsáveis por esta notificação, de tal forma que o conteúdo que é enviado é o estado do servidor após as alterações.

Os outros servidores, ao receberem uma mensagem deste tipo, extraem o conteúdo e atualizam o seu próprio estado consoante este conteúdo.

2.1.1.2 Mensagens para a *Leader Election*

No sistema, o líder é o processo com o maior *id*.

Cada servidor começa por enviar uma mensagem a todos os outros com o seu *id*. Quando recebidas todas as mensagens, os servidores pegam no maior *id* recebido, e elegem-no como líder.

2.1.1.3 Heartbeat

Falar do heartbeat

2.2 Servidor

- falar do estado, atualizações etc
- responsável pela conexão dos clientes
- tipos de mensagens que envia aos clientes

2.3 Cliente

- Coneta-se a um servidor
- falar das funcionalidades
- POST e GET requests

Capítulo 3

Desenvolvimento

3.1 Tecnologia

Falar do java, intellij e atomix

3.2 Implementação

Falar do código

Capítulo 4

Resultado

- Testes
- Resultado

Capítulo 5

Conclusão

- Discussão dos resultados
- Pros/contras da abordagem seguida
- Como poderíamos evoluir o que foi construído
- Sempre com enfoque que foi feito tudo o que foi pedido