TÉCNICO LISBOA

Projeto de Sistemas Distribuídos

2ª Parte - Fault Tolerance

Grupo A48

SDis12645111326L11 Qui 9:30 - 11:00 - Prof. Tomás Cunha

Nome	Sérgio Nóbrega	Margarida Morais
Número	86806	86473
Github	sernobrega	MargaridaMorais

Repositório Git: https://github.com/tecnico-distsys/A48-ForkExec

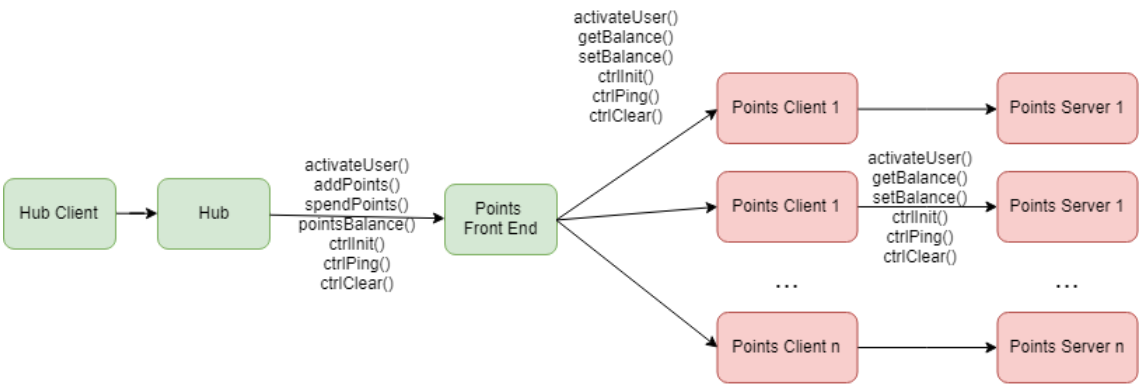
Definição do Modelo de Faltas

Apesar de o sistema **ForkExec** ser constituído por mais do que uma componente, e qualquer uma delas estar sujeita a faltas, no âmbito do projeto de Sistemas Distribuídos, apenas vai ser considerado o módulo **Points**. Assim sendo, no *modelo de faltas*, apenas irá estar presente o servidor de pontos, cujas faltas irão ser toleradas.

Assim, para esta parte do projeto foi assumido o seguinte modelo de faltas:

1. As réplicas podem podem falhar silenciosamente;
2. As mensagens podem perder-se na rede;
3. Os componentes de gestor de réplica e réplicas podem falhar;

Diagrama da Solução de Tolerância a Faltas



Descrição do diagrama

A figura apresentada mostra a conceptualização da solução implementada pelo nosso grupo. Foi criado um front end para o hub, neste caso, o Points Front End. É através do Front End que são feitos os pedidos do Hub para as réplicas de servidores Points. Depois do pedido ser enviado para o Front End, este traduz o pedido em operações permitidas pelo Points Server e envia uma mensagem a todos os gestores de réplica, ou seja os Points Client que por sua vez enviam as mensagens às réplicas correspondentes. O Front End foi desenhado como sendo parte do domínio do Hub porque sendo o Points um serviço independente, assumiu-se que não deve conter assunções sobre as necessidades específicas de um outro serviço (neste caso o Hub). Assim, por exemplo, um outro serviço (não considerado) pode aceitar fazer setBalance de valores negativos ou outros, algo que seria limitado caso o Front End fosse do domínio do Points.

Explicação da solução

A solução implementada consiste num protocolo de replicação ativa, mais precisamente, o protocolo *Quorum Consensus*. A implementação do protocolo, é feita no Points Front End, nos métodos **getBalance** e **setBalance** (homónimos dos métodos que chamam dos gestores de réplicas) assim como, num caso particular, o **activateUser**. Os métodos ctrlClear, ctrlInit e ctrlPing são chamados sincronamente por serem apenas operações de controlo.

O Front End atua como *middleman* entre o hub-ws e o points-ws-cli, fazendo a tradução dos pedidos do hub-ws em lógica permitida pelos métodos do points-ws-cli.

Em geral, quando o front end quer fazer um request aos servidores, é enviada uma **chamada assíncrona** do respetivo método a cada um dos gestores de réplica. Para as chamadas assíncronas, decidimos utilizar **polling** de forma a melhor segmentar o código e ficar de mais fácil compreensão.

Uma única thread faz a gestão de todos os pedidos a todos os servidores isto significa que existe um ***for cycle*** que itera num mapa com as respostas até obter o *quorum* (2/N + 1). A decisão de fazer single thread dá-se ao facto de o número de servidores a considerar ser pequeno (<100).

Consideramos que a rede tem baixa fiabilidade mas assumimos também a simplificação dada no enunciado de que há sempre o mínimo de réplicas possíveis para satisfazer o *quorum* pelo que sempre que um pedido falha, é reenviado sem nenhum número máximo de tentativas ou *timeout* já que temos a garantia de que N/2+1 réplicas estão funcionais.

Semântica Utilizada

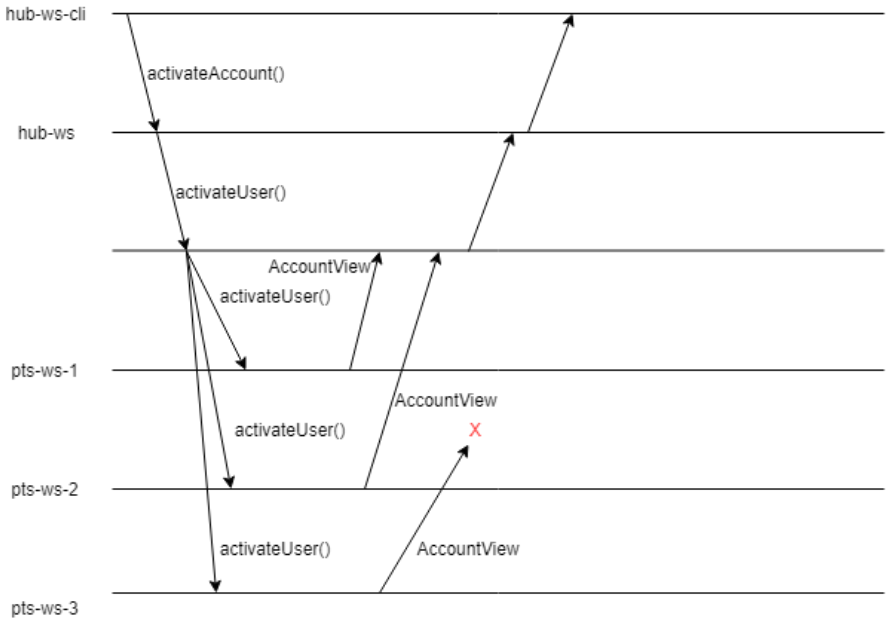
A semântica utilizada para a elaboração do projeto é a **No-Máximo-1-Vez**. Isto é permitido pelo facto de tanto o setBalance() como o getBalance() serem idempotentes. Assim, fazem-se *queries* até obter uma resposta ou até obter o quorum através de outros servidores.

Descrição de otimizações/simplificações

1. Uma simplificação foi retirar o clientId da tag, passando a ser constituído apenas por um número de sequência dado pelo Points Front End. Isto é possível pois sabemos, de certeza, que o Points Front End é o único cliente a enviar pedidos aos servidores.
2. Como dito acima, sabendo que existe uma minoria de gestores de réplica em falha em simultâneo, as operações que requerem quorum fazem queries infinitamente pois sabem que eventualmente vão obter o quorum para retornar o resultado.
3. Foi criada uma cache através de um WeakHashMap que guarda os valores resultantes de getBalance e setBalance e permitem que algumas queries de getBalance não sejam necessárias.

Troca de Mensagens

Abaixo temos dois diagramas ilustrativos da troca de mensagens entre os diferentes componentes do projeto.



No exemplo abaixo assumimos que o balance do utilizador não se encontra em cache, pelo que o getBalance() tem de ser feito. Podemos também reparar que o Front End faz setBalance() antes de receber a resposta de pts-ws-3, devido ao quorum consensus (apenas precisa de 2 respostas).