Relatório da segunda entrega do Projeto ForkExec

Licenciatura em Engenharia Informática e de Computadores

Unidade Curricular de Sistemas Distribuídos

2018/2019 – Campus Taguspark

Grupo T22

João Freitas

87671

Pedro Soares

87693

Repositório Github:

https://github.com/tecnico-distsys/T22-ForkExec

Definição do modelo de faltas:

Como modelo de faltas assumimos que:

- Os gestores de replica podem falhar silenciosamente, mas não arbitrariamente, como tal, não existem falhas bizantinas.
- No máximo, existe uma minoria de gestores de réplica em falha em simultâneo.
- O sistema é assíncrono e a comunicação pode omitir mensagens.

Figura da solução de tolerância a faltas:

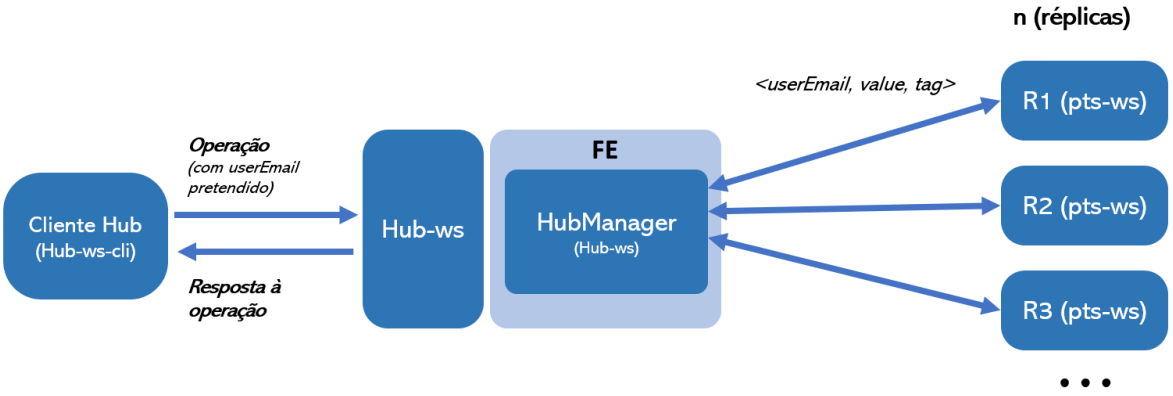

Figura 1 – Esquema da solução de tolerância a faltas

Descrição da figura e breve explicação da solução:

De modo a tornar o servidor de pontos tolerante a faltas, será utilizada uma abordagem de replicação ativa, implementando uma adaptação do protocolo Quorum Consensus para coordenar as leituras e as escritas dos saldos. A utilização da replicação ativa assegura que todos os servidores recebem pela mesma ordem os pedidos dos clientes. Para isso, é necessário:

1. Replicar os saldos de conta dos utilizadores do ForkExec pelos N servidores de réplica pts. Toda a manutenção das operações de leituras e escritas nas réplicas é assegurada pelo HubManager(FrontEnd).

2. Estes N servidores passam a guardar o saldo de todas as contas que se encontram associados a uma tag. Fazendo a associação com o protocolo Quorum Consensus, cada réplica irá guardar o saldo das contas (o valor do objeto) e uma tag para cada conta.

3. No protocolo Quorum Consensus, a tag é composta pelo número de sequência de escrita que deu origem à versão e pelo identificador do cliente que escreveu. No caso do projeto será utilizado um inteiro que indicará a versão do saldo, mas não será necessário um identificador de cliente. Isto é possível, uma vez que utilizando sincronização de threads é possível assegurar que nunca haverá chamadas concorrentes sobre a mesma conta a chegarem ao sistema replicado.

Deste modo, apesar do Hub suportar múltiplas threads comportar-se-á como se fosse apenas um cliente, permitindo a não utilização de um identificador de cliente. Para este fim, as operações de escrita e leitura do saldo das contas serão synchronized.

4. As leituras feitas aos saldos serão efetuadas pelo Hub ao chamar a função read do HubManager, onde é feita uma leitura a todos os servidores de réplica, que respondem com <saldo,tag>, encapsulado com uma Balanceview. O HubManager espera por Q respostas (N/2 + 1), escolhendo sempre a que tem o valor de tag mais elevado. Em seguida devolve uma BalanceView para o Hub.

5. As escritas são iniciadas por uma leitura para obter o valor da maior tag. Em seguida é incrementado o valor da tag em um, e é enviada uma chamada à função write que é feita para todos os servidores de réplica. Em seguida o HubManager espera por Q acks de modo a confirmar a sua escrita.

Descrição de otimizações /simplificações:

- A não existência de um identificador do cliente torna o processo mais eficiente, uma vez que são efetuadas menos verificações.
- Tanto as operações de escrita como leitura serão implementadas como operações assíncronas por polling dado que este sistema não possui um número de acessos significativo que justifique a utilização do método callback.

Detalhe do protocolo (troca de mensagens):

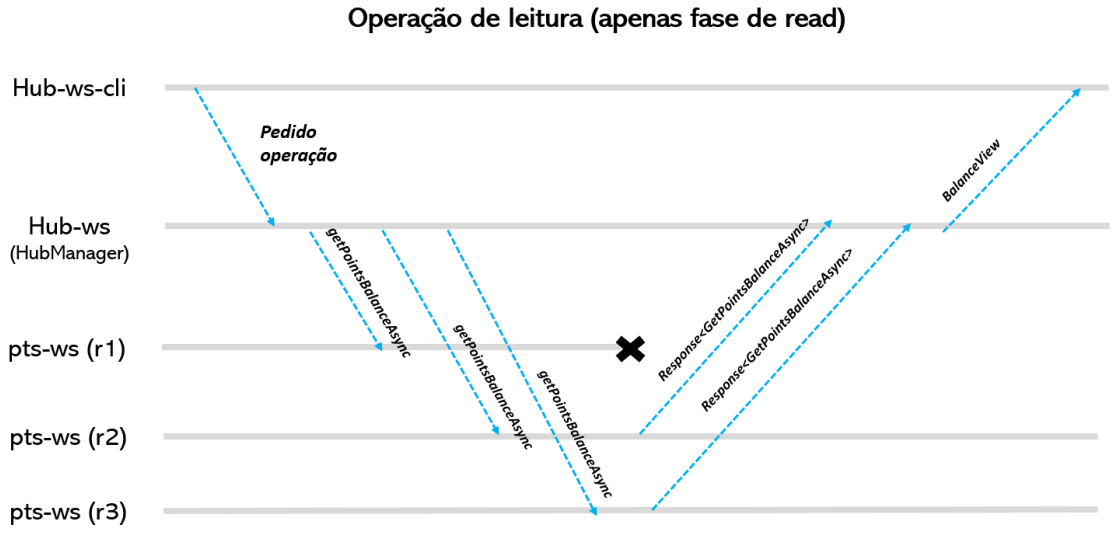

Figura 2– Esquema resumido da troca de mensagens numa operação de read

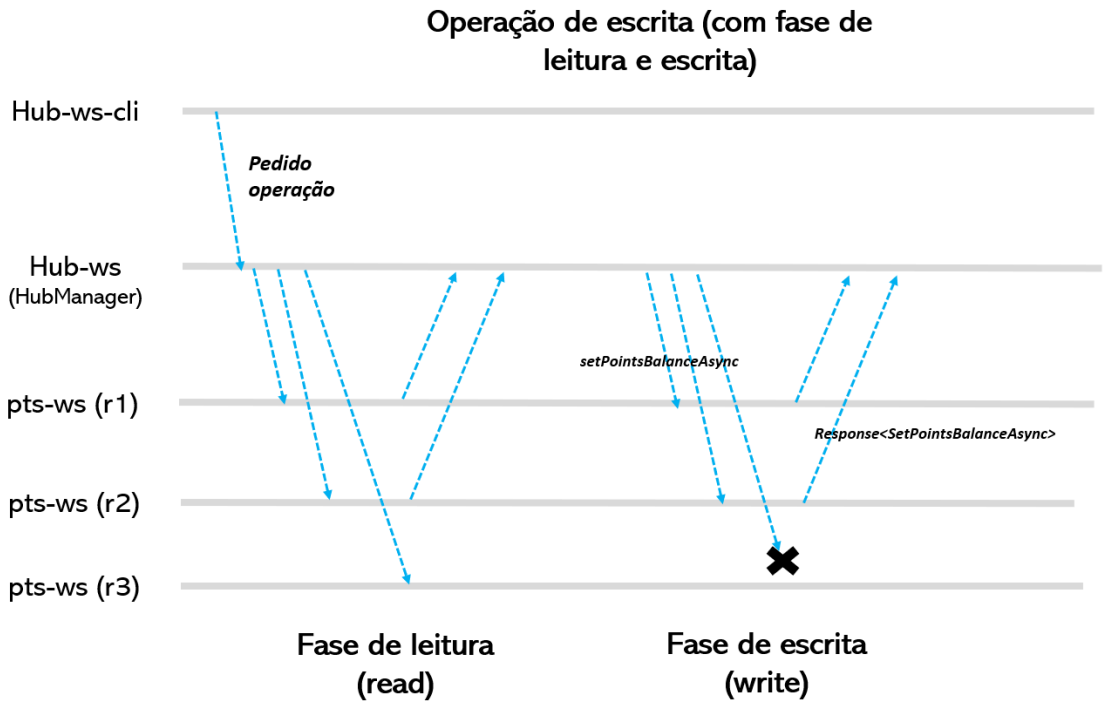

Figura 3– Esquema resumido da troca de mensagens numa operação de write