

ForkExec – T02

Entrega 2 – Tolerância a Faltas

Nome	Número
Catarina Pedreira	87524
Miguel Coelho	87687
Ricardo Silva	87700

Git Repository: <https://github.com/tecnico-distsys/T02-ForkExec>

Introdução

A segunda entrega deste projecto tem como principal objectivo garantir que as alterações aos saldos de cada utilizador não se percam nem sejam corrompidas, mesmo que o servidor de Pontos possa falhar.

Este modelo terá por base a criação de N gestores de réplica do servidor de Pontos e o algoritmo de replicação activa, *Quorum Consensus*.

Modelo de Faltas

São tomados os seguintes pressupostos para garantir a consistência e tolerância a faltas neste modelo:

- Apenas os gestores de réplica do servidor de Pontos podem falhar, ou seja, os servidores dos módulos Hub, UDDI, Restaurante e CC não falham.
- Os gestores de réplica do servidor de Pontos podem falhar silenciosamente mas não arbitrariamente, ou seja, não ocorrem falhas bizantinas.
- O sistema é assíncrono e o canal de comunicação tem pouca fiabilidade, ou seja, as mensagens podem ser omitidas, duplicadas e recebidas fora de ordem.
- No máximo, existe uma minoria de gestores de réplica em falha em simultâneo.
- O sistema não executa reparação de gestores de réplicas quando estas falham.

Protocolo *Quorum Consensus*

O protocolo funciona com um sistema de Quóruns, ou seja, um conjunto de subconjuntos de gestores de réplicas onde quais quer subconjuntos se intersectam, ou seja, $Q > N / 2$, garantindo assim consistência sequencial para cada utilizador nas leituras e escritas.

Cada gestor de réplica do servidor de Pontos guarda não só o email e o saldo associado a cada utilizador, mas também um número inteiro, tag, para cada conta.

Ao receber uma leitura para uma certa conta, o gestor de réplica responde com o tuplo <Tag, Saldo>.

Ao receber uma escrita escreve o novo par <Tag, Saldo> nessa conta e responde com uma mensagem de acknowledge.

Cada cliente comunica com um Front End que depois faz a gestão de escritas e leituras entre os gestores de réplica de forma a garantir um estado sequencialmente consistente para cada cliente.

Ao receber uma leitura, o Front End envia assíncronamente pedidos de leitura para todos os gestores de réplicas e aguarda por **Q** respostas, depois retorna o valor do par cuja Tag é maior e consequentemente mais recente.

Ao receber uma escrita, o Front End envia assíncronamente, numa primeira fase, pedidos de leitura para todos os gestores de réplicas para determinar a **Tag** mais recente, em seguida, envia assíncronamente pedidos de escrita com **<Tag+1, Novo Valor>**, aguarda por **Q** respostas e retorna ao cliente.

Este protocolo tem um grau de replicação de **2F+1**, ou seja, para tolerar **F** faltas são necessários **2F+1** gestores de réplica.

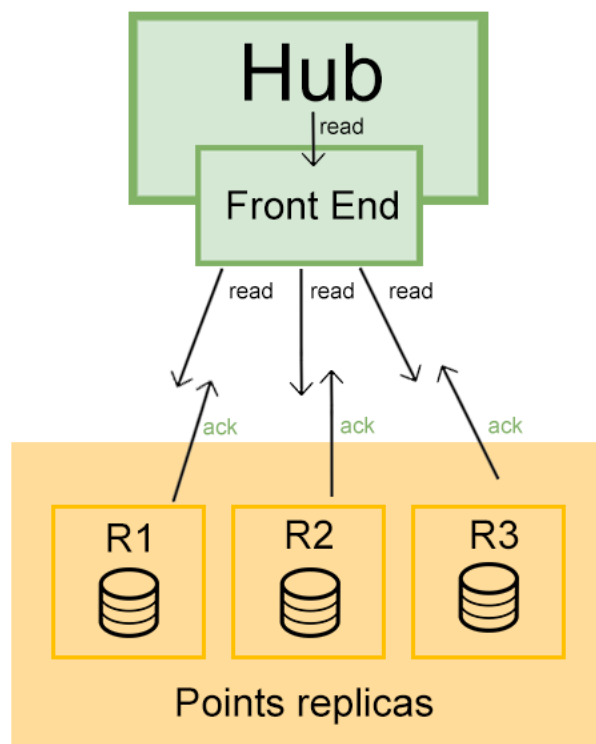


Figura 1 – 3 réplicas

Na Figura 1 conseguimos visualizar o Hub a fazer um pedido de leitura de uma conta ao Front End que por sua vez encaminha o pedido para todas as réplicas e retornará a execução do Hub quando receber um Quórum de no mínimo duas respostas.

Optimizações

É importante notar que, neste projecto, a instância única do Hub é o único cliente que acede aos gestores de réplica dos servidores de Pontos, logo não é necessário a identificação do cliente usada na Tag do protocolo base. Assim a tag passa a ser representada apenas por um inteiro.

Visto que existe apenas um Front End, este é o único que consegue escrever e ler dos gestores de réplicas, ou seja, é o único que consegue alterar as Tags de cada conta e, portanto, decidimos guardar na memória do Front End um mapa que associa cada conta à tag mais recente dessa conta e actualizar esse mapa em cada operação de escrita.

Deste modo, podemos eliminar a primeira fase de leitura da operação de escrita, e ao invés de pedir aos gestores de réplicas pela Tag mais recente apenas consultamos o mapa que existe em memória no Front End.