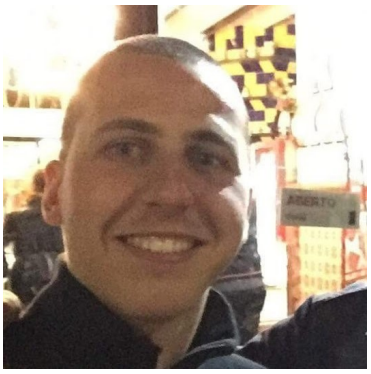
Sistemas Distribuidos

Relatório da 2ª Parte: Tolerância a Faltas

70119
André Rodrigues

86420
Francisco Sena



86447
João Palet

Repositório
`https://github.com/tecnico-distsys/A51-ForkExec`

1 Definição do modelo de faltas

O algoritmo implementado para tolerar faltas no sistema foi o Quorum Consensus. Para definir o modelo de faltas são assumidos alguns pressupostos:

- Os gestores de réplica podem falhar silenciosamente, mas não arbitrariamente, ou seja, não há 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.

2 Descrição da solução de tolerância a faltas

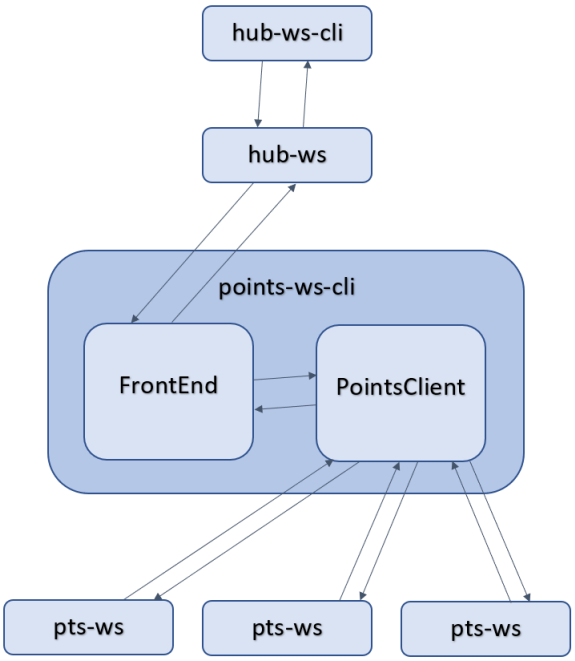


Figura 1: Arquitectura da solução

A figura acima descreve a arquitetura da solução. O Hub-Ws para tratar os pedidos do cliente Hub-Ws-Cli instancia um FrontEnd que gere a lógica do algoritmo Quorum Consensus. Este, por sua vez, instancia vários PointsClient que ficam responsáveis pela comunicação com os servidores de pontos. De facto, é instanciado um PointsClient para cada réplica ativa do servidor de pontos.

Na troca de mensagens, o FrontEnd invoca de forma assíncrona as chamadas remotas dos PointsClient que instanciou, assim, cada PointsClient fica responsável por tratar da comunicação com a sua réplica. É essencial que essas chamadas sejam assíncronas, caso contrário o sistema deixaria de ser tolerante a falhas. Finalmente, quando uma resposta chegar, FrontEnd verifica se já tem uma maioria de respostas de forma a poder tomar uma decisão.

A comunicação entre o cliente e as réplicas é feita exclusivamente a partir dos métodos write() e read() presentes no contrato WSDL. O método *write* consiste em enviar uma mensagem com o userEmail, o valor de pontos e a tag a todas as réplicas ativas.O método *read* retorna um PointsView, definido no WSDL e construído nas réplicas, contendo os atributos valor e tag.

3 Descrição de otimizações/simplificações

Foi implementada uma cache no FrontEnd que associa um email de um utilizador a um objeto que encapsula os mais recentes valores de pontos e tag associados. A cache é atualizada sempre que é feita uma escrita nas réplicas e sempre que é consultado o saldo de pontos (e respetiva tag) de um utilizador. Esta otimização permite poupar chamadas às réplicas para leitura sempre que os valores pretendidos estão disponíveis em cache.

Geralmente, o atributo tag usado no Quorum Consensus tem dois campos: "clienteID" e "sequenceNumber". Visto que só existe um cliente de cada vez a fazer pedidos, o campo "clienteID" é redundante, portanto foi desprezado.

Uma otimização possível que não foi implementada seria atribuir thresholds de escrita e leitura. Visto que as operações de leitura são mais frequentes (por cada write é feito, pelo menos, um read) atribuir-se-iam tresholds tais que:

$$threshold(read) < threshold(write) \tag{1}$$

Alguns métodos podem ser assegurados com garantias mais fracas, neste caso, o "activateAccount" e as operações de controlo.

- O método "activateAccount" limita-se a invocar o método readAsync() que irá chegar sempre a uma maioria das réplicas disponíveis. Assim, se uma réplica falhar temporariamente, quando recuperar está imediatamente pronta para participar, pois ficará atualizada assim que receber um novo pedido. Isto é possível dado que as réplicas do servidor de pontos, ao receber um pedido de leitura ou escrita para um determinado utilizador, este é criado no momento se não existir.

- As operações de controlo desprezam a possibilidade de ocorrerem faltas pois servem apenas para auxiliar a execução de testes.

No âmbito deste projeto não foram atribuídos pesos às réplicas, visto que isso só seria interessante se diferentes réplicas estivessem a correr em diferentes máquinas, em que se iria dar maior peso a réplicas mais fiáveis, com melhor conectividade ou com maior poder computacional.

4 Detalhe do protocolo – Trocas de mensagens

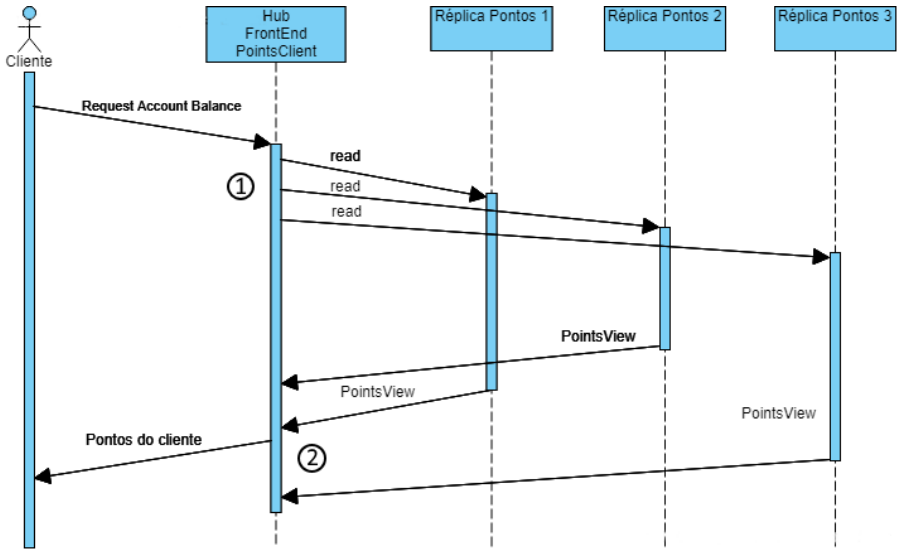


Figura 2: Troca de mensagens

O diagrama acima descreve a troca de mensagens entre as diversas entidades do sistema. É importante notar que:

- Em 1, as chamadas ao método read() são assíncronas.
- Em 2, observa-se o algoritmo de tolerância a faltas a funcionar. O FrontEnd após receber uma maioria de respostas das réplicas (neste exemplo, duas respostas) toma a decisão de retornar ao cliente, não se importando com a resposta da terceira réplica que demorou mais tempo que as outras a processar o pedido. Outro cenário possível que pode ocorrer é uma réplica falhar silenciosamente durante o processamento do pedido, nunca retornando ao FrontEnd uma resposta. O canal de comunicação entre uma réplica e o FrontEnd pode falhar fazendo com que a mensagem de retorno da réplica nunca chegue ao FrontEnd. O essencial para manter o funcionamento do sistema normal para o cliente é que uma maioria das réplicas não falhe, que é um dos pressupostos assumidos.