ForkExec - SD

Grupo A61
Daniela Lopes, n86403
Joana Teodoro, n86440
Taíssa Ribeiro, n86514

`https://github.com/tecnico-distsys/A61-ForkExec.git`

1 Definição do Modelo de Faltas

Pretende-se seguir uma abordagem de replicação ativa, implementando o protocolo *Quorum Consensus* para coordenar leituras e escritas nas réplicas.

1.1 *Quorum Consensus*

No algoritmo, o cliente (Hub) tem um front-end responsável por fazer operações de read e write, seleccionando o valor mais recentemente alterado no caso do read, determinado com a ajuda de uma tag (correspondente à versão, com valor inicial 0). Quando o front-end faz um write, ele faz read às réplicas de servidores e determina qual o valor da tag mais alto (assegurado pelo algoritmo, pois são sempre esperadas nServers // 2 + 1 respostas a esse read), enviando informação a todos os servidores réplica para atualizarem o valor com o obtido no read + número de pontos requerido pelo Hub (quer seja uma operação para gastar ou adicionar pontos) e atualizarem o valor da tag para o valor da tag anterior + 1. Nesta última, mais uma vez são esperadas nServers // 2 + 1 respostas a esse write. Assim, assegura-se que sempre que é feito write, pelo menos mais de metade das réplicas ficam com o valor mais recente e, quando se faz um read, é sempre obtido o valor mais recente a partir da tag com valor mais alto, umavez que ao receber mais de metade das respostas sabe-se que pelo menos uma dessas respostas vai ser a mais atualizada. Para assegurar a compatibilidade da nossa solução com este protocolo, no front-end considerou-se que as operações spendPoints e addPoints correspondem a um write e as operações activateAccount e accountBalance correspondem a um read, visto que caso um cliente ainda não exista, no read, ele é ativado com balance = 100 e tag = 0.

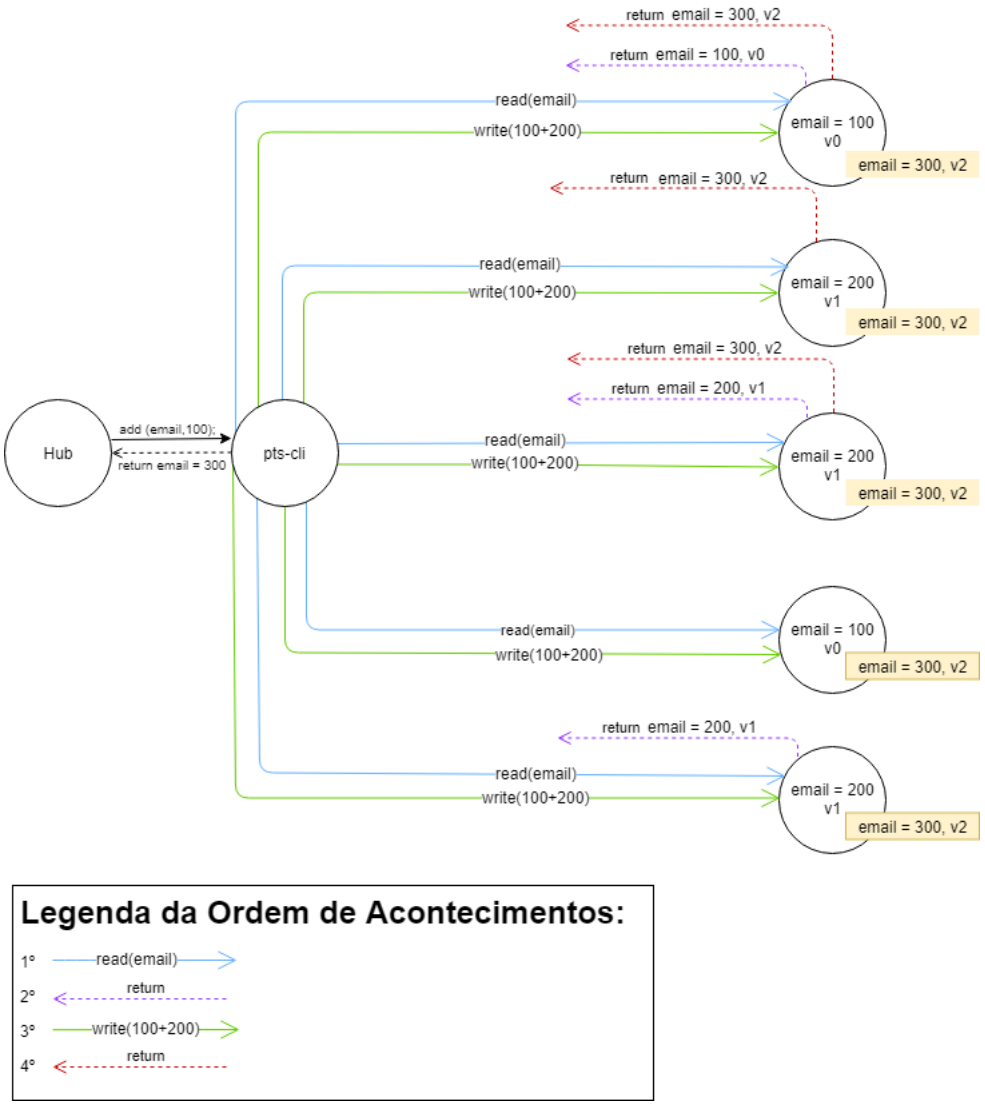
2 Descrição da Figura e Breve Explicação da Solução

Foi criado a classe PointsFrontEnd que guarda uma instância de PointsClient por cada servidor de pontos ativo no UDDI. O Hub, por sua vez, guarda uma instância do PointsFrontEnd daqual faz uso sempre que quer fazer uma chamada a um método remoto do Points. A implementação prática da solução baseia-se na geração de código a partir da teórica explicada no ponto 1.1 - *Quorum Consensus*.

2.1 Chamadas a Métodos Remotos

Optou-se por usar invocação assíncrona por polling, pois é algo facilmente implementado com o *Quorum Consensus* e visto que tem de se fazer espera pela resposta antes de prosseguir com a execução, não faria sentido implementar callback (implementação deste não é trivial) e esta não impediria a espera inevitável por resposta.

2.2 Figura da Solução de Tolerância a Faltas



2.3 Detalhe do Protocolo (Troca de Mensagens)

2.3.1 Troca de Mensagens Geral

Relativamente à troca de mensagens, o Hub, quando pretende efetuar uma operação a partir da instância de PointsClient, vai através desta última fazer pedidos de leitura e de escrita conforme a operação que deseja. Após realizar cada operação para concluir o pedido inicial terá de aplicar o *Quorum Consen-*

sus, ou seja, irá esperar pela resposta da maioria dos servidores que se encontram ativos.

2.3.2 Troca de Mensagens na Figura

Na figura apresentada acima, pretende-se adicionar pontos. Tal como explicado na subsecção anterior, a instância de PointsClient irá pedir a leitura dos pontos atuais do utilizador em causa. Posteriormente PointsClient vai esperar pela resposta de nServers / 2 + 1, aplicando o *Quorum Consensus*. Assim que obtém a primeira resposta, PointsClient fará novamente um pedido de escrita para adicionar os pontos que pretende ao saldo do cliente em causa, voltando, da mesma forma, a obter resposta por parte da maioria dos servidores, com o novo saldo e a nova tag, atualizados. Por fim, a instância PointsClient irá retornar ao Hub o resultado final da ação pedida.

3 Descrição de Otimizações/Simplificações

3.1 Multiplicidade de Clientes

Assume-se que existe apenas uma instância do Hub, ou seja, um cliente, pelo que foram feitas alterações no código no sentido em que apenas é criado um cliente no servidor de Points (front-end) que processa todas as chamadas remotas. Não se aplicou sincronização de objetos no front-end, pelo facto de ser, como já mencionado, apenas uma instância.

3.2 Leitura de Respostas dos Servidores Points Repetidas

Quando o front-end faz read ou write e espera pelas respostas. Adicionou-se uma flag a cada PointsClient para determinar se essa resposta já foi processada, e não voltar a ler respostas já recebidas.

3.3 Manter Registo dos Clientes em Todas as Réplicas

Uma vez que existe o problema de nem todos os servidores ativarem todos os utilizadores que já foram inicializados, considerou-se que, sempre que uma operação é feita sobre uma conta, se essa conta ainda não existir, é criada em vez de ser lançada a exceção InvalidEmail.

3.4 Registos em Cache

Foi implementada uma cache com tamanho máximo de 5 elementos e substituição por valor pseudoaleatório acedida com o userEmail, com complexidade O(1). Esta mantém os valores mais atualizados dos utilizadores presentes na cache, o que evita que seja aplicado o Quorum Consensus em operações repetidas com o mesmo utilizador.