TÉCNICO LISBOA

RELATÓRIO 2ª ENTREGA

Sistemas Distribuídos 2017/2018

A37 – Prof. Guilherme Ilunga, 6ª Feira 11h30

Francisco Pereira

76196

Diogo Freitas

81586

João Rodrigues

83483

https://github.com/tecnico-distsys/A37-SD18Proj

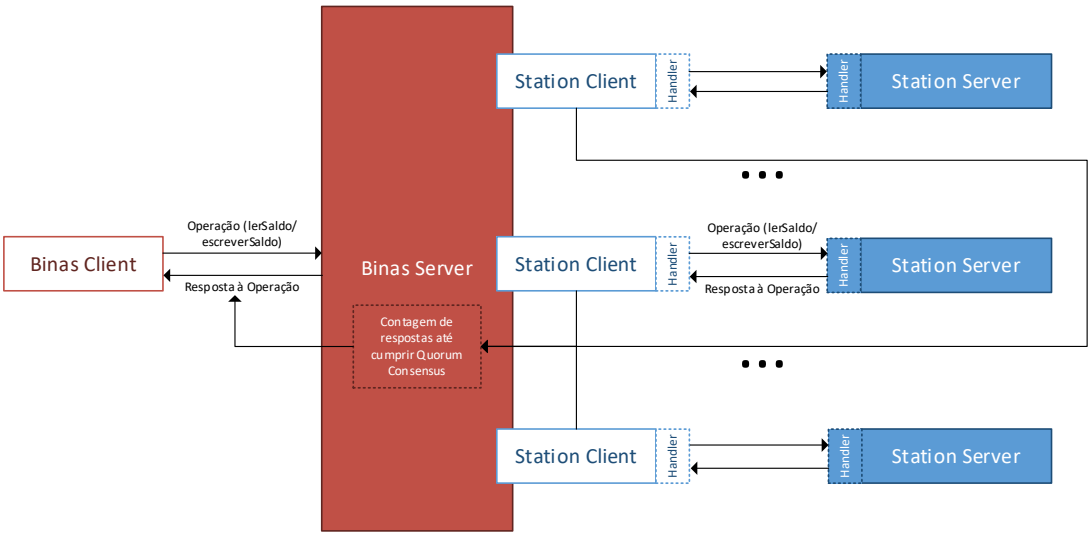

Figura 1. Esquema da solução de tolerância de faltas

Replicação

A replicação activa do saldo dos utilizadores é conseguida guardando esses dados dentro das estações, que actuam como gestores de réplicas. Cabe ao servidor Binas certificar-se de que os dados destas réplicas se mantêm consistentes – ou melhor, certificar consistência sequencial. Para tal foi utilizado o protocolo *Quorum Consensus.*

Quorum Consensus

Cada resposta a um pedido de saldo, de um dado utilizador, às estações (onde estão guardadas as réplicas) é recebida por uma callback. No entanto, ao ser mantido um número sequencial identificador do pedido do cliente, torna-se possível associar cada resposta das estações a um pedido em especifíco feito anterirormente, descartando-se as respostas sem utilidade (as respostas são consideradas sem utilidade assim que o número de respostas atinge uma maioria como definida pelo *Quorum Consensus* – no caso especifíco deste projecto N/2 + 1 respostas, sendo N o número de estações).

Visto que nos foi pedido para considerar apenas um cliente de Binas, a existência de um identificador de cliente na *tag* das réplicas torna-se desnecessária. Assim a *tag* possui apenas o número identificador sequencial também chamado de *timestamp*.

Cada vez que é feito um pedido de alteração de dados às réplicas, é igualmente feito um pedido de obtenção dos dados nas réplicas e é através desses dados que o valor da *tag* é actualizado. É ainda necessário que uma maioria destas réplicas tenha os seus dados alterados, assim garantindo a consistência sequencial.

Tolerância a Faltas

Por forma a podermos tolerar faltas fizémos com que o servidor Binas não guardasse informação acerca do saldo dos utilizadores. Desta forma se o servidor Binas falhar, a informação pertinente ao saldo é mantida pois está replicada nas estações existentes como já mencionado acima.

Sincronização de múltiplas threads

De forma a que os dados de uma réplica de utilizador, onde é mantido o saldo, não sejam alterados por múltiplas threads utilizamos trincos que bloqueiam o acesso aos dados da réplica não permitindo a sua alteração. Estes trincos bloqueiam atributos específicos de uma réplica na execução de métodos como *getUserReplic* e *addUserReplic* que como os nomes indicam são métodos que acedem directamente aos atributos da réplica.

Alterações nos métodos do *BinasManager*

Para que as funcionalidades pedidas nesta entrega fossem implementadas criámos dois métodos na classe *BinasManager*, *getBalance* e *setBalance*, estes métodos, tal como é indicado pelos seus nomes, comunicam com as estações para obter os saldos das réplicas dos utilizadores e, como tal, são estes métodos que implementam o *Quorum Consensus* ao esperarem pela maioria das respostas provenientes das estações para depois dar a resposta ao pedido do cliente.

Para além destas duas adições foram feitas alterações noutros métodos da classe *BinasManager* sendo estes: *rentBina*, *returnBina* e *activateUser*.