Grupo A65

Url para repositório :https://github.com/tecnico-distsys/A65-ForkExec

71015

Ricardo Nunes

Introdução

A aplicação *Forkexec* é uma aplicação distribuída e como tal está dependente da interação entre diferentes servidores. Sabendo isto é boa prática evitar falhas na comunicação entre os vários servidores que compõem a aplicação.

Uma das principais preocupações é a interação entre o servidor *Hub* e o servidor de pontos sendo que uma falha nesta componente pode causar indisponibilidade da principal funcionalidade do sistema.

Visto isto é importante implementar um modelo de tolerância a falhas ,por exemplo, através de replicação ativa.

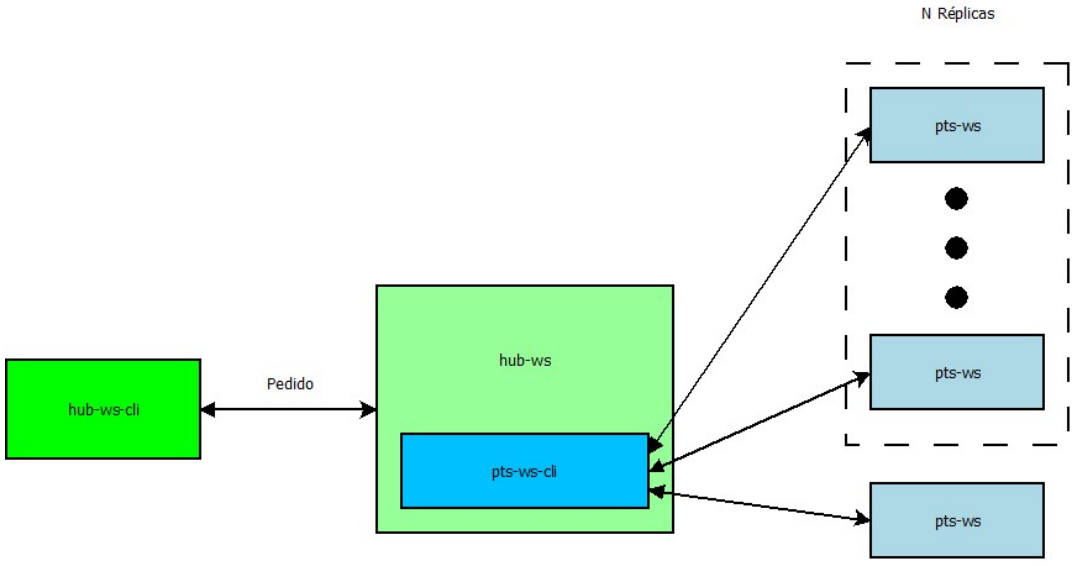
Modelo de faltas

Para o desenvolvimento da segunda parte do projeto vamos considerar que as faltas passiveis de acontecer são as seguintes:

- Falha por interrupção involuntária do servidor de pontos, impedido assim que seja possível debitar ou obter os pontos de um determinado utilizador.
- Falha por perda da mensagem durante a comunicação ou omissão da mesma da parte do servidor de pontos.

Solução com otimizações

Para suportar a replicação ativa do projeto, é utilizado uma variante do protocolo *Quorum Consensus* este protocolo incorpora um conjunto de subconjuntos de réplicas, tal que quaisquer dois subconjuntos se intersetam.



Cada réplica guarda o valor do objeto e a respetiva *tag*, no caso do *Forkexec*, o valor trata-se dos pontos de um utilizadores e a *tag* será um inteiro que será utilizado como “versão” dos pontos do utilizador.

O servidor *Hub* é *multi-threaded*, logo é possível que duas ou mais *threads* do *Hub* decidam ler ou escrever concorrentemente no sistema replicado. Para esse cenário, ambas as *threads*

comportam-se como clientes concorrentes. Assim como maneira de simplificar da solução as funções de gestão das operações de *write* e *read* do Q*uorom Consensus* são implementadas como *synchronized* para impedir que duas threads consigam aceder aos dados simultaneamente. Isto permite que a tag não inclua o *client-id*, ao contrário do previsto pelo protocolo Quorum Consensus original. Com esta optimização apesar do *Hub* ser *multi-threaded*, com a sincronização correta gerida pelo java o Hub comportar-se-á sempre como um único cliente. A não existência deste *client-id* torna o processo mais eficiente, uma vez que são efetuadas menos verificações durante os processos de *read/write*.

Para além disso outra maneira de melhorar a performance do sistema seria evitar que o cliente fique bloqueado à espera da resposta do servidor, esta melhoria é possível se as invocações das operações *read/write* forem feitas através de chamadas assíncronas , sendo que seria possível utilizar os dois métodos estudados nos laboratórios polling ou call-back. Apesar das vantagens a nível de performance que o *call-back* tem em comparação com o polling, para o sistema *ForkExec* e para a demonstração do projeto o metodo *polling* será mais simples de implementar e será suficiente para melhorar a performance comparado com as chamadas síncronas atuais.

Protocolo de interação

Nesta secção podemos considerar as letras N e Q são, respetivamente, o número total de réplicas e o quórum do sistema, ou seja, a maioria do universo de servidores de replicação |Q|>N/2.

As leituras previstas pelo protocolo são efetuadas pelo servidor *Hub* , que lança uma chamada à função *read* a todos N os gestores da réplica, que respondem com o valor e a *tag* que contém. O servidor *Hub* aguarda por Q respostas, escolhendo sempre o valor que tem a maior *tag* (mesmo que exista uma maioria de réplicas com outra *tag*).

Na fase de escrita o servidor executa primeiro uma leitura para obter o valor que tem a maior tag, estabelece uma nova tag incrementado o valor da antiga tag e envia uma chamada à função write a todos os gestores de réplica. O servidor espera então por Q *acknowledges* (ACK), que confirma que os servidores receberam a informação e a registaram .

A imagem seguinte demonstra a interação entre os diferentes servidores da aplicação com replicação ativa através de *quorom consesus*.

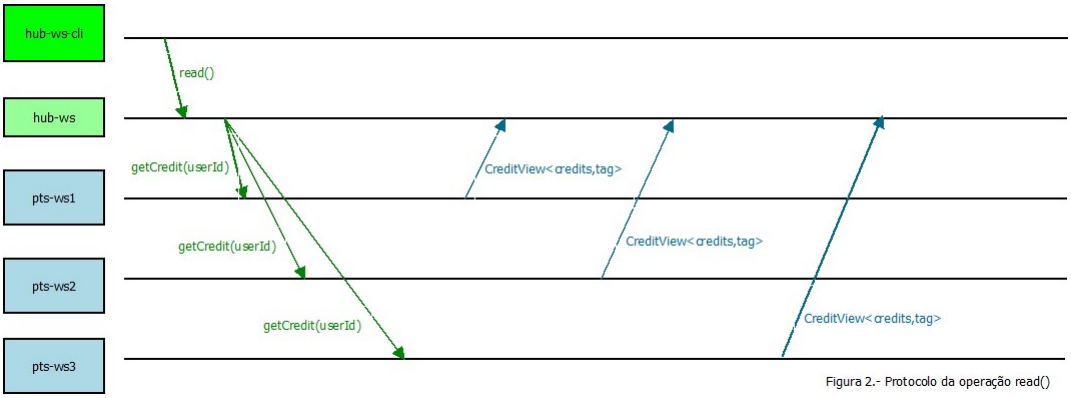


Figura 2.- Protocolo da operação read()

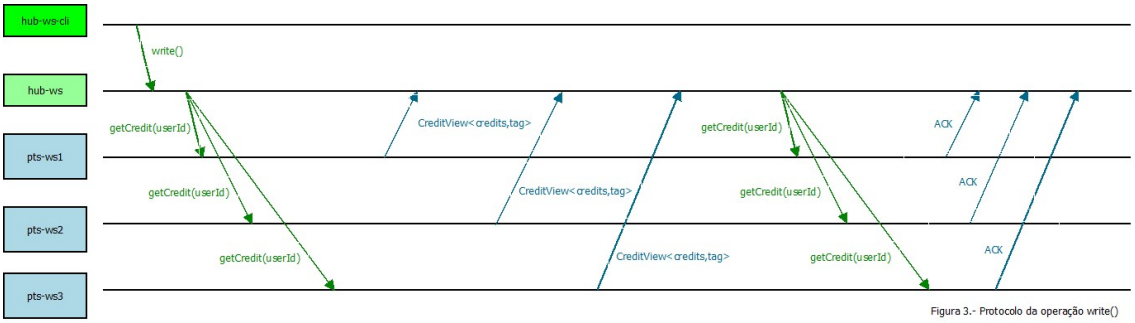


Figura 3.- Protocolo da operação write()