Grupo A14

https://github.com/tecnico-distsys/A14-ForkExec

Número	86416	86445	86450
Nome	Francisco Sousa	João Daniel	João Barata

Pressupostos

Desenhámos a nossa solução para a replicação do servidor de pontos tendo por base:

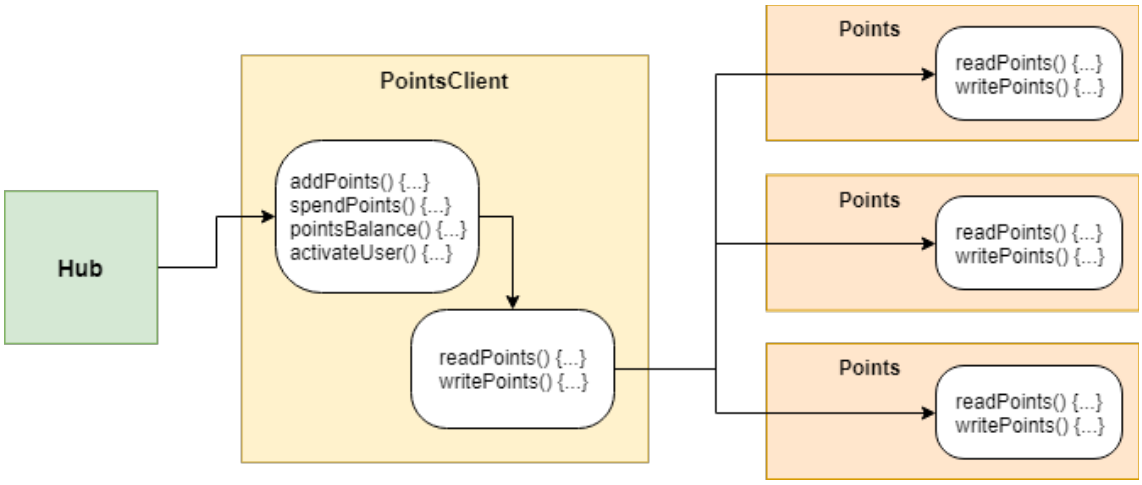
- Existe um conjunto estático de N gestores de réplica, que não pode ser alterado durante a execução do programa
- Existe apenas uma instância de cliente (Hub)
- Não há faltas bizantinas
- No máximo falha uma minoria de N
- O sistema é assíncrono

Modelo de faltas

Sabendo que não há faltas bizantinas, apenas vamos tolerar faltas silenciosas de menos de metade dos N gestores de réplica iniciais. O nosso modelo de faltas tolera ainda que haja apenas uma instância do cliente a correr ao mesmo tempo, podendo ser desconectado e conectado a qualquer momento.

Quaisquer outras faltas não vão ser tolerados.

Solução de tolerância a faltas:



O cliente (Hub) faz pedidos síncronos ao PointsClient, através da interface disponível publicamente e implementada em entregas anteriores. O PointsClient converte a lógica das funções públicas de modo a usar o readPoints() e writePoints() desta classe, também síncronos. Por sua vez, estes dois métodos vão fazer chamadas assíncronas ao readPoints() e writePoints() das várias replicas.

Assim, o PointsClient serve de Front End, abstraindo os vários gestores de réplica.

Implementação do quorum consensos com otimizações:

(sendo |Q| > N/2)

Para leitura

Front End:

```
int readPoints(userId) {

	enviar readPoints(userId) para todos os gestores de réplica, no máximo 1 vez;

	aguarda por Q respostas;

	return o valor com a tag maior;

}
```

Gestor de Réplica:

```
<int, int> readPoints(userId) {

	return <points, tag>;

}
```

Para escrita

Front End:

```
void writePoints(userId, points) {

	calcular maxtag = maxtag+1;

	envia writePoints(userId, points, maxtag) a todas as replicas;

	espera por Q acks;

}
```

Gestor de réplica:

```
int writePoints(userId, points, tag) {

	se tag > currentTag, substitui o valor guardado;

	return ack;

}
```

Sincronização de pedidos

Para evitar bloqueamento do Front-End ao fazer pedidos aos vários gestores de réplica, usamos chamadas assíncronas. Decidimos implementar isto através de polling porque é mais interessante para a solução.

Otimizações

No protocolo original, as tags são compostas pelo número de sequência e pelo id do cliente que escreveu o último valor. Como só há no máximo um cliente ativo ao mesmo tempo, decidimos omitir o id do cliente da tag, sendo esta composta apenas por um número de sequência.

Também no protocolo original, antes de cada escrita, é feito um read a todas as réplicas para descobrir a tag maior. Pela mesma razão mencionada acima, decidimos que o cliente pode ir guardando este valor numa variável local maxTag. Apenas necessita de o calcular quando é criado, já que podem ter existido outras instâncias de cliente antes da atual.

Ainda no protocolo original, existe a possibilidade de implementar write-back, cuidando de escritas concorrentes. Pela mesma razão, decidimos não o fazer.

Há variantes de pesos variáveis ou quóruns de leitura e escrita que não se justificam para a nosso caso.

Em relação ao polling, ao verificarmos se cada resposta já foi concluída, adicionamos as ainda por terminar a uma nova lista. Assim, diminui-se o número de respostas a serem verificadas sucessivamente.

Outras funções

Além do readPoints e do writePoints no Front-End, também decidimos criar a função getMaxTag(), para calcular a tag maior de todas as réplicas, na criação do cliente.