Universidade Federal de Santa Catarina, câmpus Florianópolis
Curso: Ciências da Computação
Unidade Curricular: Computação distribuída
Atividade: Relatório do Trabalho 2
Acadêmicos: João Pedro Perez Resmer (22100627), Rafael Francisco Réus (19100543) e Tiago Oliveira da Luz (21203771)

1. Descrição da solução

Utilizamos-nos de um caso de uso para o espaço de tuplas compartilhado no qual nos baseamos em um serviço de biblioteca que gerencia dados de clientes, livros e empréstimos. Para isso, fizemos uso do serviço *Apache ZooKeeper* o qual garante, segundo sua documentação e nossos testes, a replicação, ordem, atomicidade, uma visão única do sistema e confiabilidade. Dessa forma, para a implementação, fizemos uso de algumas funcionalidades importantes e que eram requisitos da proposta:

- Classe *Node*: É o nodo local da aplicação, sua interface para o acesso ao espaço de tuplas. Cada instância da classe possui um cliente zookeeper que monitora o znode correspondente ao id da instância. A partir desse znode é feita a comunicação entre os Nodes do espaço de tuplas;
    - Método *replicate* (replicação de tuplas): envia solicitação de escrita aos nós, guarda a tupla localmente e atualiza a réplica;
    - Método *look_for_tuple* (busca de tuplas): Busca a tupla correspondente ou o molde e responde a quem solicita;
    - Método *react_to_change* (reagir a mudanças): *Callback* que trata as mudanças (escrita, busca com remoção).
- Classe *LibraryManager*: intermedia a comunicação cliente, empréstimo e livro, por meio da instanciação de *Node* com o espaço de tuplas.

Além disso, realizamos o tratamento de questões de confiabilidade, como: atomicidade, replicação e comunicação confiável, diretamente ou indiretamente.

Atomicidade: garantida a partir do *ZooKeeper* que gerencia operações críticas e sincroniza os dados nos nodos.

Replicação das tuplas: garantida por meio das operações de escritas e leituras com remoção nos nodos e que assim asseguram que haja consistência mesmo com a falha de *f* nodos.
Comunicação confiável: garantida pelo *Apache Zookeeper* de maneira que realiza um serviço semelhante ao *Reliable Broadcast* (com algumas diferenças). Além da garantia de entrega de mensagem, o serviço implementa a ordenação de eventos (*Total Order Broadcast*) e a reconfiguração do sistema em caso de falhas. Por fim, usamos *callbacks* para monitorar os *znodes* caso ocorram ações ou falhas.

Finalmente, devido à distribuição do sistema e à modelagem do sistema, optamos pela utilização dos padrão de projeto: *Observer* (aguardar reações dos *znodes* e reagir quando houver uma mudança).

2. Dificuldades e soluções

Inicialmente, tivemos dificuldade em definir a biblioteca que usaríamos, visto que buscávamos por uma intersecção entre uma linguagem de programação que tínhamos familiaridade e uma biblioteca que facilitaria ao máximo nossa codificação. Entretanto, optamos pela utilização do *ZooKeeper* (tivemos que além de optar pela biblioteca, tivemos de aprender o comportamento da biblioteca por meio da documentação e de testes com os métodos e atributos como *znodes* e *watchers*), que apesar de não apresentar algumas funcionalidades (tais quais: flexibilidade de comunicação, tolerância a falhas bizantinas, quantidade menor de primitivas de coordenação, etc), atende ao primeiro requisito.

Além disso, houve certa dificuldade na escolha quanto ao armazenamento das tuplas. Armazená-las dentro do atributo value do znode seria um grande facilitador. Porém isso iria contra a recomendação dos desenvolvedores do zookeeper e limitaria o espaço de cada nodo em 1mb. Por isso, optamos por armazenar as tuplas na classe na memória heap por meio classe Node.

3. Aplicativo

O aplicativo desenvolvido foi um sistema simples de gerenciamento de biblioteca, onde o estoque de livros é implementado usando a solução de espaço de tuplas desenvolvida. Isso permite que o sistema de duas bibliotecas diferentes, com seus próprios clientes, possam compartilhar um mesmo estoque de livros. O programa é acessado por terminal usando comandos onde são passados os argumentos. Entre as funcionalidades estão:

3.1. addbk -i id -t title -a author -y year -p publisher

Adiciona novo livro ao espaço de tupla utilizando a função write. Para o contexto da aplicação todos os campos precisam ser preenchidos.

3.2. getbk -i id -t title -a author -y year -p publisher

Verifica se o livro existe no espaço de tupla e retorna suas informações. Faz uso da função read e precisa receber apenas um dos argumentos para realizar a busca.

3.3. mkloan -c clientID -b bookid -t title -a author -y year -p publisher

Cria um empréstimo retirando livro do espaço de tupla se existir. Faz uso da função get que é bloqueante. Esse comportamento foi mantido como tal para efeito de demonstração, mas para o contexto da aplicação poderia se utilizar um get não bloqueante ou verificar a existência da tupla com a função read antes de chamar a função get. Precisa receber pelo menos uma das informações do livro para realizar a busca.

3.4. rmloan -c clientID -b bookid

Remove empréstimo do sistema local e retorna livro ao espaço de tupla

4. Caso de uso

Teste para dois processos app gerenciando duas bibliotecas, com clientes próprios e estoque de livros compartilhados.

Sequência de comandos executados:

1. Processo 1: addcli -i 1 -n Paul -a 20
2. Processo 1: addbk -i 1 -t Dune -a Frank Herbert -y 1965 -p Chilton Books
3. Processo 2: addcli -i 1 -n Geralt -a 115
4. Processo 2: addbk -i 2 -t Witcher -a Andrzej Sapkowski -y 1986 -p Fantastyka
5. Processo 1: getbk -t Witcher
6. Processo 2: getbk -a Andrzej Sapkowski

7. Processo 1: mkloan -c 1 -t Witcher
8. Processo 2: getbk -i 2
9. Processo 2: mkloan -c 1 -t Witcher
10. Processo 1: rmloan -c 1 -b 2
11. Processo 1: getbk -i 2
12. Processo 2: rmloan -c 1 -b 2

A sequência de comandos acima realiza as seguintes operações:

1. Processo 1 cadastra cliente Paul
2. Processo 1 cadastra livro Dune
3. Processo 2 cadastra cliente Geralt
4. Processo 2 cadastra livro Witcher
5. Processo 1 verifica se livro Witcher existe no estoque
6. Processo 2 verifica se livro escrito por Andrzej Sapkowski existe no estoque
7. Processo 1 realiza empréstimo do livro Witcher para cliente Paul
8. Processo 2 verifica se livro Witcher existe no estoque (Está em empréstimo)
9. Processo 2 realiza empréstimo do livro Witcher para cliente Geralt (Bloqueia até livro ser retornado)
10. Processo 1 retorna livro Witcher para estoque
11. Processo 1 verifica se livro Witcher existe no estoque (Está em empréstimo)
12. Processo 2 retorna livro Witcher para estoque

Comandos executados por processo:

Processo 1:

```
addcli -i 1 -n Paul -a 20
addbk -i 1 -t Dune -a Frank Herbert -y 1965 -p Chilton Books
getbk -t Witcher
mkloan -c 1 -t Witcher
rmloan -c 1 -b 2
getbk -i 2
```

Processo 2:

```
addcli -i 1 -n Geralt -a 115
addbk -i 2 -t Witcher -a Andrzej Sapkowski -y 1986 -p Fantastyka
getbk -i 2
getbk -i 2
mkloan -c 1 -b 2
rmloan -c 1 -b 2
```