



UNIVERSIDADE FEDERAL DE SANTA CATARINA
CAMPUS REITOR JOÃO DAVID FERREIRA LIMA
CIÊNCIAS DA COMPUTAÇÃO

Gabriel Martins Bergmann

Gustavo Russi

Tainá da Cruz

Trabalho 2: Espaço de Tuplas Tolerante a Falhas

Florianópolis

2024

Gabriel Martins Bergmann (20105234)

Gustavo Russi (20100526)

Tainá da Cruz (20100547)

Trabalho 2: Espaço de Tuplas Tolerante a Falhas

Trabalho submetido à disciplina de Computação Distribuída, no curso de Ciências da Computação da Universidade Federal de Santa Catarina, como requisito para a sua aprovação.
Professor: Odorico Machado Mendizabal

Florianópolis

2024

RESUMO

Este trabalho consiste na implementação de um serviço de espaço de tuplas que permite a adição, remoção e leitura de tuplas de um espaço de memória compartilhada. As operações devem ser atômicas e o sistema deve ser capaz de tolerar falhas sem perder dados. A replicação e a tolerância a falhas são elementos essenciais para a robustez de um serviço. Para isso, o uso de bibliotecas auxiliares, como Apache ZooKeeper, mostra ser importante para garantir a confiabilidade e a ordem de entrega das mensagens. A API do serviço deve incluir as operações `write(tupla p)`, que adiciona a tupla `p` ao espaço de tuplas; `read(tupla p)`, que faz uma cópia de uma tupla que corresponda ao padrão `p`; `get(tupla p)`, que retira uma tupla que corresponda ao padrão `p`. Além da implementação do serviço, uma aplicação simples foi desenvolvida, demonstrando o uso do serviço de espaço de tuplas, com as operações `read`, `get` e `write`.

Palavras-chave: Espaço de Tuplas; Replicação; Tolerância a Falhas.

SUMÁRIO

1 INTRODUÇÃO	7
2 DESENVOLVIMENTO	8
2.1 Serviço de Espaço de Tuplas	8
2.2 Aplicação	8
2.2.1 Backend	10
2.2.2 Frontend	10
3 EXECUÇÃO	10
3.1 Inicialização do Apache Zookeeper	10
3.2 Configuração e Inicialização do Backend	11
3.2 Configuração e Inicialização do Frontend	11
4 REPLICAÇÃO	12
CONCLUSÃO	13
REFERÊNCIAS	14

1 INTRODUÇÃO

O seguinte trabalho, proposto para a disciplina de Computação Distribuída no semestre 2024/1, tem como objetivo desenvolver um serviço de espaço de tuplas que seja tolerante a falhas. Esse serviço deverá garantir o acesso confiável às tuplas armazenadas, mesmo na ocorrência de falhas entre os servidores. O trabalho visa exercitar o uso de bibliotecas para comunicação confiável e implementar um sistema que suporte a atomicidade das operações de leitura, escrita e remoção de tuplas em um ambiente distribuído.

2 DESENVOLVIMENTO

A seguir são descritas as seções acerca do desenvolvimento do serviço e da aplicação.

2.1 Serviço de Espaço de Tuplas

O serviço de espaço de tuplas foi implementado utilizando o framework Flask para o backend e React para o frontend. O backend faz uso do Apache ZooKeeper para coordenação e armazenamento das tuplas, garantindo a tolerância a falhas e a replicação dos dados. As operações principais implementadas no serviço incluem:


- **write(Tupla p)**: Adiciona uma tupla ao espaço de tuplas.
- **read(Tupla p)**: Lê uma tupla correspondente ao padrão fornecido, sem removê-la do espaço.
- **get(Tupla p)**: Retira uma tupla correspondente ao padrão fornecido, removendo-a do espaço.

Essas operações são atômicas, garantindo que não haja interferência entre leituras, escritas e remoções. Isso é assegurado através da biblioteca Apache ZooKeeper.

2.2 Aplicação

A aplicação escolhida para demonstrar o serviço de espaço de tuplas consiste em um sistema de troca de livros. O usuário pode realizar o *login* ou *signup* no sistema, caso ainda não possua uma conta. Após entrar, ele poderá acessar três abas diferentes: registro de livros (operação write), busca de livros (operação get) e lista de livros (operação read). Ao adicionar um livro no espaço de tuplas, o saldo de créditos do usuário é incrementado em 1 ponto. Somente com o saldo positivo será possível realizar uma troca (ao buscar um livro) e remover um livro do espaço de tuplas. Ao remover um livro, o saldo é decrementado em 1 ponto.

Figura 1 – Tela de Registro de um Livro

INE5418 Computação Distribuída REGISTRAR LIVROS BUSCAR LIVROS LISTA DE LIVROS REGISTRADOS  taina (Créditos: 0) LOGOUT

Sistema de Troca de Livros

Registrar Livro

<input type="text" value="Título"/>	<input type="text" value="Autor"/>
<input type="text" value="Editora"/>	<input type="text" value="Ano"/>
<input type="text" value="Gênero"/>	

Fonte: elaborado pelos autores

Figura 2 – Tela da Lista de Livros

INE5418 Computação Distribuída REGISTRAR LIVROS BUSCAR LIVROS LISTA DE LIVROS REGISTRADOS  taina (Créditos: 0) LOGOUT

Sistema de Troca de Livros

Lista de Livros

aaaa, bbbb, ccc, ddd, eee

O Livro do Desassossegado, Fernando Pessoa, *, *, *

Fonte: elaborado pelos autores

2.2.1 Backend

O backend foi desenvolvido utilizando o Flask, e integra-se com o ZooKeeper para gerenciamento das tuplas. Funções de autenticação e gerenciamento de créditos foram implementadas para garantir que somente usuários autenticados possam interagir com o espaço de tuplas e que suas operações sejam contabilizadas.

2.2.2 Frontend

O frontend foi desenvolvido utilizando React com Javascript, além do Material-UI para a interface do usuário. A aplicação permite que usuários registrem, leiam e removam livros, além de visualizar a lista de livros disponíveis. A interface também exibe o nome do usuário logado e seu saldo de créditos, que é atualizado dinamicamente conforme as operações são realizadas.

3 EXECUÇÃO

Para executar o sistema, é necessário inicializar tanto o backend quanto o frontend. O backend deve ser iniciado em um ambiente Python, com o ZooKeeper em execução para gerenciar o armazenamento das tuplas. O frontend deve ser iniciado em um servidor de desenvolvimento React. A seguir são descritos os passos necessários para subir a aplicação:

3.1 Inicialização do Apache Zookeeper

Inicialmente, é necessário baixar e extrair a versão estável mais recente do Zookeeper em três diretórios separados. Após isso, configura-se cada instância do Zookeeper pelo arquivo `zoo.cfg` para especificar diferentes portas de cliente (2181, 2182, 2183), além da adição de um arquivo `myid` com os valores 1, 2 e 3 em cada diretório respectivo. Por fim, cada instância do Zookeeper é iniciada em terminais

separados com o comando `bin/zkServer.sh start`. A verificação de status pode ser possível através do comando `bin/zkServer.sh status`.

3.2 Configuração e Inicialização do Backend

1. Clone o repositório: `git clone https://github.com/tainadacruz/computacao-distribuida.git`
2. Navegue até o diretório do backend: `cd computacao-distribuida/backend`
3. Crie e ative um ambiente virtual: `python -m venv venv` e `source venv/bin/activate` (No Windows, use `venv\Scripts\activate`)
4. Instale as dependências necessárias: `pip install -r requirements.txt`
5. Inicie o servidor Flask: `python app.py`

3.2 Configuração e Inicialização do Frontend

1. Navegue até o diretório do frontend: `cd ../frontend`
2. Instale as dependências: `npm install`
3. Inicie a aplicação React: `npm start`

Após a inicialização, a aplicação estará acessível via navegador em `http://localhost:3000`, onde os usuários poderão se autenticar, registrar, ler e remover tuplas conforme necessário.

4 REPLICAÇÃO

O sistema desenvolvido faz uso de três servidores do Apache ZooKeeper, o mínimo recomendado na documentação da API, para fins de teste. Eles estão todos associados ao mesmo *localhost* e rodam na mesma máquina. Isso não proporciona segurança adicional real, mas demonstra o funcionamento da replicação da API em um ambiente de teste. Em um ambiente de produção, a estrutura seria similar, mas cada servidor estaria rodando em máquinas diferentes.

O ZooKeeper utiliza esses servidores como um quórum, com um deles sendo declarado líder. A própria API do ZooKeeper implementa algoritmos de eleição de líder e consenso. As decisões são tomadas com base na maioria dos votos, permitindo a eleição de novos líderes e a manutenção do consenso. Embora não tenhamos encontrado detalhes específicos sobre os protocolos exatos utilizados pelo ZooKeeper na sua documentação, o sistema consegue demonstrar a eficácia dos conceitos de replicação e tolerância a falhas em um ambiente distribuído.

CONCLUSÃO

O sistema de espaço de tuplas desenvolvido atende aos requisitos de confiabilidade e tolerância a falhas propostos no trabalho. A integração com o ZooKeeper garante que as tuplas sejam replicadas, permitindo a continuidade do serviço mesmo em caso de falhas entre os servidores. A aplicação frontend oferece uma interface intuitiva, facilitando a interação dos usuários com o serviço. O sistema desenvolvido demonstra a viabilidade e eficácia do uso de bibliotecas como ZooKeeper em ambientes distribuídos para garantir a consistência e disponibilidade dos dados.

REFERÊNCIAS

APACHE. **ZooKeeper Documentation**. Disponível em:
<https://zookeeper.apache.org/documentation.html>. Acesso em: 29 jun. 2024.