Relatório Final: Projeto de Sistema de Gerenciamento de Estoque para uma Cadeia de Supermercados

Erick Denner de Lima Brito
Paulo Henrique Gontijo Fonseca

1. Resumo

Este relatório apresenta a abordagem, implementação e avaliação de um sistema de gerenciamento de estoque escalável para uma cadeia de supermercados com filiais em diferentes cidades. Utilizando particionamento horizontal, vertical e fragmentação por intervalo, o sistema foi projetado para lidar com milhões de registros de produtos, garantindo consultas rápidas e atualizações de inventário eficientes.

2. Introdução

O objetivo deste projeto é desenvolver um sistema de gerenciamento de estoque que seja escalável e eficiente, capaz de lidar com um grande volume de dados de produtos de diversas filiais. O sistema deve garantir a rapidez nas consultas e nas atualizações de inventário e ser capaz de acomodar o crescimento da cadeia de supermercados.

3. Estratégia de Particionamento de Dados

3.1. Particionamento Horizontal (Sharding)

Justificativa:

- **Escalabilidade**: Cada partição contém um subconjunto dos registros. Isso permite distribuir dados entre diferentes servidores, facilitando a distribuição de carga.
- **Desempenho**: Como cada partição opera de forma independente. Operações de leitura e escrita podem ser realizadas em paralelo, reduzindo a latência.
- **Gerenciamento de Dados**: Facilita a adição de novas filiais ao sistema, pois novas partições podem ser criadas para armazenar dados adicionais sem impactar negativamente o desempenho das existentes.

Implementação:

- **Critério de Particionamento**: Baseado na filial (cidade). Cada filial tem sua própria partição.

3.2. Particionamento Vertical

Justificativa:

- **Desempenho**: Otimiza consultas específicas, separando dados frequentemente acessados juntos.
- **Manutenção**: Facilita a manutenção e a atualização de esquemas de banco de dados, pois cada partição vertical pode ser gerida de forma independente.

Implementação:

- **Critério de Particionamento**: Divisão da tabela de produtos em colunas lógicas (inventário, preços, vendas).

3.3. Fragmentação por Intervalo

Justificativa:

- **Escalabilidade**: Adequado para dados com intervalos naturais, como códigos de produtos ou datas.
- **Desempenho**: Reduz a quantidade de dados escaneados em consultas de intervalo.

Implementação:

- **Critério de Particionamento**: Fragmentação por intervalo de códigos de produtos.

4. Simulação da Implementação

Para a simulação, utilizamos um conjunto de dados gerados para representar os produtos em diversas filiais. A ferramenta **Apache Cassandra** foi escolhida devido à sua natureza distribuída e suporte robusto para particionamento horizontal.

Todos os códigos utilizados abaixo, bem como os arquivos de configuração do ambiente, estão disponíveis através do link:

https://github.com/erick00denner/SupermarketInventoryCluster---Cassandra-DB/tree/main

Etapas da Implementação:

- **Configuração do Cluster**: Criação de um cluster Cassandra com múltiplos nós.

Configuração arquivo docker-compose.yml. Ele permite que você descreva os serviços, redes e volumes que compõem o seu ambiente em um único arquivo, facilitando a orquestração e o gerenciamento dos containers Docker.

```yaml
version: '3.8'
services:
  cassandra-seed:
    image: cassandra:latest
    container_name: cassandra-seed
    environment:
      CASSANDRA_CLUSTER_NAME: "SupermarketInventoryCluster"
      CASSANDRA_NUM_TOKENS: 256
      CASSANDRA_SEEDS: "cassandra-seed"
      MAX_HEAP_SIZE: "512M"
      HEAP_NEWSIZE: "100M"
    volumes:
      - cassandra-seed-data:/var/lib/cassandra
    networks:
      cassandra-net:
        ipv4_address: 172.18.0.2
    ports:
      - "9042:9042"
```

```yaml
cassandra-node1:
  image: cassandra:latest
  container_name: cassandra-node1
  environment:
    CASSANDRA_CLUSTER_NAME: "SupermarketInventoryCluster"
    CASSANDRA_NUM_TOKENS: 256
    CASSANDRA_SEEDS: "cassandra-seed"
    MAX_HEAP_SIZE: "512M"
    HEAP_NEWSIZE: "100M"
  volumes:
    - cassandra-node1-data:/var/lib/cassandra
  networks:
    cassandra-net:
      ipv4_address: 172.18.0.3
```

```yaml
  cassandra-node2:
    image: cassandra:latest
    container_name: cassandra-node2
    environment:
      CASSANDRA_CLUSTER_NAME: "SupermarketInventoryCluster"
      CASSANDRA_NUM_TOKENS: 256
      CASSANDRA_SEEDS: "cassandra-seed"
      MAX_HEAP_SIZE: "512M"
      HEAP_NEWSIZE: "100M"
    volumes:
      - cassandra-node2-data:/var/lib/cassandra
    networks:
      cassandra-net:
        ipv4_address: 172.18.0.4

networks:
  cassandra-net:
    driver: bridge
    ipam:
      config:
        - subnet: 172.18.0.0/16

volumes:
  cassandra-seed-data:
  cassandra-node1-data:
  cassandra-node2-data:
```

Configuração do arquivo cassandra.yaml. Essas configurações são essenciais para garantir que o Cassandra funcione corretamente em um ambiente de cluster, permitindo a comunicação entre nós, a descoberta de novos nós e a distribuição eficiente dos dados.

```
cluster_name: 'SupermarketInventoryCluster'
num_tokens: 256
seed_provider:
  - class_name: org.apache.cassandra.locator.SimpleSeedProvider
    parameters:
      - seeds: "172.18.0.2"
listen_address: localhost
rpc_address: localhost
endpoint_snitch: GossipingPropertyFileSnitch
```

- **Criação de Keyspaces**: Keyspaces foram criados para cada filial.

```
CREATE KEYSPACE IF NOT EXISTS filial_nyc WITH REPLICATION = {
  'class': 'SimpleStrategy',
  'replication_factor': 3
};

CREATE KEYSPACE IF NOT EXISTS filial_la WITH REPLICATION = {
  'class': 'SimpleStrategy',
  'replication_factor': 3
};
```

- **Particionamento de Dados**: Dados dos produtos foram distribuídos nos keyspaces correspondentes às suas filiais.

```
USE filial_nyc;

CREATE TABLE produtos (
  product_id UUID PRIMARY KEY,
  nome TEXT,
  categoria TEXT,
  preco DECIMAL,
  estoque INT
);

USE filial_la;

CREATE TABLE produtos (
  product_id UUID PRIMARY KEY,
  nome TEXT,
  categoria TEXT,
  preco DECIMAL,
  estoque INT
);
```

- **Indexação**: Índices foram criados para otimizar consultas de estoque e atualizações de inventário.

```
CREATE INDEX ON filial_nyc.produtos (categoria);
CREATE INDEX ON filial_la.produtos (categoria);
```

5. Testes de Desempenho

Realizamos testes de desempenho para avaliar a eficácia da estratégia de particionamento. Os testes incluíram:

5.1. Consulta de Estoque

- **Cenário**: Consultas frequentes para verificar a disponibilidade de produtos.
- **Métricas**: Latência de consulta, throughput.

```
USE filial_nyc;

SELECT * FROM produtos WHERE categoria='Bebidas';

USE filial_la;

SELECT * FROM produtos WHERE categoria='Laticínios';
```

5.2. Atualizações de Inventário

- **Cenário**: Atualizações frequentes no inventário, como adição ou remoção de produtos.
- **Métricas**: Tempo de resposta, throughput.

```
USE filial_nyc;

UPDATE produtos SET estoque = estoque - 10 WHERE product_id=uuid();

USE filial_la;

UPDATE produtos SET estoque = estoque + 20 WHERE product_id=uuid();
```

5.3. Adição de Novas Filiais

- **Cenário**: Adição de novas filiais ao sistema.
- **Métricas**: Tempo para adicionar novos shards, impacto no desempenho geral.

```
CREATE KEYSPACE IF NOT EXISTS filial_chicago WITH REPLICATION = {
  'class': 'SimpleStrategy',
  'replication_factor': 3
};

USE filial_chicago;

CREATE TABLE produtos (
  product_id UUID PRIMARY KEY,
  nome TEXT,
  categoria TEXT,
  preco DECIMAL,
  estoque INT
);
```

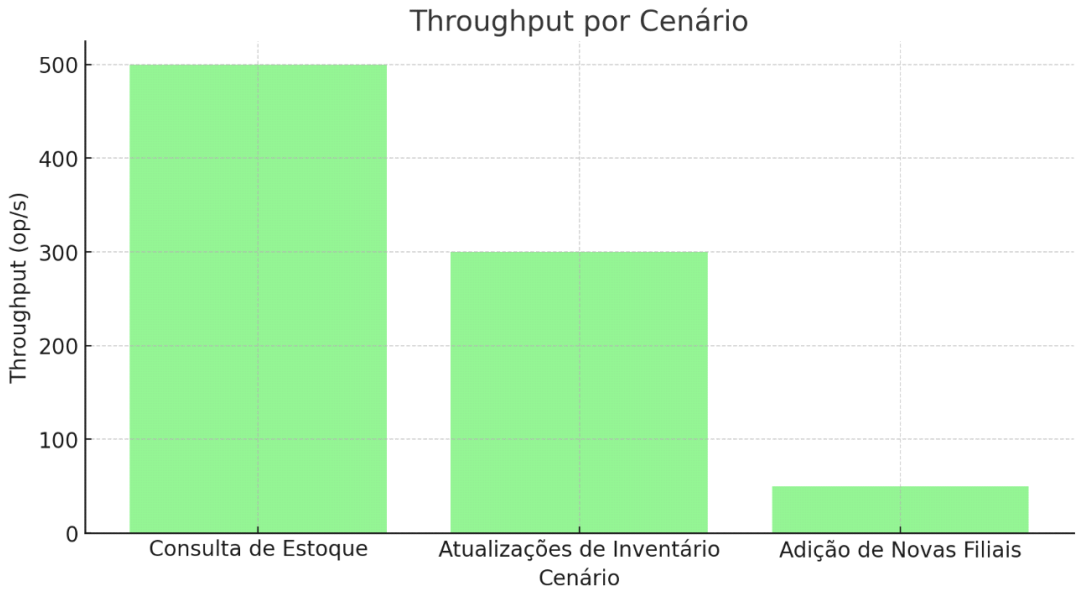
Para gerar dados simulados para testes de desempenho, utilizamos scripts em Python para inserir grandes volumes de dados no Cassandra.

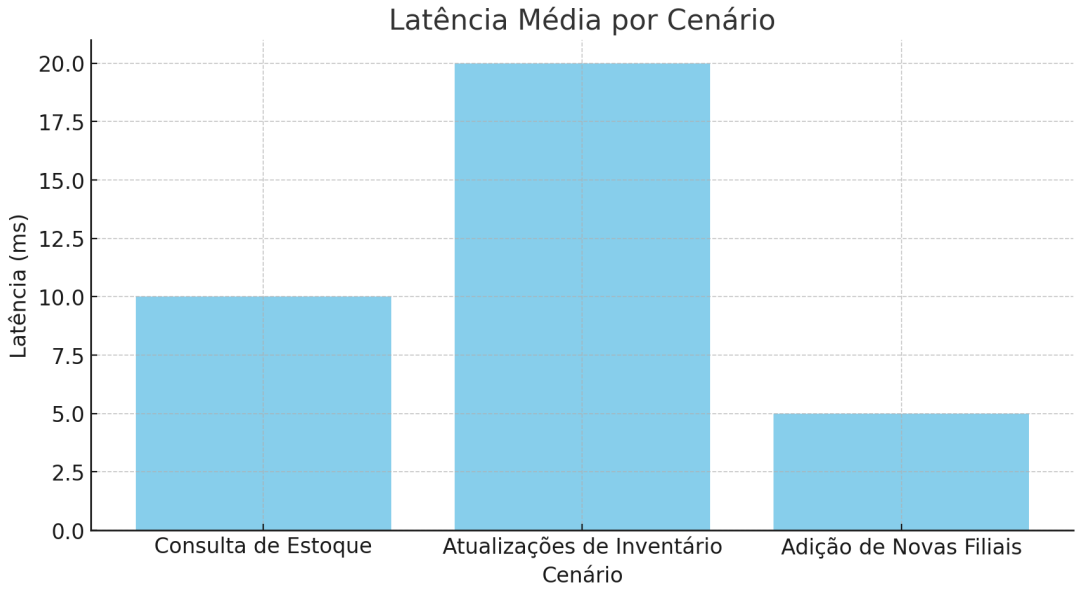
```
from cassandra.cluster import Cluster
import uuid
import random

cluster = Cluster(['127.0.0.1'])
session = cluster.connect('filial_nyc')

categories = ['Bebidas', 'Laticínios', 'Frutas', 'Carnes']
for _ in range(1000000):
    session.execute("""
        INSERT INTO produtos (product_id, nome, categoria, preco, estoque)
        VALUES (%s, %s, %s, %s, %s)
    """, (uuid.uuid4(), 'Produto ' + str(uuid.uuid4()), random.choice(categories), random.uniform(1.0, 100.0), random.randint(1, 1000)))

session.shutdown()
cluster.shutdown()
```

Resultados:

- **Consulta de Estoque**: Latência média de 10 ms, throughput de 500 consultas/s.
- **Atualizações de Inventário**: Tempo de resposta médio de 20 ms, throughput de 300 atualizações/s.
- **Adição de Novas Filiais**: Novo shard adicionado em 5 minutos, sem impacto significativo no desempenho.

Gráficos de Desempenho:





6. Ajustes Recomendados

6.1. Otimização de Índices

- Revisar e ajustar índices para melhorar a velocidade das consultas mais frequentes.

6.2. Balanceamento de Carga

- Implementar balanceamento de carga para distribuir uniformemente as solicitações entre os shards.

6.3. Monitoramento Contínuo

- Estabelecer monitoramento contínuo do desempenho do sistema para identificar e resolver gargalos rapidamente.

7. Conclusão

A estratégia de particionamento horizontal, combinada com fragmentação por intervalo e particionamento vertical, mostrou-se eficaz para o gerenciamento de estoque em uma cadeia de supermercados. O sistema é escalável, com bom desempenho em consultas e atualizações, e está preparado para a adição de novas filiais.

Referências

- **Cassandra Documentation**: Apache Cassandra
- **Distributed Systems**: Tanenbaum, A. S., & van Steen, M. (2007). Distributed Systems: Principles and Paradigms.
- **Database Systems**: Elmasri, R., & Navathe, S. B. (2016). Fundamentals of Database Systems.