UNIVERSIDADE FEDERAL
DE SANTA CATARINA

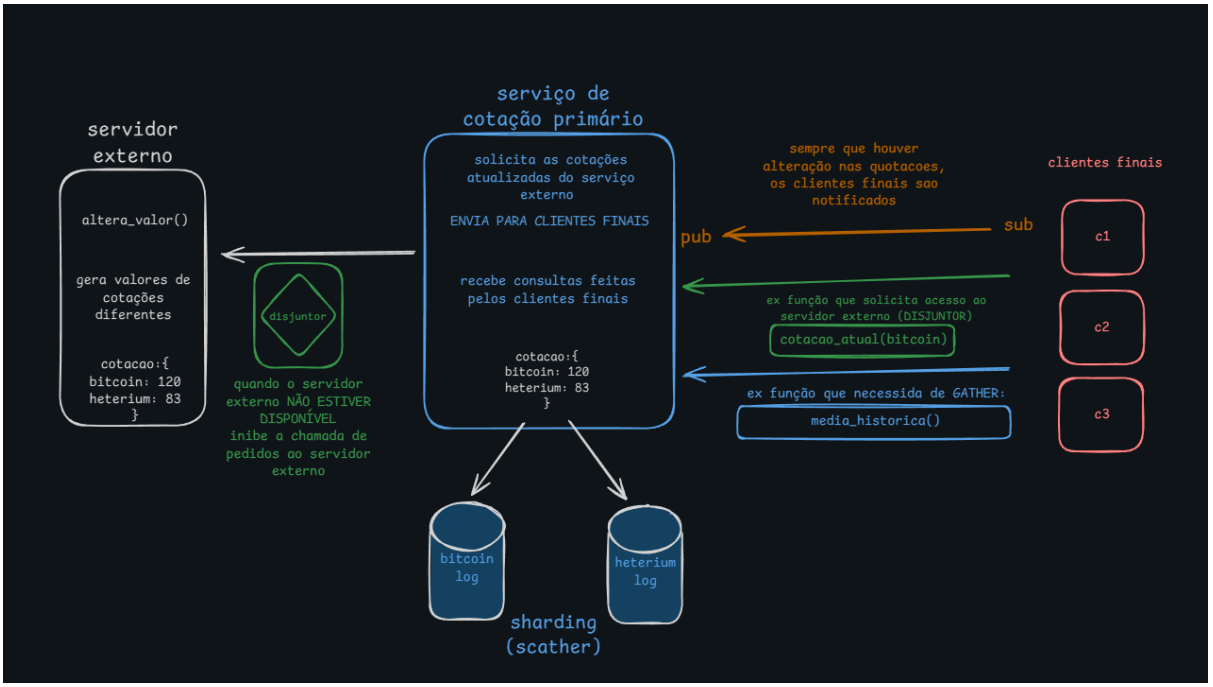
Universidade Federal de Santa Catarina
Centro de Tecnológico - CTC - Sistemas de Informação
INE 5645 - Programação Paralela e Distribuída
Professora Odorico Machado Mendizabal
Ana Luiza Bastos - 23203396
Rafael Nascimento Siqueira Pinto - 23203399

Relatório Técnico: Sistema de Cotações Distribuído

1. Estratégia de Implementação e Arquitetura

O sistema foi arquitetado seguindo o modelo centralizado com offloading de persistência. A solução é composta por três tipos de nós independentes que se comunicam via sockets TCP/IP. O sistema foi arquitetado seguindo o diagrama abaixo:

Figura 1: Diagrama do Sistema de Cotação



Componentes:

1. **Servidor Principal (primary_server):** Atua como orquestrador e *Message Broker*. Sua arquitetura é baseada em eventos (Event-Driven) utilizando **I/O Multiplexing** (via poll()). Isso permite gerenciar múltiplas conexões de clientes simultaneamente sem o overhead de criação de threads por conexão.

2. **Serviço Externo (cripto_server):** Simula uma API de Bolsa de Valores instável (com injeção de latência e falhas aleatórias) para testar a resiliência do sistema.
3. **Armazenamento (db_server):** Instâncias de banco de dados em memória (Shards) que persistem o histórico de cotações.

Protocolo de Comunicação:

Foi definido um protocolo de aplicação textual para facilitar a depuração e interoperabilidade.

- **Requisições:** SUBSCRIBE <TÓPICO>, SALVAR <VALOR>, LER <QTD>.
- Respostas: Strings formatadas (ex: OK, JSON, ou CSV delimitado por ;).
  A serialização é feita via sprintf e o parsing via sscanf, garantindo simplicidade e baixo custo de processamento.

2. Padrões de Projeto: Justificativa e Adoção

O sistema implementa quatro padrões fundamentais de sistemas distribuídos para atender aos requisitos de escalabilidade e tolerância a falhas.

2.1. Circuit Breaker (Disjuntor)

- **Problema Resolvido:** Impede o esgotamento de recursos (threads/sockets) no servidor principal quando o serviço externo (cripto_server) falha ou apresenta alta latência. Evita o efeito de espera em cascata.
- **Implementação e Adoção:** * Foi implementada uma máquina de estados completa: **Closed** (Normal), **Open** (Falha) e **Half-Open** (Recuperação).
  - **Fail Fast:** Se o circuito está aberto, o sistema retorna erro imediato ou dados de cache, sem tentar abrir conexão TCP.
  - **Parâmetros:** O circuito abre após 3 falhas consecutivas e possui um ***timeout*** de reset de **10 segundos**. O estado *Half-Open* permite testar a recuperação do serviço externo com uma única requisição antes de reabrir o tráfego total.

2.2. Publisher/Subscriber (Pub/Sub)

- **Problema Resolvido:** Desacopla a produção de dados (cotações chegando) do consumo (clientes visualizando), permitindo escalabilidade no número de leitores sem impactar a lógica de monitoramento.
- **Implementação e Adoção:** * O primary_server mantém uma tabela de assinaturas em memória (struct ClienteSubscriber).
  - **Filtragem de Tópicos:** Clientes enviam comandos SUBSCRIBE BTC, ETH ou TODOS.
  - **Difusão:** No ciclo de atualização, o servidor itera sobre a lista de descritores de arquivo (pollfd). Se o descritor estiver ativo e o tópico coincidir com a atualização recebida, a mensagem é enviada de forma assíncrona.

2.3. Sharding (Particionamento)

- **Problema Resolvido:** Distribui a carga de escrita e o volume de armazenamento, evitando que um único banco de dados se torne o gargalo (bottleneck) do sistema.

- **Implementação e Adoção:** * Utilizou-se **Particionamento Baseado em Chave** (Key-Based Partitioning).
    - Uma tabela de roteamento estática mapeia a chave da moeda para a porta do serviço correspondente:
        - Chave "BTC" -> Shard na porta 9801.
        - Chave "ETH" -> Shard na porta 9802.
    - Isso garante isolamento de falhas: se o Shard BTC cair, o histórico de ETH continua acessível.

2.4. Scatter/Gather (Espalhar e Reunir)

- **Problema Resolvido:** Permite responder a consultas complexas que exigem dados fragmentados em múltiplos nós (ex: "Média Geral do Mercado").
- **Implementação e Adoção:** * Utilizado na funcionalidade media_historica.
    - **Scatter (Sequencial):** O servidor solicita os últimos 10 registros ao Shard BTC e, em seguida, ao Shard ETH. Optou-se por chamadas sequenciais (bloqueantes) pela simplicidade de implementação dado o baixo número de shards.
    - **Gather:** O servidor agrega os dados brutos recebidos, calcula as médias aritméticas localmente e compila um relatório unificado em texto para o cliente. O cliente final não tem visibilidade da complexidade dessa distribuição.
    -

3. Conclusão e Considerações Finais

O desenvolvimento do trabalho permitiu observar na prática os *trade-offs* de sistemas distribuídos. A introdução do **Circuit Breaker** aumentou significativamente a resiliência da aplicação: durante os testes de caos (onde o mock falhava 20% das vezes), o servidor principal permaneceu responsivo, servindo dados em cache.

O uso de **Sharding** e **Scatter/Gather** demonstrou como dividir a complexidade de dados, embora tenha introduzido a complexidade adicional de gerenciar múltiplas conexões de banco de dados e a necessidade de lógica de agregação no nível da aplicação.