Rede Social com Publicações e Troca de Mensagens

Este projeto implementa uma rede social utilizando um sistema distribuído que permite a interação entre usuários por meio de publicações, troca de mensagens privadas e notificações em tempo real. O sistema utiliza múltiplos servidores para garantir alta disponibilidade, replicação dos dados e consistência dos eventos. Além disso, ele incorpora mecanismos de sincronização de relógios (utilizando o algoritmo de Berkeley com eleição de coordenador via Bully) e ordenação de mensagens via relógios lógicos.

Padrão de Mensagem Utilizado

As mensagens trocadas entre os componentes seguem um formato baseado em strings delimitadas por “|”, onde cada campo tem uma função específica. Os padrões de mensagens são padronizados para garantir integridade e facilitar o processamento.

Estrutura Geral das Mensagens

<TIPO>|<ID_ORIGEM>|<ID_DESTINO>|<TIMESTAMP>|<DADOS>

TIPO: Indica o tipo da mensagem (PUB, SEGUIR, PRIV, ELEC, SYNC, etc.).

ID_ORIGEM: Identificador do remetente (usuário ou servidor).

ID_DESTINO: Identificador do destinatário (pode ser um usuário específico ou "ALL" para envio global).

DADOS: Conteúdo da mensagem (texto da publicação, mensagem privada, evento de eleição, etc.).

TIMESTAMP: Marca de tempo lógica ou física para ordenação**.**

Funcionamento

Broker (Java)

- Função:
    - Atua como intermediário entre clientes e servidores, encaminhando mensagens.
    - Possui dois loops:

        - ReplicationLoop: Recebe mensagens no canal PULL (porta 5571) e as retransmite via PUB (porta 5570) para a replicação entre servidores.
        - NotificationLoop: Recebe notificações via PULL (porta 5581) e envia via PUB (porta 5560) para os clientes.

- Execução:

    - Compilado com o JDK e dependente da biblioteca JeroMQ (disponível em lib/jeromq-0.5.1.jar).

    - Inicia um proxy para encaminhar mensagens entre o frontend (ROUTER, porta 5555) e o backend (DEALER, porta 5556).

Servidor (Python)

- Função:
    - Implementa a lógica de eleição para definir um coordenador entre os servidores.
    - O coordenador sincroniza os relógios dos servidores periodicamente.

- Execução:
    - Pode ser executado diretamente com Python.
    - Exemplo de execução: python src/python/server.py.

Cliente (C++)

- Função:
    - Conecta-se ao Broker para receber notificações e, possivelmente, interagir com o servidor.
    - Usa ZeroMQ (através do libzmq) para comunicação.

- Compilação:
    - O cliente em C++ é compilado usando o Visual Studio, referenciando as bibliotecas e headers instalados pelo vcpkg.
    - Exemplo de compilação (em um script ou pelo terminal):
        - cl /EHsc client.cpp /I"C:\vcpkg\installed\x64-windows\include" /link /LIBPATH:"C:\vcpkg\installed\x64-windows\lib" libzmq.lib