SISTEMAS DISTRIBUÍDOS

Guilherme Couto Gomes
RA: 22.122.035-3

Pedro Henrique Algodoal Pinto
RA: 22.122.072-6

Samir Oliveira da Costa
RA: 22.122.030-4

1. Visão Geral

Este sistema implementa uma rede social distribuída com alta disponibilidade, consistência de dados e sincronização de relógios. O sistema permite que usuários publiquem mensagens, sigam outros usuários, enviem mensagens privadas e recebam notificações, tudo isso em um ambiente distribuído com múltiplos servidores.

1.1 Características Principais

- **Alta Disponibilidade:** Múltiplos servidores com balanceamento de carga
- **Consistência:** Replicação automática de dados entre servidores
- **Sincronização:** Algoritmo de Berkeley para sincronização de relógios
- **Ordenação:** Relógios lógicos de Lamport para ordenação de eventos
- **Tolerância a Falhas:** Eleição de coordenador via algoritmo Bully
- **Escalabilidade:** Adição/remoção dinâmica de servidores

1.2 Funcionalidades do Sistema

Usuários

- Publicação de textos associados ao usuário com timestamp
- Sistema de seguir outros usuários com notificações
- Envio de mensagens privadas com entrega confiável e ordenada
- Visualização de feed personalizado

Servidores

- Replicação de mensagens e postagens em pelo menos três servidores
- Adição e remoção dinâmica de servidores

- Sincronização de relógios via algoritmo de Berkeley
- Eleição automática de coordenador via algoritmo Bully

2. Camadas da Arquitetura

1. **Camada de Apresentação:** Clientes Python e JavaScript
2. **Camada de Balanceamento:** LoadBalancer com algoritmo Round Robin
3. **Camada de Serviços:** Serviços especializados em cada servidor
4. **Camada de Dados:** Repositórios com persistência em arquivos
5. **Camada de Sincronização:** Algoritmos de Berkeley e Bully

3. Componentes Principais

3.1 Balanceador de Carga (LoadBalancer)

Responsabilidades:

- Distribuir requisições entre servidores ativos
- Monitorar saúde dos servidores
- Implementar algoritmo Round Robin

Configuração:

balancer.properties

server.id=balancer

server.address=localhost

server.port=5000

is.balancer=true

sync.port=6030

seed.servers=server1:localhost:6038,server2:localhost:6040,server3:localhost:6042

3.2 Serviços de Aplicação

UserService

- **Porta:** Base + 300 (ex: 5855)

- **Funções:** Registro e autenticação de usuários
- **Endpoints:** USER_REGISTER, USER_LOGIN

PostService

- **Porta:** Base + 0 (ex: 5555)
- **Funções:** Gerenciamento de publicações
- **Endpoints:** CREATE_POST, UPDATE_POST, DELETE_POST, GET_USER_POSTS, GET_FEED

MessageService

- **Porta:** Base + 100 (ex: 5655)
- **Funções:** Mensagens privadas entre usuários
- **Endpoints:** SEND_MESSAGE, MARK_AS_READ, GET_CONVERSATION, GET_UNREAD_MESSAGES

FollowService

- **Porta:** Base + 200 (ex: 5755)
- **Funções:** Relações de seguidor/seguindo
- **Endpoints:** FOLLOW_USER, UNFOLLOW_USER, GET_FOLLOWERS, GET_FOLLOWING

3.3 Repositórios de Dados

Cada repositório gerencia a persistência de um tipo específico de dados:

- **UserRepository:** Dados de usuários
- **PostRepository:** Publicações
- **MessageRepository:** Mensagens privadas

4. Algoritmos de Sincronização

4.1 Algoritmo de Berkeley

Objetivo: Sincronizar relógios físicos entre servidores

Funcionamento:

1. O coordenador solicita o tempo de todos os servidores
2. Calcula a média das diferenças de tempo
3. Envia ajustes individuais para cada servidor

4. Cada servidor aplica o ajuste ao seu relógio

Implementação: A classe BerkeleyAlgorithm gerencia todo o processo de sincronização, executando periodicamente conforme configurado no arquivo de propriedades.

4.2 Algoritmo Bully (Eleição de Coordenador)

Objetivo: Eleger um coordenador para sincronização de relógios

Funcionamento:

1. Servidor detecta falha do coordenador
2. Inicia eleição enviando mensagens para servidores com ID maior
3. Se nenhum responde, se declara coordenador
4. Notifica todos os outros servidores

4.3 Relógios Lógicos de Lamport

Objetivo: Ordenação consistente de eventos distribuídos

Regras:

- Incrementa relógio antes de evento local
- Atualiza relógio ao receber mensagem: max(local, recebido) + 1
- Anexa timestamp lógico em todas as mensagens

5. APIs e Comunicação

Protocolo de Comunicação

O sistema utiliza ZeroMQ com padrões REQ/REP para comunicação síncrona entre componentes. Todas as mensagens seguem o formato JSON com timestamps lógicos para ordenação.

6. Replicação de Dados

6.1 Processo de Replicação

1. **Evento Local:** Operação executada em um servidor
2. **Registro:** Evento registrado no ReplicationManager
3. **Fila:** Evento adicionado à fila de replicação

4. **Distribuição:** Evento enviado para todos os outros servidores
5. **Aplicação:** Outros servidores aplicam o evento localmente

6.2 Tipos de Eventos Replicados

Evento	Descrição	Dados Replicados
USER_CREATED	Criação de usuário	username, password, createdA
POST_CREATED	Nova publicação	id, username, content, timesta
POST_UPDATED	Atualização de publicação	id, content, updatedAt
POST_DELETED	Remoção de publicação	id
MESSAGE_SENT	Envio de mensagem	id, sender, receiver, content, tir
FOLLOW_ADDED	Nova relação de seguidor	follower, followed
FOLLOW_REMOVED	Remoção de relação de seguidor	follower, followed

6.3 Garantias de Consistência

- **Eventual Consistency:** Todos os servidores convergem para o mesmo estado
- **Causal Ordering:** Eventos causalmente relacionados são aplicados na ordem correta
- **Duplicate Detection:** Eventos duplicados são ignorados baseado no ID da entidade
- **Conflict Resolution:** Timestamps lógicos resolvem conflitos de ordenação

7. Clientes

7.1 Cliente Python

Localização: social_network_client.py

Características:

- Interface de linha de comando interativa
- Suporte a todas as funcionalidades da rede social
- Tratamento de erros robusto
- Validação de entrada do usuário

Uso:

python social_network_client.py

7.2 Cliente JavaScript

Localização: social_network_client.js

Características:

- Interface assíncrona com Node.js
- Suporte completo às APIs
- Tratamento de sinais de interrupção
- Gerenciamento automático de conexão

Instalação e Uso:

npm install zeromq

node social_network_client.js

7.3 Funcionalidades dos Clientes

Autenticação

- Registro de novos usuários
- Login/logout seguro
- Validação de credenciais

Publicações

- Criação de posts
- Visualização de posts próprios
- Visualização de posts de outros usuários
- Feed personalizado baseado em seguindo

Sistema Social

- Seguir/deixar de seguir usuários
- Visualizar lista de seguidores
- Visualizar lista de usuários seguidos

Mensagens

- Envio de mensagens privadas
- Visualização de conversas
- Listagem de mensagens não lidas
- Marcação de mensagens como lidas

8. Tolerância a Falhas

8.1 Detecção de Falhas

Mecanismos de Detecção

- **Heartbeat:** Coordenador envia heartbeats periódicos
- **Timeout:** Servidores detectam falha por timeout de comunicação
- **Health Check:** Balanceador monitora saúde dos servidores
- **Ping/Pong:** Verificação ativa de conectividade

Configuração de Timeouts

Tipo	Timeout	Descrição
Heartbeat	30s	Intervalo entre heartbeats do coordenador
Comunicação	3s	Timeout para operações de rede
Eleição	5s	Timeout para processo de eleição
Descoberta	15s	Intervalo de descoberta de servidores

8.2 Recuperação de Falhas

Tipos de Falha e Recuperação

1. **Falha de Servidor:**

- Balanceador remove servidor da rotação
- Requisições redirecionadas para servidores ativos
- Dados permanecem disponíveis nos outros servidores

2. **Falha de Coordenador:**
   - Algoritmo Bully inicia nova eleição
   - Servidor com maior ID assume coordenação
   - Sincronização de relógios continua
3. **Falha de Balanceador:**
   - Clientes podem conectar diretamente aos servidores
   - Sistema continua operacional com funcionalidade reduzida
4. **Partição de Rede:**
   - Servidores continuam operando independentemente
   - Dados são sincronizados após reconexão
   - Conflitos resolvidos via timestamps

8.3 Reconexão e Sincronização

Processo de Reconexão:

1. Servidor detecta reconexão de par
2. Troca de informações de estado
3. Sincronização de dados perdidos
4. Atualização de relógios
5. Retomada de operação normal

Garantia de Consistência: O sistema garante que após a reconexão, todos os servidores convergem para o mesmo estado através do mecanismo de replicação eventual.