UNIVERSIDADE ESTADUAL DE MATO GROSSO DO SUL

CURSO DE CIÊNCIA DA COMPUTAÇÃO

DISCIPLINA DE PROGRAMAÇÃO DISTRIBUÍDA

Prof. Dr. Rubens Barbosa Filho.

TRABALHO: IMPLEMENTAÇÃO DE UM BITTORRENT

DOURADOS(MS), 09/09/2026

Data final para o trabalho: 06/11/2026.

ESPECIFICAÇÃO DE ARQUITETURA DE SOFTWARE DISTRIBUÍDO

Sistema Distribuído P2P Híbrido para Compartilhamento Inteligente de Documentos

Baseado em

- DHT-Chord
- Gossip Protocol
- SHA-256
- Hash Table Distribuída
- LFU Cache
- Compressão LZ4
- Bully Algorithm
- Two Phase Commit (2PC)
- State Machine Replication
- Incremental State Transfer

INTRODUÇÃO

Objetivo do Projeto

O presente documento especifica a arquitetura de um sistema distribuído Peer-to-Peer híbrido destinado ao compartilhamento eficiente de documentos digitais utilizando técnicas modernas de computação distribuída, algoritmos de consenso, replicação determinística e descoberta dinâmica de recursos.

Diferentemente das arquiteturas P2P tradicionais, que frequentemente sofrem com problemas de escalabilidade, disponibilidade e consistência, o sistema proposto combina uma rede estruturada baseada em DHT-Chord com uma camada hierárquica de Super Peers, proporcionando alta eficiência na localização de recursos e tolerância a falhas.

O documento descreve todos os componentes arquiteturais, protocolos de comunicação, estruturas de dados e mecanismos necessários para permitir que o sistema seja implementado de forma consistente, modular e escalável.

Motivação

O crescimento exponencial da produção de documentos digitais torna cada vez mais relevante a construção de sistemas distribuídos capazes de armazenar, localizar e compartilhar informações de maneira descentralizada.

Arquiteturas centralizadas apresentam limitações relacionadas à disponibilidade, escalabilidade horizontal e resiliência, uma vez que dependem da operação contínua de servidores específicos.

Sistemas P2P eliminam parte dessas limitações, porém introduzem novos desafios, como:

- descoberta eficiente de recursos;
- sincronização distribuída;
- manutenção de consistência;
- eleição de coordenadores;
- tolerância a falhas;
- gerenciamento de metadados;
- balanceamento de carga;
- recuperação após falhas.

Este projeto busca integrar soluções consolidadas para esses desafios em uma arquitetura única e coerente.

Escopo

O sistema será responsável por:

- cadastrar nós distribuídos;
- permitir entrada e saída dinâmica de participantes;
- localizar documentos através de uma DHT;
- transferir documentos comprimidos;
- replicar metadados;
- detectar falhas automaticamente;
- eleger novos coordenadores;
- manter consistência distribuída.

Não faz parte do escopo:

- interface gráfica;
- autenticação federada;
- armazenamento em nuvem pública;
- criptografia ponta a ponta.

Objetivos

Objetivo Geral

Projetar uma arquitetura distribuída capaz de fornecer um ambiente robusto para compartilhamento de documentos PDF utilizando tecnologias modernas de computação distribuída.

Objetivos Específicos

- desenvolver uma arquitetura híbrida P2P;
- reduzir tráfego de pesquisa;
- eliminar pontos únicos de falha;
- garantir integridade dos documentos;
- minimizar tempo de recuperação;
- maximizar disponibilidade;
- facilitar expansão horizontal.

Premissas

A arquitetura assume:

- comunicação TCP/IP confiável;
- relógios locais independentes;
- nós potencialmente heterogêneos;
- falhas por parada (crash failures);
- perda eventual de mensagens;
- possibilidade de particionamento temporário da rede.

Restrições

O projeto deverá:

- operar em ambiente Linux;
- utilizar linguagem C;
- empregar POSIX Sockets;
- manter independência de banco de dados relacional;
- suportar múltiplas arquiteturas x86-64.

Visão Geral da Arquitetura

A arquitetura é organizada em nove subsistemas principais:

1. Cliente P2P
2. Super Peer
3. Overlay DHT-Chord
4. Gossip Manager
5. Metadata Manager

6. Consensus Manager
7. Replication Manager
8. Cache Manager
9. Compression Manager

Cada subsistema possui responsabilidades claramente definidas, reduzindo acoplamento e aumentando a modularidade.

REQUISITOS DO SISTEMA

Introdução

Esta introdução especifica todos os requisitos funcionais, não funcionais, restrições arquiteturais e atributos de qualidade do sistema distribuído.

A arquitetura foi concebida para atender aos seguintes princípios:

- Alta disponibilidade;
- Escalabilidade horizontal;
- Consistência dos metadados;
- Tolerância a falhas;
- Balanceamento de carga;
- Modularidade;
- Baixo acoplamento;
- Alta coesão.

Atores do Sistema

O sistema possui quatro atores principais.

Ator 1: Usuário

Responsável por:

- compartilhar documentos;
- pesquisar documentos;
- realizar download;
- remover documentos.

Ator 2: Peer

Máquina participante da rede.

Responsabilidades:

- armazenar documentos;

- responder pesquisas;
- enviar blocos;
- receber blocos;
- atualizar metadados.

Ator 3: Super Peer
Nó responsável pelo gerenciamento de um domínio.

Funções:

- manter índice local;
- gerenciar cache;
- executar Gossip;
- participar da DHT;
- executar replicação;
- executar consenso;
- participar da eleição.

Ator 4: Coordenador

Coordenador lógico eleito dinamicamente pelo algoritmo Bully.

Funções:

- coordenar transações 2PC;
- supervisionar replicação;
- registrar Super Peers;
- detectar inconsistências;
- iniciar recuperação.

Casos de Uso

UC-01 Registrar Peer

Descrição: Permite que um novo Peer entre na rede.

Fluxo principal

1. Peer envia JOIN.
2. Gossip localiza Super Peer.
3. Super Peer valida o nó.
4. Nó recebe NodeID.
5. Finger Table é atualizada.

6. Metadados são sincronizados.

UC-02 Publicar Documento

Fluxo: Upload → SHA-256 → Compressão LZ4 → Fragmentação → Hash Table → Replicação → Confirmação.

UC-03 Pesquisar Documento

Fluxo: Cliente → Super Peer → Hash Table → Chord Lookup → Localização → Resposta.

UC-04 Download

Fluxo: Lookup → Lista de Peers → Transferência Paralela → Descompressão → Validação SHA-256 → Documento.

UC-05 Entrada de Novo Nó

Etapas

- JOIN
- Gossip
- Finger Table
- Incremental State Transfer
- Atualização da DHT

UC-06 Saída de Nó

Etapas

- LEAVE
- Redistribuição
- Atualização da Finger Table
- Atualização da Hash Table

UC-07 Falha

Etapas: Heartbeat → Timeout → Gossip → Election → Recovery → Replicação.

Requisitos Funcionais

RF-001

Cadastrar novos Peers.

RF-002

Cadastrar novos Super Peers.

RF-003

Remover nós.

RF-004

Compartilhar documentos PDF.

RF-005

Gerar SHA-256 para cada documento.

RF-006

Armazenar metadados em Hash Table.

RF-007

Executar Lookup via DHT-Chord.

RF-008

Executar compressão LZ4.

RF-009

Executar descompressão automática.

RF-010

Fragmentar documentos.

RF-011

Transferência paralela.

RF-012

Executar Gossip Protocol.

RF-013

Atualizar Finger Tables.

RF-014

Replicar metadados.

RF-015

Executar State Machine Replication.

RF-016

Executar Incremental State Transfer.

RF-017

Executar Bully Algorithm.

RF-018

Executar Two Phase Commit.

RF-019

Executar Cache LFU.

RF-020

Detectar falhas automaticamente.

RF-021

Realizar balanceamento de carga.

RF-022

Executar Heartbeats.

RF-023

Executar recuperação automática.

RF-024

Atualizar estatísticas.

RF-025

Registrar logs distribuídos.

Requisitos Não Funcionais

RNF-001 Disponibilidade

Disponibilidade mínima: 99,5%

RNF-002 Escalabilidade

Suportar crescimento sem necessidade de reconfiguração manual.

RNF-003 Consistência

Garantir consistência forte para metadados.

RNF-004 Segurança

Validação SHA-256.

RNF-005 Portabilidade

Linux x86-64.

RNF-006 Modularidade

Cada componente será implementado como módulo independente.

RNF-007 Desempenho

Lookup médio:O(log N)

RNF-008 Replicação

Tempo máximo de sincronização incremental: < 2 segundos.

RNF-009 Compressão

LZ4 com throughput superior a 500 MB/s.

RNF-010 Recuperação

Eleição inferior a 5 segundos.

Restrições Arquiteturais

O sistema deverá utilizar obrigatoriamente:

- linguagem C;
- POSIX Threads;
- POSIX Sockets;
- SHA-256;
- LZ4;
- Hash Tables;
- Chord;
- Gossip;
- Bully;
- 2PC;

- Incremental State Transfer;
- State Machine Replication.

Critérios de Aceitação

O sistema será considerado operacional quando:

✓ realizar upload;

✓ localizar documentos;

✓ realizar downloads;

✓ recuperar falhas;

✓ eleger novo líder;

✓ manter consistência;

✓ sincronizar novos nós.

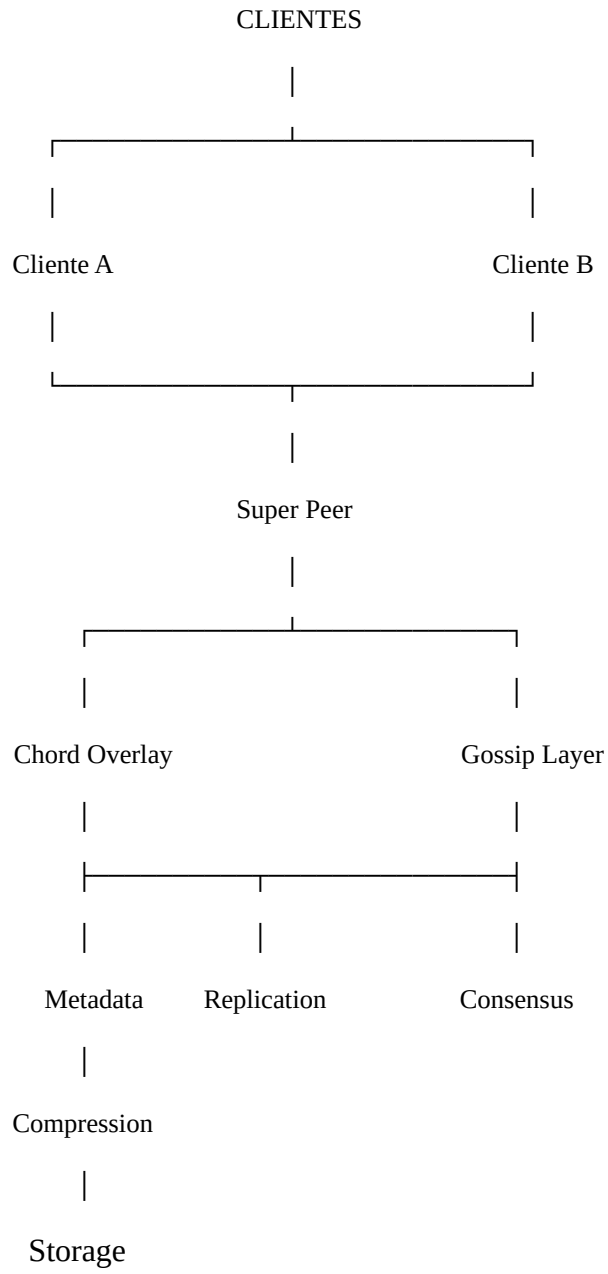
ARQUITETURA GERAL DO SISTEMA

Visão Geral

O sistema adota uma arquitetura híbrida composta por:

- Overlay DHT-Chord;
- Super Peers;
- Gossip Protocol;
- Coordenador dinâmico;
- Replicação determinística.
- Cada componente possui responsabilidades específicas.

Arquitetura Física



Arquitetura em Camadas

Application Layer → Service Layer → Metadata Layer → Replication Layer → Consensus Layer → Compression Layer → Network Layer → TCP/IP.
Cada camada abstrai os detalhes da inferior, promovendo modularidade e facilitando manutenção e evolução.

Componentes Arquiteturais

Cliente

Responsabilidades:

- upload;
- download;
- fragmentação;

- recomposição;
- pesquisa;
- validação SHA-256;
- comunicação com Super Peer.

Super Peer

Responsabilidades:

- Finger Table;
- Lookup;
- Hash Table;
- Gossip;
- Cache LFU;
- Replicação;
- Consenso.

Coordenador

Responsabilidades:

- iniciar 2PC;
- supervisionar State Machine Replication;
- controlar replicação;
- coordenar eleição;
- controlar/sincronizar logs.

DHT Manager

Responsável por:

- Join;
- Leave;
- Lookup;
- Stabilize;
- Notify.

Gossip Manager

Responsável por:

- Membership;

- Heartbeats;
- Disseminação;
- Failure Detection.

Metadata Manager

Gerencia:

- índices;
- Hash Table;
- estatísticas;
- versões.

Replication Manager

Executa:

- SMR;
- Snapshot;
- Incremental Transfer.

Consensus Manager

Executa:

- Prepare;
- Commit;
- Abort;
- Recovery.

Compression Manager

Executa:

- LZ4 Compress;
- LZ4 Decompress.

Cache Manager

Gerencia:

- LFU;
- contadores;

- substituição.

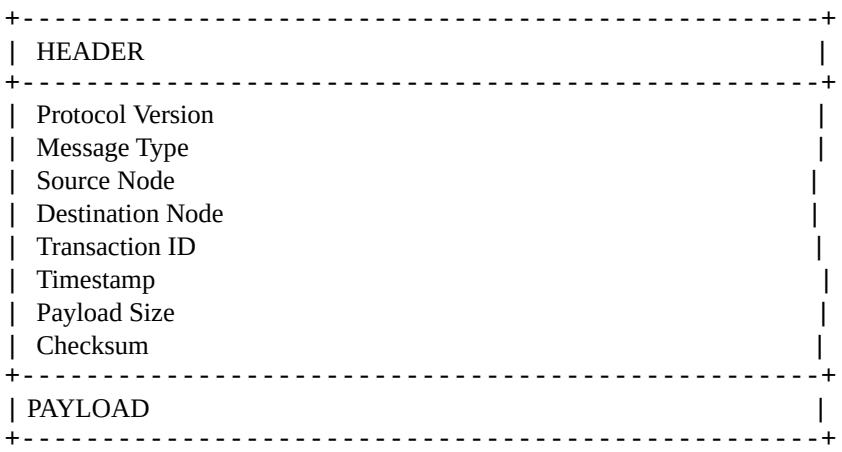
Modelo de Dados

Cada documento é representado por um objeto distribuído:

Documento → SHA-256 → ObjectID → Metadata Entry → Hash Table → Replicação

Modelo de Comunicação

Todas as mensagens seguem uma estrutura padronizada.



Fluxo Geral do Sistema

Upload → SHA-256 → Compressão → Hash Table → Chord → Replicação → SMR → 2PC → Commit → Disponibilização

Fluxo de Pesquisa

Cliente → Super Peer → Cache LFU → Hash Table → Chord Lookup → Peer Encontrado → Download

Fluxo de Recuperação

Heartbeat → Timeout → Gossip → Election → Novo Coordenador → State Transfer → Sistema Restaurado

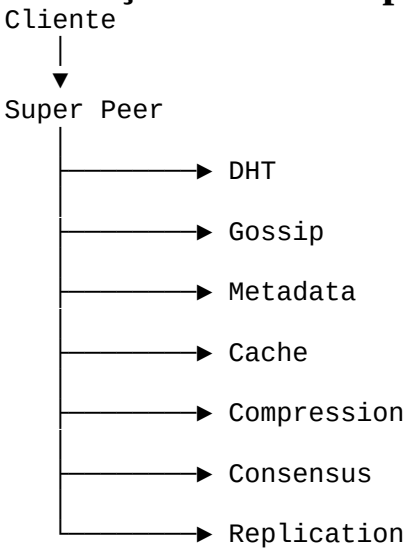
Fluxo de Entrada de Novos Nós

JOIN → Validação → NodeID → Finger Table → Incremental State Transfer → Atualização Hash Table → Operacional

Fluxo de Replicação

Evento → State Machine → Log → 2PC → Commit → Snapshot → Replicação Incremental → Confirmação

Interação entre Componentes



ARQUITETURA P2P HÍBRIDA

Objetivo

A arquitetura proposta combina características das arquiteturas Peer-to-Peer estruturadas e hierárquicas.

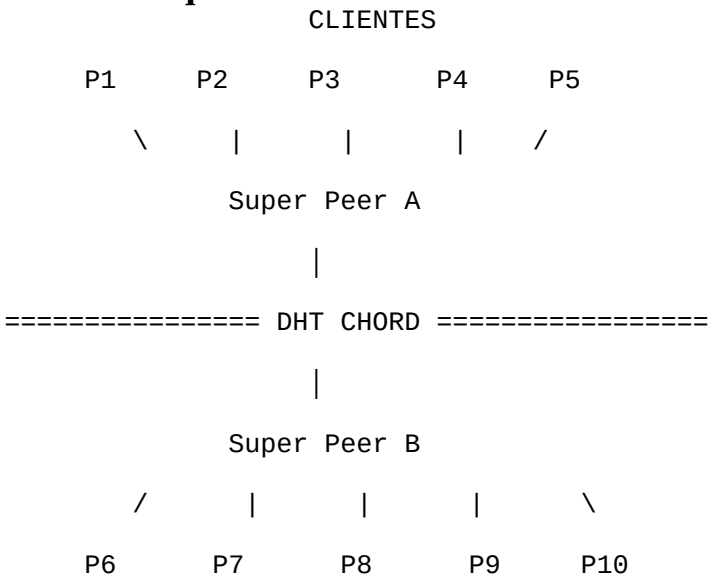
Seu objetivo é reunir:

- eficiência de localização;
- escalabilidade;
- tolerância a falhas;
- consistência distribuída;
- simplicidade operacional.

A rede é organizada em dois níveis:

- **Peers (Clientes)**: armazenam documentos e participam da transferência de arquivos.
- **Super Peers**: mantêm metadados, executam protocolos distribuídos e formam o overlay Chord.

Modelo Arquitetural



Cada Super Peer administra um domínio de clientes, enquanto todos os Super Peers participam do anel lógico Chord.

MODELO DE COMUNICAÇÃO DISTRIBUÍDA

Todos os componentes comunicam-se utilizando mensagens padronizadas sobre TCP/IP.

Tipos de Mensagem

Código	Descrição
JOIN	Entrada
LEAVE	Saída
LOOKUP	Pesquisa
STORE	Registro
DOWNLOAD_REQ	Solicitação
DOWNLOAD_REP	Resposta
PREPARE	2PC
COMMIT	2PC
ABORT	2PC
HEARTBEAT	Monitoramento
GOSSIP	Disseminação
ELECTION	Bully
OK	Resposta eleição
COORDINATOR	Novo líder
SNAPSHOT	Replicação
STATE_TRANSFER	Sincronização
ACK	Confirmação
ERROR	Erro

Estados de Comunicação
DISCONNECTED → CONNECTING → CONNECTED → AUTHENTICATED → SYNCHRONIZED → ACTIVE → LEAVING

Heartbeats
Cada Super Peer transmite Heartbeats periodicamente.

Intervalo: 5 segundos

Timeout: 15 segundos

Após o timeout:

- Gossip propaga a suspeita.
- Bully Election é iniciado.

Controle de Transações
Cada transação recebe um TransactionID único de 128 bits.

Formato: Timestamp + NodeID + Sequence Number

Controle de Integridade
Cada mensagem possui:

- CRC32 (integridade da mensagem)
- SHA-256 (integridade do conteúdo)

PROJETO DA DHT-CHORD

Objetivo
Eliminar pesquisas por inundação. Cada documento será localizado através de hashing consistente.

Espaço de Identificadores
O sistema utiliza SHA-256.

Portanto: $0 \rightarrow 2^{256}-1.$ Todos os nós pertencem ao mesmo espaço.

NodeID
Cada Super Peer recebe: NodeID = SHA256(IP || Porta || UUID)

ObjectID
Cada documento recebe: ObjectID = SHA256(Arquivo)

Lookup
Algoritmo: Lookup(ObjectID) → Finger Table → Closest Preceding Node → Forward → Successor → Owner.

Operações Fundamentais

JOIN

Novo Super Peer.

LEAVE

Saída ordenada.

STABILIZE

Atualização periódica.

NOTIFY

Atualização do predecessor.

FIX_FINGERS

Atualização das Finger Tables.

Redistribuição

Quando um Super Peer entra:

- recebe parte das chaves.

Quando sai:

- transfere todas.

GOSSIP PROTOCOL

Objetivo

O Gossip Protocol mantém o conhecimento global da rede sem necessidade de um servidor central.

Serviços

- descoberta;
- heartbeat;
- disseminação;
- detecção de falhas;
- atualização de estado.

Membership

Cada Super Peer mantém:

- Membership Table
- NodeID
- IP
- Port
- State
- LastHeartbeat
- Version

Ciclo Gossip

Seleciona Vizinho → Envia Estado → Recebe Estado → Mescla Informações → Atualiza Membership

Detecção de Falhas

Heartbeat → Timeout → Suspect → Gossip → Confirm → Remove Node

Estados
ALIVE → SUSPECT → FAILED → REMOVED

Integração com Bully
Quando Gossip confirma uma falha do coordenador:

FAILED → Election → Bully → Coordinator → Broadcast

Integração com Chord
Quando ocorre:

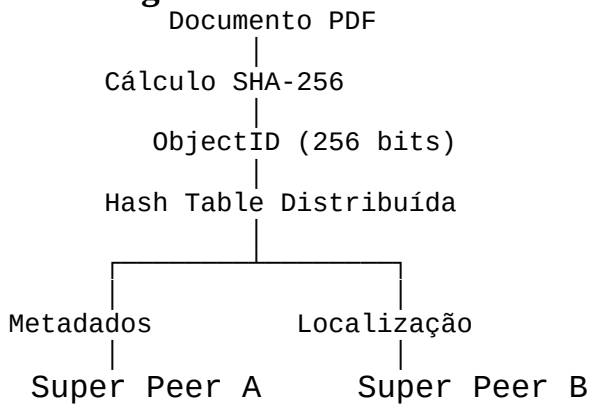
- entrada;
- saída;
- falha.

O Gossip solicita: Fix Fingers + Stabilize + Notify

mantendo a DHT consistente.

BANCO DE METADADOS DISTRIBUÍDO

Objetivo
O Banco de Metadados Distribuído (Distributed Metadata Repository - DMR) constitui o principal mecanismo de indexação do sistema. Sua função é armazenar apenas informações descritivas sobre os documentos, mantendo a localização lógica dos recursos sem armazenar os arquivos propriamente ditos. Cada Super Peer mantém uma réplica consistente do subconjunto de metadados sob sua responsabilidade, determinado pelo particionamento da DHT-Chord.

Arquitetura Lógica


Estrutura da Entrada de Metadados
Cada documento é representado por um registro estruturado:

Campo	Tipo	Descrição
ObjectID	SHA-256	Identificador único
Filename	String	Nome lógico
Version	uint64	Versão do objeto
Size	uint64	Tamanho em bytes
Compression	Enum	LZ4
OwnerPeer	NodeID	Proprietário original
ReplicaPeers	Lista	Réplicas
ChunkCount	uint32	Número de fragmentos
ChunkHash[]	SHA-256[]	Integridade dos blocos
UploadDate	Timestamp	Data de publicação
LastAccess	Timestamp	Último acesso
DownloadCounter	uint64	Estatística
Status	Enum	Ativo, Replicando, Removido

Operações

Inserção

1. Calcular SHA-256.
2. Calcular índice.
3. Inserir registro.
4. Replicar operação via SMR.
5. Confirmar via 2PC.

Consulta

Lookup → Hash Table → Encontrou?

Sim → Retorna Registro

Não → Consulta DHT

Atualização

As atualizações são versionadas utilizando **Version Vector**.

Cada modificação incrementa: Version++

Remoção

A remoção lógica marca o registro como **REMOVED**.

Índices Auxiliares

Além da chave principal (ObjectID), cada Super Peer mantém índices secundários:

- Nome do documento.

- Proprietário.
- Data de publicação.
- Número de downloads.
- Palavras-chave.
- Categoria.

Esses índices aceleram consultas locais.

Política de Replicação
Cada entrada possui fator de replicação configurável.

Exemplo: ReplicationFactor = 3

O sistema tenta manter três réplicas consistentes em Super Peers distintos.

CACHE DISTRIBUÍDO LFU

Objetivo
O Cache Manager reduz o número de consultas à DHT e ao Banco de Metadados, armazenando temporariamente registros frequentemente acessados.

Arquitetura
Lookup → LFU Cache → Hit?

Sim → Retorna

Não → Hash Table → Atualiza Cache

Algoritmo LFU
Cada acesso incrementa: Frequency++

Ao atingir o limite do cache:

Menor Frequência → Menor Último Acesso → Remover Entrada

Balanceamento
O cache é dividido em:

- Metadata Cache
- Lookup Cache
- Finger Cache
- Routing Cache

Cada região possui capacidade independente.

Atualização
Quando Gossip dissemina alterações:

Invalidate → Atualizar → Replicar

Métricas
- Hit Rate
- Miss Rate
- Tempo Médio
- Número de Evictions

SISTEMA DE COMPRESSÃO LZ4

Objetivo
Reduzir o volume de dados transmitidos na rede sem comprometer o desempenho.

Pipeline
Arquivo → Chunking → Compressão LZ4 → Checksum → Transmissão

Processo de Compressão
Cada fragmento é comprimido individualmente.

Chunk → LZ4 Compress → Compressed Chunk

Processo de Descompressão
Compressed Chunk → LZ4 Decode → SHA-256 → Validação → Entrega

Compressão Paralela
O Compression Manager utiliza um pool de threads.

Chunk 1 → Thread 1

Chunk 2 → Thread 2

Chunk 3 → Thread 3

Chunk N → Thread N

SISTEMA DISTRIBUÍDO DE TRANSFERÊNCIA DE ARQUIVOS

Objetivo
Permitir transferência paralela, segura e tolerante a falhas.

Fragmentação
Os documentos são divididos em blocos fixos.

Chunk Size = 4 MB

Estrutura
Documento → Chunk0 → Chunk1 → Chunk2 → … → ChunkN

Identificação
Cada fragmento recebe: ChunkID + SHA-256 + Offset

Upload
Cliente → SHA-256 → Chunking → Compressão → Metadata → Replicação →Commit

Download
Lookup → Lista de Peers → Transferência Paralela → Descompressão → Validação → Reconstrução

Recuperação
Caso um Peer falhe:

Timeout → Selecionar Réplica → Retomar Download

Controle de Integridade
Cada fragmento possui SHA-256 próprio.

Ao final: SHA256(Documento) → Comparação → Documento Válido.

Estados da Transferência
CREATED → QUEUED → STARTED → TRANSFERRING → VERIFYING → FINISHED → REPLICATED

STATE MACHINE REPLICATION (SMR)

Objetivo
A State Machine Replication (SMR) garante que todos os Super Peers mantenham exatamente o mesmo estado lógico do banco de metadados distribuído. Cada operação aprovada é registrada em um log e executada na mesma ordem por todas as réplicas. Dessa forma, qualquer Super Peer pode assumir o papel de coordenador sem perda de consistência.

Arquitetura da Replicação
```
Cliente
   |
   ▼
Super Peer Primário
   |
Registro em Log
   |
Two-Phase Commit
   |
Replicação para Réplicas
   |
Execução Determinística
   |
Estado Consistente
```

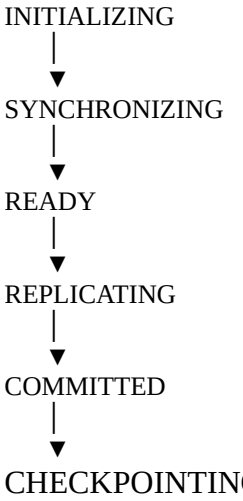
Log Distribuído
Cada comando confirmado gera um registro:

Campo	Descrição
LogIndex	Sequência global
TransactionID	Identificador único
Operation	STORE, DELETE, UPDATE
ObjectID	Documento
Timestamp	Tempo lógico
Version	Número da versão
Checksum	SHA-256 do comando

Os logs são mantidos em ordem crescente de LogIndex.

Máquina de Estados

Estados possíveis:



INCREMENTAL STATE TRANSFER (IST)

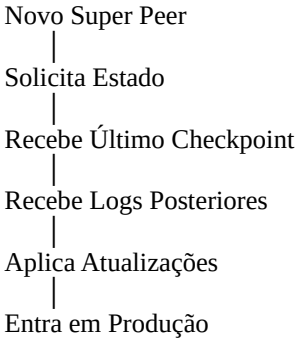
Objetivo

Evitar a cópia integral do banco de metadados sempre que um novo Super Peer entrar ou um nó recuperar-se de falha.

O mecanismo transfere apenas:

- logs pendentes;
- diferenças de estado;
- checkpoints recentes.

Fluxo



Sincronização
O coordenador identifica a versão do nó solicitante.

Exemplo:

```
Coordenador: versão 1050
Novo nó: versão 1032
```

O sistema envia apenas as operações de 1033 a 1050.

TWO-PHASE COMMIT (2PC)

Objetivo
Garantir que operações críticas sejam executadas de forma consistente em todos os Super Peers.

Operações protegidas:

- inserção de documento;
- atualização de metadados;
- remoção lógica;
- criação de snapshots.

Participantes
- Coordenador.
- Participantes (Super Peers).

Estados
IDLE → PREPARING → READY → COMMIT → DONE ou READY → ABORT

Fase 1 – Prepare
O coordenador envia:

```
PREPARE(TransactionID)
```

Cada participante:

- valida operação;
- grava em log;

- responde:

```
YES
```

ou

```
NO
```

Fase 2 – Commit

Se todos responderem **YES**:

COMMIT(TransactionID)

Caso contrário:

ABORT(TransactionID)

Recuperação

Cada participante registra:

- PREPARE;
- COMMIT;
- ABORT.

Após falha, o estado é reconstruído a partir do log.

Timeout

Se o coordenador falhar durante PREPARE:

Timeout
  │
Election (Bully)
  │
Novo Coordenador
  │
Consulta Logs
  │
Decisão Final

Fluxo Completo

Cliente
  |
STORE
  |
Coordenador
  |
PREPARE
  |
Participantes
  |
YES / NO
  |
COMMIT ou ABORT
  |
Atualização SMR

BULLY ALGORITHM E RECUPERAÇÃO DO COORDENADOR

Objetivo

Eleger automaticamente um novo coordenador quando o atual falhar.

Identificação

Cada Super Peer possui prioridade baseada em:

Priority = NodeID

O maior NodeID vence a eleição.

Estados

FOLLOWER
  |
  ▼
CANDIDATE
  |
  ▼
COORDINATOR

Processo de Eleição

Timeout
  |
Election
  |
Mensagens para nós superiores
  |
Resposta?

Se houver resposta:

Aguardar Coordenador

Caso contrário:

Assume Coordenação

Mensagens

Mensagem	Função
ELECTION	Inicia eleição
OK	Existe nó superior
COORDINATOR	Novo líder
HEARTBEAT	Monitoramento

Heartbeats

Configuração sugerida:

Heartbeat = 5 s
Timeout = 15 s

Recuperação do Coordenador Antigo

Quando um antigo coordenador retorna:

1. entra como FOLLOWER;
2. sincroniza via IST;
3. recebe estado atualizado;
4. só participa de nova eleição se ocorrer outra falha.

Isso evita interrupções desnecessárias.

Integração com Gossip

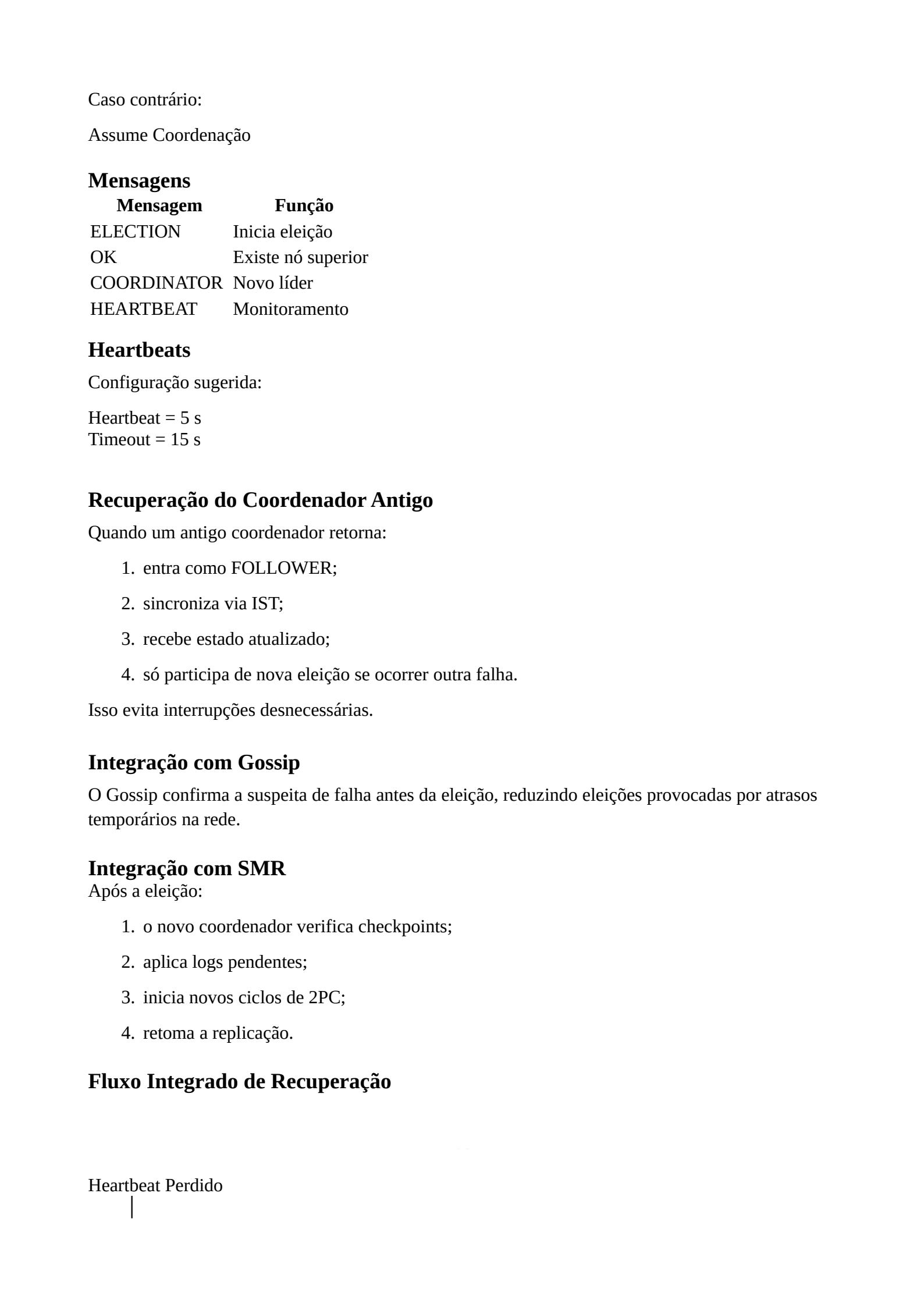
O Gossip confirma a suspeita de falha antes da eleição, reduzindo eleições provocadas por atrasos temporários na rede.

Integração com SMR

Após a eleição:

1. o novo coordenador verifica checkpoints;
2. aplica logs pendentes;
3. inicia novos ciclos de 2PC;
4. retoma a replicação.

Fluxo Integrado de Recuperação



Timeout
|
Gossip Confirma Falha
|
Bully Election
|
Novo Coordenador
|
Carrega Checkpoint
|
Aplica Logs
|
Incremental State Transfer
|
Sistema Operacional

==========

==========

INSTRUÇÕES E METODOLOGIA PARA O DESENVOLVIMENTO DO TRABALHO

1) O trabalho deverá ser feito em dupla;

2) Pode usar IA;

3) O tempo de duração do trabalho é de 8 semanas.

4) A cada checkpoint uma parte do trabalho deverá ser apresentada. As datas de checkpoint são: 18/09, 30/09, 09/10, 21/10, 30/10 e 06/11.

5) Como são dois alunos, a divisão do trabalho será de acordo com a tabela abaixo para que não sobrecarregue um aluno apenas. E, desta forma, no dia do checkpoint, apenas o aluno responsável pela sua parte deverá apresentar.

Área	Aluno 1	Aluno 2
Protocolo de mensagens	**Responsável**	Apoio
TCP/Sockets	**Responsável**	Apoio
Cliente P2P	**Responsável**	—
Upload/download	**Responsável**	Apoio
Fragmentação	**Responsável**	—
SHA-256	**Responsável**	—
LZ4	**Responsável**	—
Armazenamento de arquivos	**Responsável**	—
LFU Cache	**Responsável**	—
Metadata	Apoio	**Responsável**
Hash table	Apoio	**Responsável**
Super Peer	Apoio	**Responsável**

Área	Aluno 1	Aluno 2
Chord	—	**Responsável**
Finger Table	—	**Responsável**
Gossip	—	**Responsável**
Heartbeat	—	**Responsável**
Detecção de falhas	—	**Responsável**
Bully	—	**Responsável**
SMR	—	**Responsável**
2PC	—	**Responsável**
IST	—	**Responsável**
Integração	**50%**	**50%**
Testes	**50%**	**50%**
Documentação	**50%**	**50%**

6) **CHECKPOINT 1**

Deverá estar implementado:

- processo distribuído;
- socket TCP;
- cliente/servidor;
- serialização;
- framing de mensagens;
- concorrência básica;
- identificação de nós.

Aluno 1 implementa:

network.c

protocol.c

peer.c

Esses três códigos deverão realizar:

- criação do socket;
- bind;
- listen;
- accept;
- connect;
- send;
- recv;
- framing;
- header;
- checksum.

Aluno 2 implementa:

node.c

superpeer.c

Esses dois códigos deverão realizar:

- NodeID;
- configuração;
- identificação do processo;
- registro do nó;
- tabela básica de membros;
- criação do Super Peer.

Itens de verificação:

- conexão TCP funcionando;
- mensagem chega corretamente;
- header é interpretado;
- checksum é validado;
- dois processos podem conversar;
- não pode ocorrer segmentation fault.

7) **CHECKPOINT 2**

Aluno 1 será responsável por:

- upload
- download
- storage
- chunking
- SHA-256
- LZ4
- parallel transfer

e pipeline:

arquivo
↓
SHA-256
↓
fragmentação
↓
LZ4
↓
checksum
↓
transferência

Aluno 2 é responsável por:

- metadata
- hash table

- ObjectID
- registro dos chunks

Estrutura em C do metadado:

```
typedef struct {
    uint8_t object_id[32];

    char filename[256];

    uint64_t size;

    uint32_t chunk_count;

    uint8_t **chunk_hashes;

    uint32_t version;

    uint32_t owner;

} FileMetadata;
```

Itens de verificação:
./peer upload arquivo.pdf
O sistema deverá produzir:
File: arquivo.pdf
Size: 12458291 bytes

ObjectID:
a7f3...

Chunks: 3

Chunk 0: ...
Chunk 1: ...
Chunk 2: ...

Compression: LZ4

Depois:
./peer download arquivo.pdf
O sistema deverá produzir:
Download completed
SHA-256 verified

8) **CHECKPOINT 3**
Aluno 1:

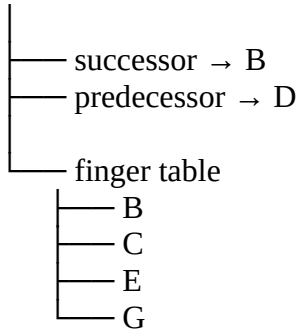
membership
heartbeat
suspect
failed
removed

Estados:
ALIVE
↓
SUSPECT
↓
FAILED
↓
REMOVED

Heartbeat: 5 segundos
Timeout: 15 segundos

Aluno 2:
NodeID
successor
predecessor
finger table
lookup
join
stabilize
notify
fix_fingers

Exemplo:
Node A



Itens de verificação:
Inicialização:
SP1
SP2
SP3
SP4
SP5

Posteriormente:
JOIN SP2
JOIN SP3

JOIN SP4
JOIN SP5

Teste:
lookup <ObjectID>
Saída:
Lookup path:

SP1
↓
SP3
↓
SP5

Owner: SP5

Teste de falha:
kill -9 SP3
Saída:
SP3 SUSPECT
SP3 FAILED

9) **CHECKPOINT 4**

Aluno 1:
ELECTION
OK
COORDINATOR

Fluxo:
Coordinator morreu
↓
timeout
↓
ELECTION
↓
nós de maior prioridade
↓
OK
↓
novo coordenador
↓
COORDINATOR

Aluno 2:
operation log
log index
transaction ID
version
checksum
Exemplo:
LOG #101

PUT fileA

LOG #102
PUT fileB

LOG #103
DELETE fileA

Itens de avaliação:
Início:
SP1
SP2
SP3
SP4

SP4 é coordenador.

Executa: kill -9 SP4

Sistema deve retornar:
detect failure
↓
start election
↓
highest NodeID wins
↓
COORDINATOR
↓
resume operations

Próxima execução:

upload arquivo.pdf

E compara:
metadata SP1
metadata SP2
metadata SP3

Todos precisam estar logicamente consistentes.

10) **CHECKPOINT 5**

Aluno 1: implementa LFU + Transferência.
LFU
cache hit
cache miss
frequency
eviction

Aluno 2: implementa 2PC + IST
PREPARE

↓
YES / NO
↓
COMMIT / ABORT

Funcionamento do IST:
Novo nó: version 1032
Coordenador: 1050
Transferir: 1033 … 1050 em vez de toda a base.
Isso corresponde diretamente ao mecanismo de Incremental State Transfer descrito no texto.

Itens de avaliação:
Início:
SP1
SP2
SP3

Retorna:
UPLOAD A
UPLOAD B
UPLOAD C

Vamos derrubar: SP2
Enquanto SP2 está fora:
UPLOAD D
UPLOAD E

Reinicia SP2

Após reinício, tem-se:
SP2
↓
JOIN
↓
IST
↓
versions missing
↓
receive logs
↓
READY

Teste de 2PC:
Vamos forçar uma falhar durante a fase PREPARE.
O sistema deve evitar deixar os participantes em estados inconsistentes.

10) **CHECKPOINT 6 – Integração e teste final.**
Aluno 1 deve demonstrar:

upload
download
chunks
compression
cache
replica

Aluno 2 deve demonstrar:
Chord
Gossip
Bully
SMR
2PC
IST

Por fim, ambos podem demonstrar juntos:
Integração, Debugging, Documentação e Testes.

Nessa fase eu vou fornecer um conjunto de testes para testar o sistema.

OBSERVAÇÃO IMPORTANTE: SÓ SE AVANÇA PARA O CHECKPONT SEGUINTE E O ATUAL ESTIVER FUNCIONAL. CASO CONTRÁRIO, NA PRÓXIMA DATA DO CHECKPOINT, O GRUPO DEVERÁ APRESENTAR O CHECKPOINT ANTERIOR.

OBSERVAÇÃO 2: AO FINAL DE CADA CHECKPOINT, O GRUPO DEVERÁ FORNECER UM DOCUMENTO COM A EXPLICAÇÃO DO DESENVOLVIMENTO DAQUELE CHECKPOINT.

Gradação do trabalho:
Checkpoint 1: 1,0 ponto;
Checkpoint 1: 1,0 pontos;
Checkpoint 1: 2,0 pontos;
Checkpoint 1: 2,0 pontos;
Checkpoint 1: 2,0 pontos;
Checkpoint 1: 2,0 pontos;