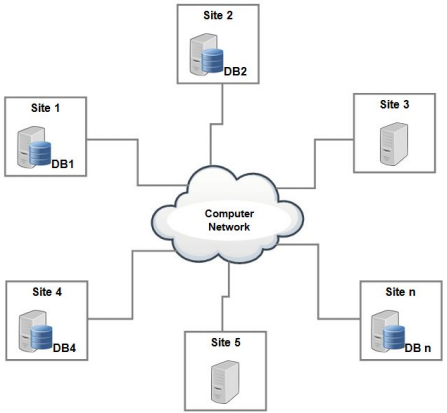
CSC-27: Processamento Distribuído

Banco de Dados Distribuído com *Log-Based Recovery*

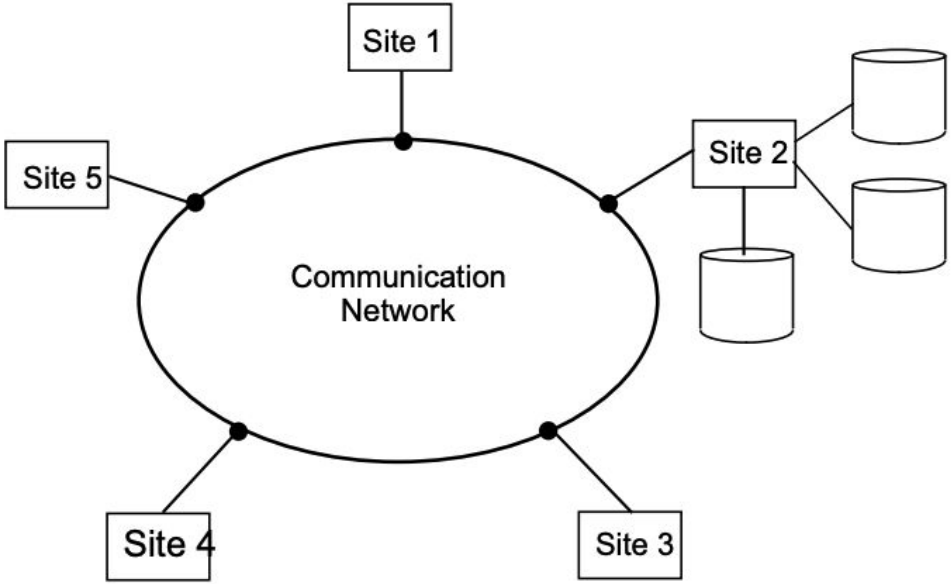
Caio Bianchi Nocrato Gomes
Frederick Del Gaudio Campbell
Victor Hugo Mendes Zenteno **Zuleta**
Zander do Valle Bernardino



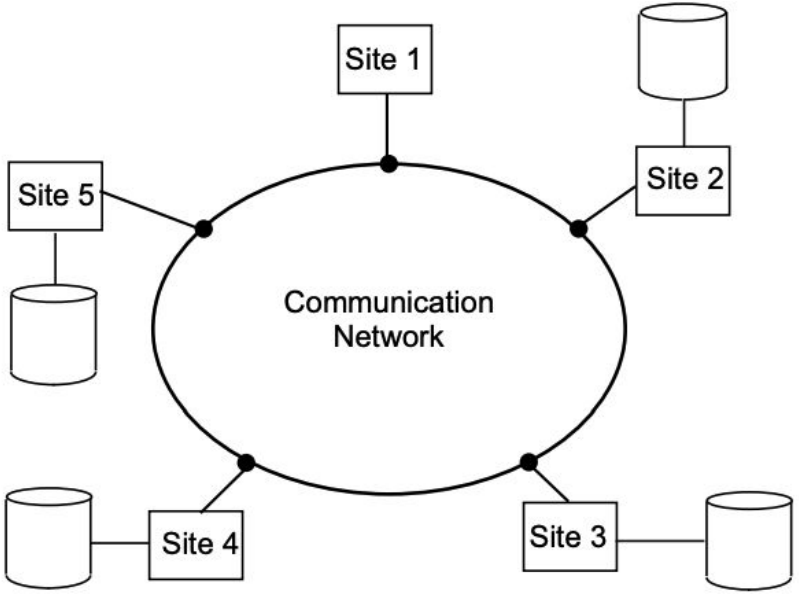
Motivação

- **Cenário**:
  - Aplicações modernas (bancos digitais, e-commerce, sensores IoT) geram e acessam grandes volumes de dados de forma distribuída.
  - A disponibilidade e integridade dos dados são críticas – não se pode perder informações em caso de falhas.
- **Problema**: Como garantir consistência e recuperação confiável em um banco de dados distribuído quando ocorrem falhas (de nós, rede, energia, etc.)?
- **Desafio em Sistemas Distribuídos**:
  - Sincronizar transações entre múltiplos nós sem comprometer o desempenho.
  - Detectar e recuperar falhas mantendo as propriedades ACID (Atomicidade, Consistência, Isolamento e Durabilidade).

Motivação: DB Centralizado *vs* DB Distribuído

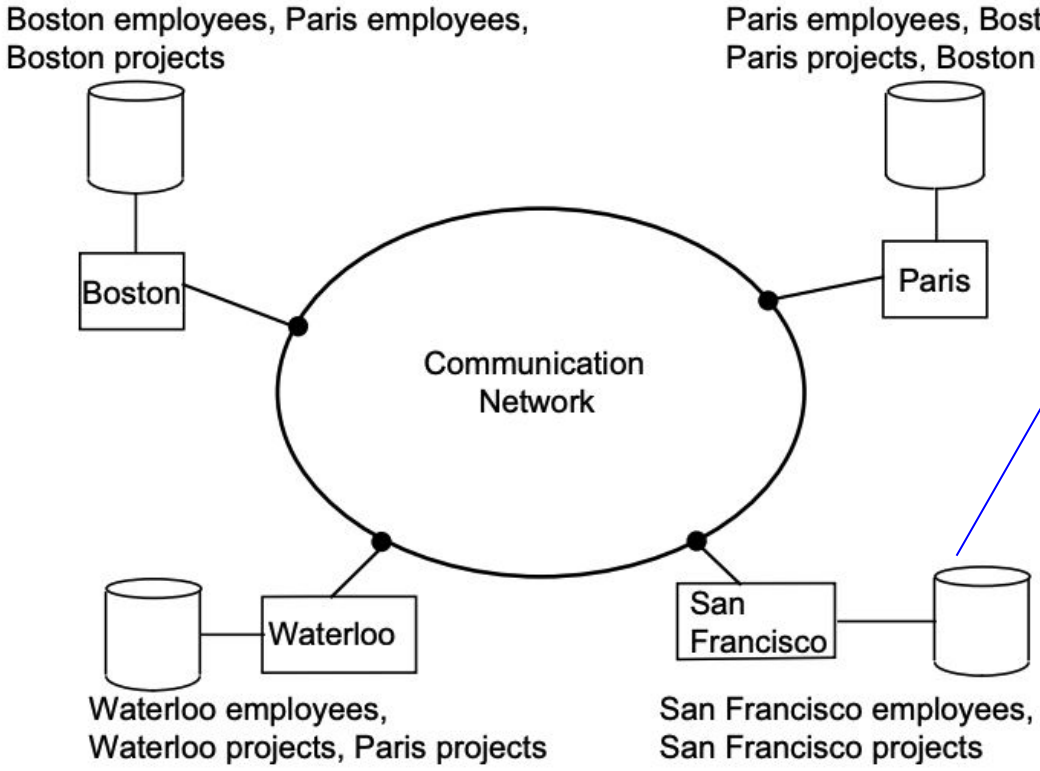


DB Centralizado



DB Distribuído

Motivação: Exemplo



Fragmentação

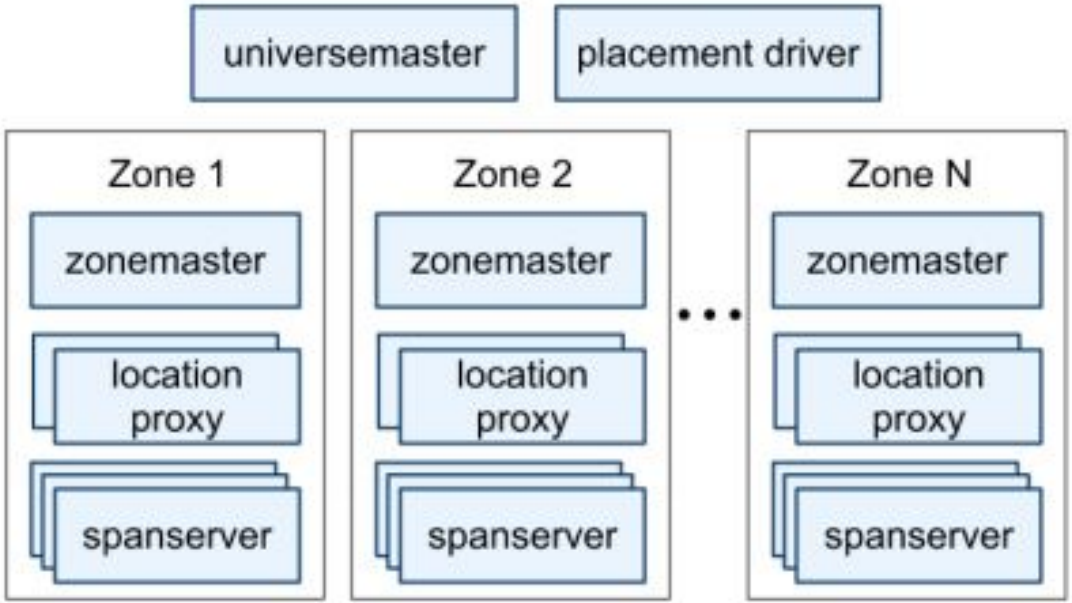
Begin transaction
SALARY_UPDATE
begin
Each Salary S
Updates to 1.1*S
end.

- Como garantir a atomicidade e a consistência?
- E se um nó falhar e os outros não?

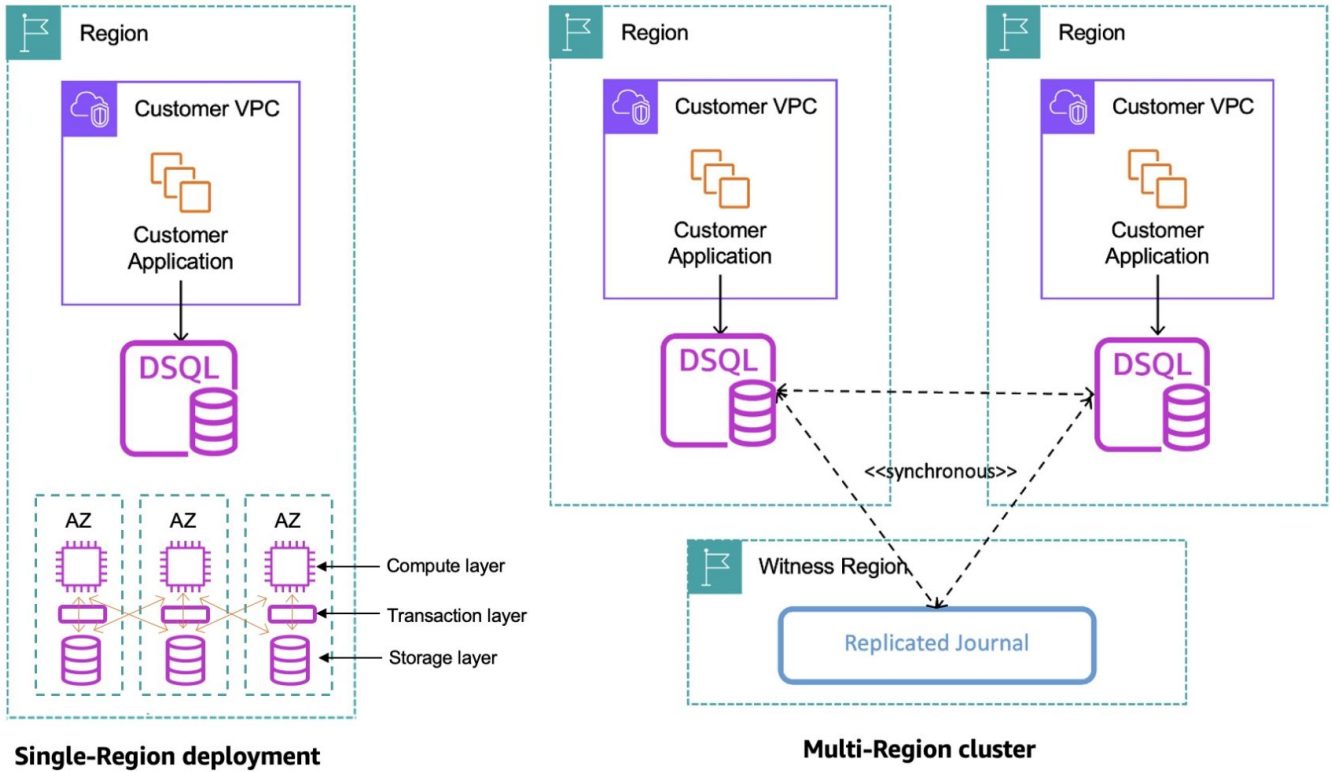
Related Work

- Google Spanner: banco de dados distribuído com consistência global e relógios atômicos.
- Amazon Aurora: usa logs para replicação e recuperação rápida.
- Embora esses sistemas sejam robustos, eles são altamente complexos.
- Nosso projeto busca uma implementação didática e simplificada desses princípios, mostrando o funcionamento interno de um banco distribuído com log e commit distribuído de forma prática e acessível.

Related Work: Google Spanner



Related Work: Amazon Aurora

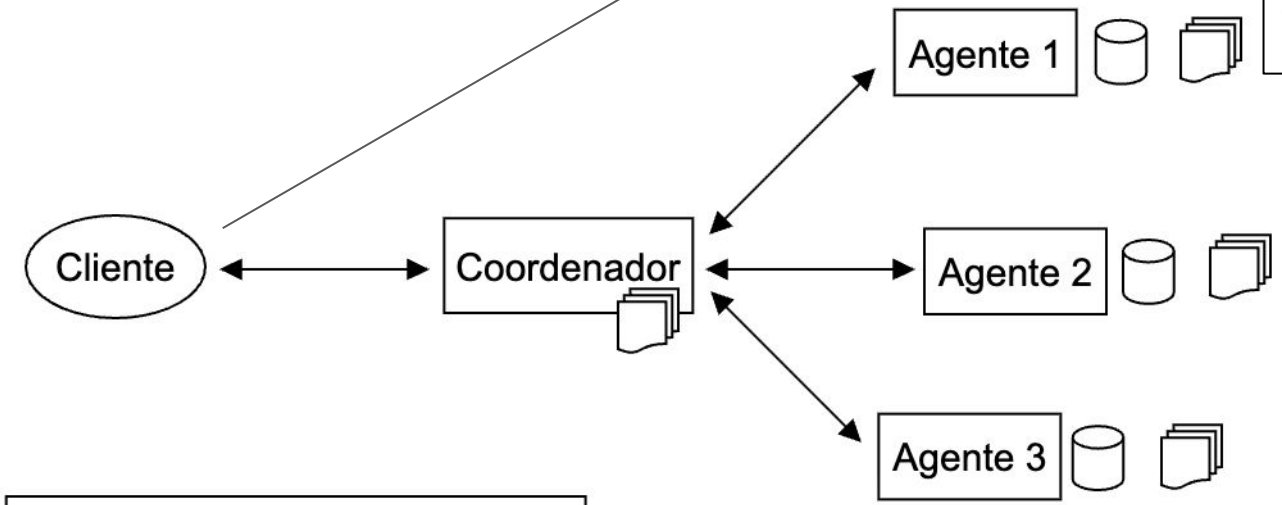


Objetivo

- Desenvolver um **sistema de banco de dados distribuído** com **log-based recovery**, capaz de executar e recuperar transações após falhas de nós ou rede.
- Objetivos Específicos:
  - Implementar **coordenador** e **agentes** com comunicação distribuída.
  - Empregar **Write-Ahead Logging (WAL)** com checkpoints para permitir undo/redo.
  - Suportar transações de leitura e escrita com **commit distribuído** (2PC).
  - **Simular falhas** controladas e demonstrar recuperação automática.
  - Fornecer **interface simples** para o cliente enviar consultas (ex.: saldo, histórico, etc.).

Objetivo – Fluxograma

A complexidade do sistema é mascarada para o cliente → Transparência

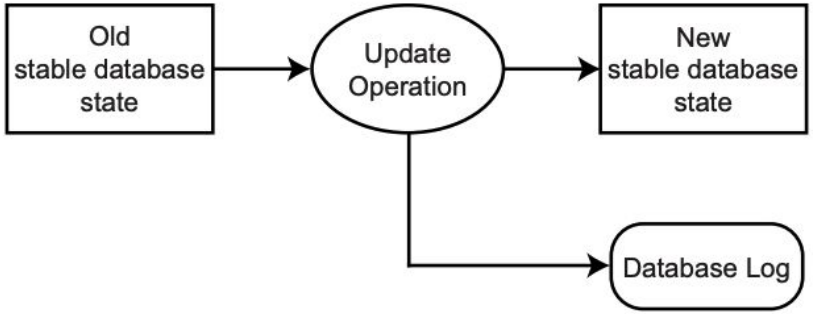


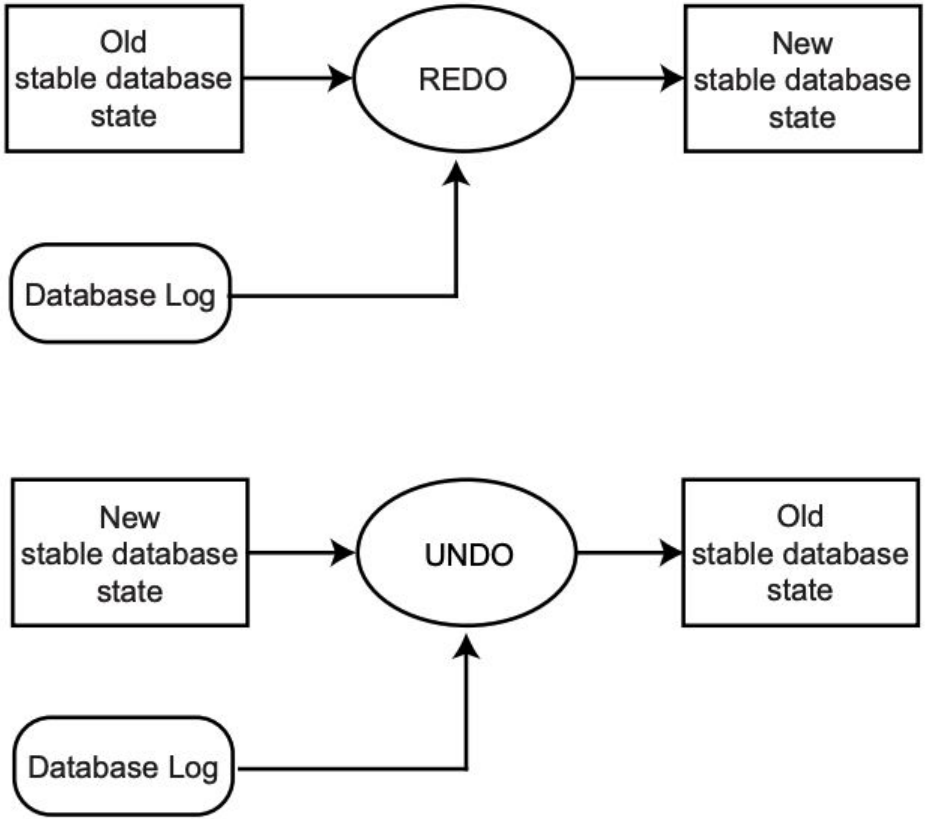
Legenda:

Partição Local do BD

Arquivo de Log

Objetivo – Logging





Objetivo – *Two-Phase Commitment (2PC) Protocol*

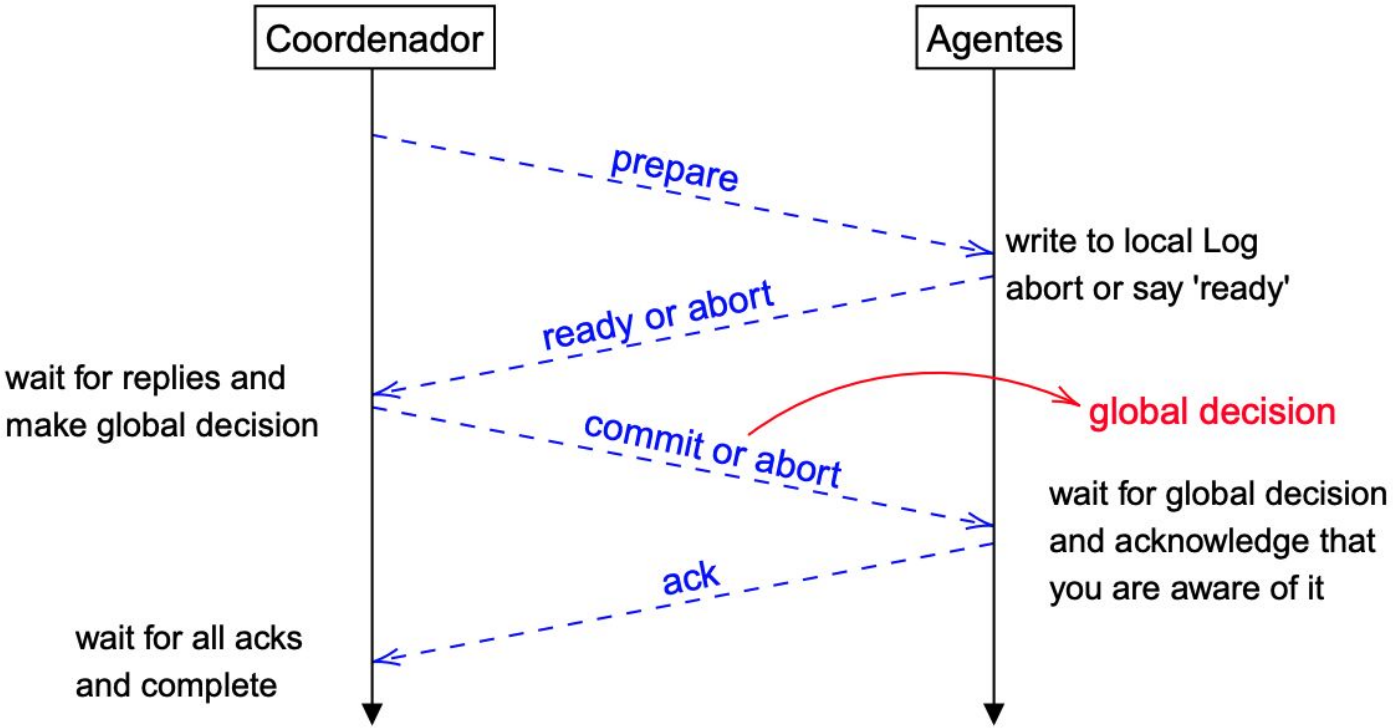


Diagrama Simplificado do 2PC

Objetivo – *Two-Phase Commitment (2PC) Protocol*

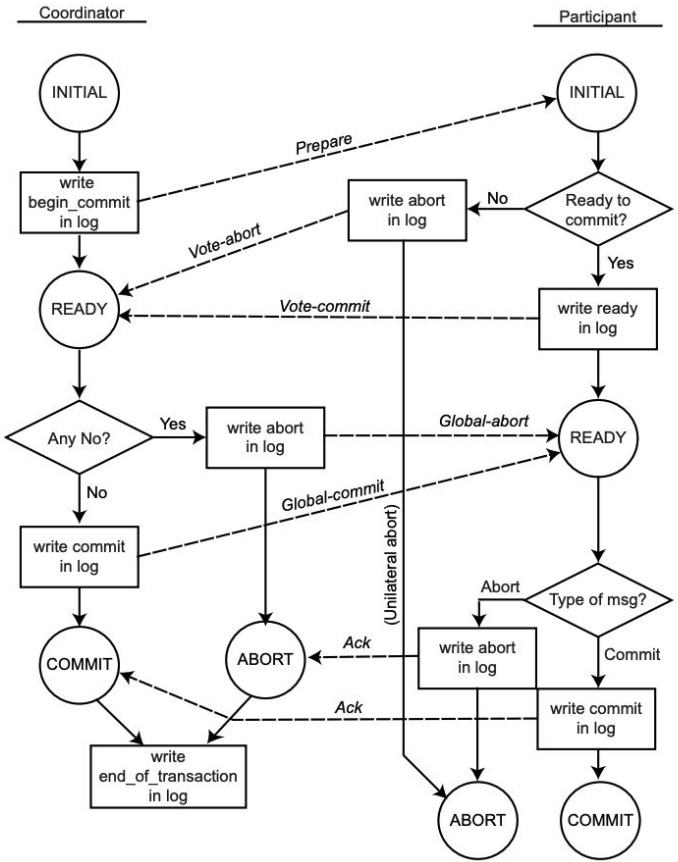


Diagrama Completo do 2PC

Objetivo – Expansão do Projeto

- Implementação do 3PC Protocol para evitar bloqueio dos agentes.
- Implementação de Algoritmo de Eleição de Líder para caso o coordenador caia.

Proposta

- **Requisitos Funcionais**:
  - Executar transações distribuídas (read/write).
  - Garantir atomicidade entre múltiplos nós.
  - Registrar logs de operações (write-ahead logging).
  - Recuperar estado após falhas (usando undo/redo).
  - Permitir consultas consistentes (read queries).
- **Requisitos Não Funcionais**:
  - Implementação em Go (concorrência e RPC nativos).
  - Contêinerização via Docker (execução e replicação facilitadas).
  - Logs armazenados em arquivos locais.
  - Suporte à escalabilidade (adição de nós).