Controle de Concorrência e Transações em Sistemas Distribuídos

Bruno Henrique Vono de Sant'Ana Nildemar de Oliveira Garcia Neto
Victor Guilherme de Oliveira Barreto

December 5, 2025

Abstract

Este artigo explora o controle de concorrência e transações, desde os conceitos fundamentais em sistemas operacionais até os desafios em sistemas distribuídos. Analisamos mecanismos de sincronização clássicos, o Teorema CAP e suas implicações em arquiteturas SQL e NoSQL, além de modelos descentralizados como P2P e Blockchain.

1 Introdução

O controle de concorrência é uma área fundamental da computação que trata de como múltiplas tarefas, processos ou threads podem executar simultaneamente compartilhando recursos sem causar inconsistências, conflitos ou falhas de execução.

Em sistemas modernos, especialmente aqueles que exigem alta performance, escalabilidade e paralelismo, compreender mecanismos de sincronização é essencial. Em sistemas distribuídos, onde não há um relógio global e os componentes falham independentemente, o controle de transações torna-se crucial para garantir a coerência dos dados.

2 Conceitos Fundamentais

2.1 Concorrência, Processos e Threads

Concorrência é a capacidade de um sistema realizar múltiplas tarefas de forma sobreposta no tempo, podendo ocorrer em um único processador (*time slicing*) ou em múltiplos processadores (paralelismo real).

- **Processo:** Uma instância de um programa em execução, com memória isolada.
- **Thread:** Unidade de execução dentro de um processo, compartilhando memória e recursos.

2.2 Sistemas Distribuídos e Transações

Um Sistema Distribuído é uma rede que armazena dados em múltiplos nós simultaneamente. Uma **transação** é uma sequência de operações que deve ser atômica e isolada. O controle de concorrência nesses sistemas visa garantir a **equivalência serial**, assegurando que o resultado da execução concorrente seja idêntico ao de uma execução sequencial.

3 Problemas de Concorrência e Controle de Recursos

Recursos compartilhados (como arquivos, bancos de dados e memória) exigem controle rigoroso para evitar condições de corrida. Os problemas clássicos incluem:

- **Deadlock (Impasse):** Dois processos esperam eternamente por recursos um do outro. Em sistemas distribuídos, o grafo de espera pode envolver servidores distintos.
- **Livelock:** Processos continuam ativos, mudando de estado, mas sem progredir.
- **Starvation:** Um processo nunca recebe acesso ao recurso devido a prioridades mal gerenciadas.

4 Mecanismos de Sincronização

4.1 Primitivas Locais

- **Mutex:** Garante que apenas uma thread acesse um recurso por vez.
- **Semáforos:** Controlam acesso a recursos contáveis (binários ou de contagem).
- **Monitores:** Estruturas de alto nível que encapsulam recursos com exclusão mútua automática.

4.2 Coordenação Distribuída

- **Coordenador de Transações:** Gerencia o início e o fim (commit/abort) das transações, alocando identificadores únicos (TIDs).
- **Protocolo de Confirmação de Duas Fases (2PC):** Garante a atomicidade em transações distribuídas, assegurando que todos os nós participantes concordem com o resultado.
- **Ordenação por Carimbo de Tempo:** Serializa acessos baseando-se no tempo de início da transação, resolvendo conflitos com regras como "a última escrita vence".

5 Teorema CAP e Arquiteturas de Dados

O **Teorema CAP** afirma que um sistema distribuído só pode garantir simultaneamente duas das três propriedades: Consistência (C), Disponibilidade (A) e Tolerância a Partições (P). Como a tolerância a partições (P) é inevitável em redes, a escolha recai entre CP e AP:

5.1 SQL (Escolha CP)

Bancos de dados relacionais priorizam a consistência estrita (ACID). Em caso de falha de rede, sacrificam a disponibilidade para evitar dados inconsistentes.

5.2 NoSQL (Escolha AP)

Sistemas como Cassandra e Dynamo priorizam a disponibilidade e escalabilidade. Utilizam consistência eventual e mecanismos otimistas de controle de concorrência, não garantindo isolamento ACID completo.

6 Concorrência em Redes Descentralizadas

6.1 BitTorrent (P2P de Conteúdo)

Adota o modelo **AP**. Foca na integridade da transferência (via hashes) para reconstrução de arquivos, permitindo que downloads continuem a partir de diferentes nós mesmo com falhas na rede.

6.2 Blockchain (P2P de Registros)

Adota o modelo **CP**. A consistência é garantida por **consenso distribuído** (como Proof of Work ou Paxos). A imutabilidade é assegurada pelo encadeamento de blocos via hash, prevenindo o problema do gasto duplo.

7 Controle em Linguagens Modernas

- **Java:** Suporte via `synchronized`, `ReentrantLock` e `Executors`.
- **Python:** Utiliza `threading.Lock` e `asyncio` para concorrência assíncrona.
- **Go:** Baseia-se em Goroutines e Channels, com o princípio de "compartilhar memória através da comunicação".

8 Conclusão

O controle de concorrência evoluiu de primitivas locais para complexos protocolos distribuídos. A escolha da arquitetura depende do domínio: sistemas financeiros exigem consistência forte (CP), enquanto aplicações de conteúdo massivo priorizam disponibilidade (AP). O domínio dessas técnicas é estratégico para o desenvolvimento de sistemas robustos e escaláveis.

References

[1] TANENBAUM, Andrew S. *Modern Operating Systems*. Upper Saddle River: Prentice Hall, 2015.

[2] SILBERSCHATZ, Abraham; GALVIN, Peter; GAGNE, Greg. *Operating System Concepts*. Hoboken: Wiley, 2018.

[3] GOETZ, Brian. *Java Concurrency in Practice*. Upper Saddle River: Addison-Wesley, 2006.

[4] DIJKSTRA, Edsger W. *Cooperating Sequential Processes*. New York: Academic Press, 1968.

[5] COULOURIS, George; DOLLIMORE, Jean; KINDBERG, Tim; BLAIR, Gordon. *Sistemas Distribuídos: Conceitos e Projeto*. 5. ed. Porto Alegre: Bookman, 2013.

[6] IBM. *O que é o Teorema CAP?*. Disponível em: ibm.com. Acesso em: 2024.

[7] GTA/UFRJ. *Centralização x Descentralização - Redes P2P*. Grupo de Teleinformática e Automação, UFRJ.

[8] IC/UFF. *Sistemas de redes peer to peer*. Instituto de Computação, Universidade Federal Fluminense.

[9] AUTOR DESCONHECIDO. *Blockchain Aplicada a Redes de Computadores*. (Baseado no material fornecido).