



Centro Universitário Católica de Quixadá

CENTRO UNIVERSITÁRIO CATÓLICA DE QUIXADÁ

CURSO DE SISTEMAS DE INFORMAÇÃO

BRUNO DA SILVA DE SOUSA

**RELATÓRIO DE ATIVIDADES PRÁTICAS E TEÓRICAS: RELÓGIOS LÓGICOS E
CONTROLE DE CONCORRÊNCIA**

QUIXADÁ

2025

INTRODUÇÃO

Este relatório apresenta as soluções desenvolvidas para as atividades propostas sobre sincronização e tolerância a falhas em sistemas distribuídos. O documento está dividido em três partes: respostas às questões teóricas baseadas no material didático (Aulas 9 a 15), a descrição e resultados da implementação prática do Algoritmo de Lamport para ordenação de eventos lógicos, e um estudo de caso sobre o controle de concorrência no Google Docs.

1. (Aula 9) Diferença entre Abordagem Reativa e Proativa

A abordagem reativa foca no que fazer após a ocorrência de uma falha. Suas técnicas, como "Tentar novamente", "Replicação", "Checkpointing" e "Log das mensagens", visam recuperar o sistema ou mitigar o impacto de um erro já manifestado.

Já a abordagem proativa foca na prevenção, ou seja, em como reduzir a probabilidade de que falhas ocorram antes que elas afetem o sistema. Técnicas como "Migração preemptiva", "Auto Cura" e "Rejuvenescimento do software" buscam antecipar problemas através de monitoramento constante.

Por que a proativa pura é inviável: Em sistemas distribuídos complexos, uma abordagem puramente proativa é difícil de implementar isoladamente devido à complexidade inerente ao ambiente distribuído. Determinar, por exemplo, o momento exato para migrar um serviço ou decidir que um servidor parou de responder adiciona uma "enorme complexidade em projetos de sistemas distribuídos". É praticamente impossível prever todas as falhas possíveis em ambientes heterogêneos e dinâmicos, tornando necessário o uso combinado com técnicas reativas (como replicação) para garantir a disponibilidade e confiabilidade.

2. (Aula 9 e 10) Falhas Bizantinas vs. Mascaramento

- Falhas Bizantinas (Arbitrárias): Descritas na Aula 9, ocorrem quando um servidor produz respostas erradas ou arbitrárias em momentos imprevisíveis. O servidor pode, inclusive, "trabalhar maliciosamente em conjunto com outros servidores para produzir respostas erradas intencionalmente". É uma falha interna de comportamento do componente.
- Mascaramento: Descrito na Aula 10, é uma ameaça de segurança onde um invasor ou processo "envia ou recebe mensagens usando a identidade de outro, sem sua autorização".

Relação: Os dois conceitos estão intrinsecamente ligados à confiabilidade e segurança. Uma falha Bizantina é o "pior cenário" de falha, onde o sistema não apenas para, mas mente. Um ataque de Mascaramento é, muitas vezes, a manifestação externa ou a causa de um comportamento bizantino: se um invasor consegue mascarar sua identidade (ataque de segurança), ele pode injetar dados falsos no sistema, fazendo com que o sistema exiba um comportamento arbitrário (falha bizantina) aos olhos dos outros processos, comprometendo a integridade e a autenticidade das informações.

3. (Aula 11) Relógios Lógicos de Lamport

O propósito fundamental dos relógios lógicos de Lamport é fornecer uma ordenação de eventos baseada na premissa de "acontece antes" e na causalidade, independentemente da hora real (física).

Por que é mais crítica que o tempo real: Em sistemas distribuídos, "medir o tempo é problemático", pois cada computador possui seu próprio relógio físico (baseado em cristal de quartzo) e esses relógios divergem entre si (taxas de deriva), sendo muito difícil mantê-los perfeitamente sincronizados. Para garantir a consistência de dados (como a ordem de mensagens em um chat ou transações bancárias), é irrelevante que todos concordem com a "hora exata" (ex: 14:00:01), mas é crucial que concordem com a ordem em que os eventos ocorreram (ex: Evento A causou Evento B). O algoritmo de Lamport resolve isso ajustando contadores lógicos ao enviar/receber mensagens, garantindo que se A envia para B, B terá um timestamp maior que A.

4. (Aula 12) Algoritmo do Valentão (Bully Algorithm)

O Algoritmo do Valentão assume que o processo com o maior número de identificação (ID) ativo deve ser o coordenador. Quando um processo nota que o coordenador não responde, ele inicia uma eleição.

Cenário: Sistema com 8 processos (IDs 0 a 7, assumindo numeração padrão do texto ou similar). Coordenador anterior (digamos, 7) falhou. P1 (ID 5) e P2 (ID 6) detectam a falha simultaneamente.

Passo a Passo (Baseado na Figura 1 e Texto):

1. Início Simultâneo: P1 (5) e P2 (6) iniciam eleições independentes ao notarem que o coordenador caiu.
2. Envio de Mensagens (P1): P1 (5) envia mensagens de "Eleição" para todos os processos com ID maior que o dele: IDs 6 e 7.

3. Envio de Mensagens (P2): P2 (6) envia mensagens de "Eleição" para todos com ID maior: ID 7.
4. Respostas:
 - Como o ID 7 (antigo coordenador) está falho, ele não responde a ninguém.
 - O processo P2 (6) recebe a mensagem de P1 (5). Como 6 é maior que 5, P2 (6) responde "OK" para P1 (5), indicando que ele (P2) assumirá a eleição.
5. P1 Desiste: Ao receber o "OK" de P2, P1 (5) encerra sua tentativa e aguarda o vencedor.
6. P2 Vence: P2 (6) não recebe resposta do ID 7 (inativo). Logo, P2 (6) "vence a eleição".
7. Anúncio: P2 (6) envia uma mensagem de "Coordenador" para todos os outros processos ativos, informando que ele é o novo líder.

5. (Aula 13 e 14) Consenso e Transações Distribuídas

Consenso: É o problema onde processos precisam entrar em acordo sobre um determinado valor ou ação, após um ou mais processos terem proposto qual deve ser esse valor. É fundamental para manter a consistência de logs e variáveis compartilhadas.

Relação com Atomicidade: Uma Transação Distribuída possui a propriedade de Atomicidade (parte do ACID), que exige que "tudo é feito, ou nada é feito". Como a transação envolve múltiplos servidores, todos devem concordar se a transação será confirmada (commit) ou cancelada (abort). Para garantir essa atomicidade, utiliza-se um protocolo de consenso, como o protocolo de confirmação de duas fases (2PC). Neste protocolo, todos os participantes votam e devem chegar a uma "decisão conjunta" (consenso) na segunda fase. Se qualquer participante votar pelo cancelamento, o consenso deve ser o cancelamento global da transação.

6. (Aula 15) Travamento de Duas Fases vs. Controle Otimista

- Travamento de Duas Fases (2PL): É uma abordagem pessimista. Bloqueia (trava) o objeto assim que uma transação quer acessá-lo (leitura ou escrita), fazendo outras transações esperarem até que a primeira termine (confirme ou cancele). É vantajoso quando há muitos conflitos (muitas escritas).
- Controle de Concorrência Otimista: Supõe que conflitos são raros ("melhor situação ocorrerá com maior frequência"). Permite que as operações ocorram livremente em

versões de tentativa válida apenas no final (fases de trabalho, validação e atualização). Se houver conflito na validação, a transação é cancelada.

Por que Google Apps e Wikipédia usam Otimista: Sistemas colaborativos exigem alta responsividade. O travamento (pessimista) impediria que múltiplos usuários editassem ao mesmo tempo, pois travaria o documento ou parágrafo, causando lentidão. A abordagem otimista permite que todos editem suas cópias locais simultaneamente. O Google Apps e a Wikipédia aceitam o risco de conflito e, quando ocorrem, "delegam aos usuários a resolução dos conflitos" (ex: mostrando "editar conflito" na Wiki ou atualizações em tempo real no Docs), garantindo uma experiência de usuário fluida em vez de bloqueada.

7. (Aula 14) Impasse Fantasma (Phantom Deadlock)

Um impasse fantasma é um "falso positivo" na detecção de deadlocks em sistemas distribuídos. Ele ocorre quando um algoritmo detecta um ciclo de dependência (impasse) que, na realidade, não existe mais.

Como ocorre: Em sistemas distribuídos, não existe um relógio global perfeito e a comunicação tem latência. Para detectar impasses, os servidores trocam informações de seus grafos locais ("espera por") para construir um grafo global. Devido à demora em reunir e transmitir essas informações de diferentes servidores, o estado do sistema pode mudar durante o processo. Por exemplo, uma transação que estava segurando uma trava pode tê-la liberado no momento em que a informação estava trafegando para o detector central. O detector, olhando para informações desatualizadas de diferentes partes do sistema, enxerga um ciclo (impasse) que já foi desfeito na realidade.

ESTUDO DE CASO - GOOGLE DOCS

1. INTRODUÇÃO

O Google Docs representa um desafio complexo de sistemas distribuídos, exigindo alta disponibilidade e consistência eventual. Diferente de sistemas bancários rígidos, ele permite que múltiplos usuários editem o mesmo recurso simultaneamente. Para viabilizar isso, o sistema deve abandonar técnicas tradicionais de bloqueio em favor de abordagens mais flexíveis, lidando com desafios de sincronização de relógios físicos discrepantes e falhas parciais inerentes a ambientes distribuídos.

2. A Inadequação do Algoritmo Centralizado de Exclusão Mútua

Algoritmo Centralizado para exclusão mútua, onde um processo é eleito como coordenador para gerenciar o acesso a um recurso crítico. Neste modelo, quando um processo deseja acessar o recurso (como editar o documento), ele deve pedir permissão ao coordenador. Se o recurso estiver ocupado, o coordenador enfileira a requisição e o solicitante deve esperar. Se o Google Docs utilizasse essa abordagem:

- Latência Inaceitável: Cada caractere digitado exigiria uma viagem de ida e volta ao servidor coordenador para obter permissão ("trava"), tornando a edição em tempo real lenta e travada.
- Ponto Único de Falha: Conforme citado na aula, "a possibilidade de falha do coordenador central pode comprometer o funcionamento de todo o sistema". Se o coordenador cair, ninguém edita o documento.
- Gargalo de Desempenho: O coordenador central pode se tornar um gargalo ao tentar processar milhares de requisições de edição simultâneas.

Portanto, o uso de travas (locking) para controlar o acesso em tempo real é inviável, pois "travaría o parágrafo ou o documento inteiro", impedindo a colaboração fluida.

3. A Superioridade da Abordagem Otimista

Para resolver isso, o Google Docs adota o Controle de Concorrência Otimista descrito na Aula 15. Esta abordagem "supõe que a melhor situação ocorrerá com maior frequência", permitindo que as transações (edições) ocorram livremente e agindo apenas quando conflitos são identificados posteriormente. No fluxo do Google Docs:

1. Fase de Trabalho: O usuário edita sua cópia local (versão de tentativa) imediatamente, sem bloqueios. Isso garante a responsividade da interface.
2. Validação e Detecção: As edições são enviadas ao servidor. Como não há um relógio global perfeito, o servidor utiliza algoritmos baseados em Relógios Lógicos ou Vetoriais (Aula 11) para ordenar os eventos causalmente e detectar conflitos. O sistema verifica se duas operações são concorrentes e conflitantes (ex: editar a mesma palavra ao mesmo tempo).
3. Resolução (Delegação): Se houver conflito, em vez de simplesmente cancelar a transação, sistemas como o Google Apps "delegam aos usuários a resolução dos conflitos". O sistema tenta mesclar automaticamente, mas se não for possível, destaca a divergência para que o usuário humano decida a versão final.

4. Tolerância a Falhas e Recuperação

Considerando a Aula 9, sistemas distribuídos sofrem falhas parciais, como a queda de conexão de um usuário. O Google Docs lida com falhas de comunicação e omissão permitindo que o usuário continue editando offline ("falha parcial" onde o componente local funciona mas a rede não). Quando a conexão é restabelecida, o sistema utiliza os logs de operações e timestamps para ressincronizar o estado local com o servidor, garantindo que "um sistema com alta confiabilidade continue a funcionar sem interrupção".

5. Conclusão

A análise demonstra que a colaboração em tempo real exige o abandono do pessimismo do Algoritmo Centralizado (Aula 12) em favor do otimismo da Aula 15. Ao permitir edições locais e resolver conflitos a posteriori (delegando ao usuário quando necessário), o sistema maximiza a disponibilidade e a experiência do usuário, suportado por algoritmos lógicos (Aula 11) que garantem a ordem correta dos eventos mesmo diante de falhas e latência

