INSTITUTO FEDERAL DE EDUCAÇÃO, CIÊNCIA E TECNOLOGIA DA BAHIA
CAMPUS - SANTO ANTÔNIO DE JESUS - BAHIA

CURSO DE TECNOLOGIA EM ANÁLISE E DESENVOLVIMENTO DE SISITEMAS

MARCELO DE JESUS

RONALDO CORREIA

Sistema Distribuído de Controle Colaborativo com Exclusão Mútua e Recuperação de Falhas

SANTO ANTONIO DE JESUS – BA
2025

MARCELO DE JESUS

RONALDO CORREIA

Sistema Distribuído de Controle Colaborativo com Exclusão Mútua e Recuperação de Falhas

Relatório técnico da atividade de Sistemas Distribuído- Controle Colaborativo com Exclusão Mútua e Recuperação de Falhas, elaborado como requisito parcial de avaliação para a disciplina de Sistemas Distribuído, ministrada pelo Prof. Felipe Silva.

SANTO ANTONIO DE JESUS – BA

2025

1. INTRODUÇÃO

O presente relatório apresenta a implementação de um sistema distribuído de controle colaborativo, com ênfase em exclusão mútua centralizada, replicação do recurso compartilhado e mecanismos robustos de recuperação de falhas. O objetivo principal é assegurar que múltiplos nós possam acessar de forma segura e coordenada um recurso comum, mantendo a integridade dos dados mesmo diante de atrasos de rede ou falhas inesperadas. O sistema simula a interação entre diversos nós clientes e um servidor coordenador, utilizando comunicação via TCP e mensagens serializadas em JSON. Cada nó mantém um relógio lógico de Lamport para garantir a ordenação causal dos eventos, enquanto o servidor central gerencia a entrada na seção crítica, aplicando políticas de prioridade baseadas nos timestamps.

Além disso, o sistema implementa checkpoints periódicos e rollback de estado para assegurar que, em caso de falhas ou operações interrompidas, os nós possam restaurar o estado consistente anterior. A simulação de atrasos e crashes permite avaliar o comportamento do sistema em cenários adversos, fornecendo insights sobre a confiabilidade, eficiência e consistência da replicação.

2. ARQUITETURA DO SISTEMA

2.1 Modelo Cliente Servidor

O Sistema foi desenvolvido utilizando uma arquitetura cliente-servidor centralizada, na qual um coordenador atua como servidor central e os demais processos funcionam como nós clientes. Os processos se comunicam via sockets TCP, enviando mensagens serializadas em JSON usando a biblioteca Gson. Cada nó envia pedidos de entrada na seção crítica (REQUEST), recebe permissões (GRANT) do coordenador, executa operações (DO_OP) e libera o acesso (RELEASE).

A coordenação é centralizada, pois o servidor mantém uma fila de prioridades (PriorityBlockingQueue) baseada nos relógios de Lamport e decide qual nó pode entrar na seção crítica.

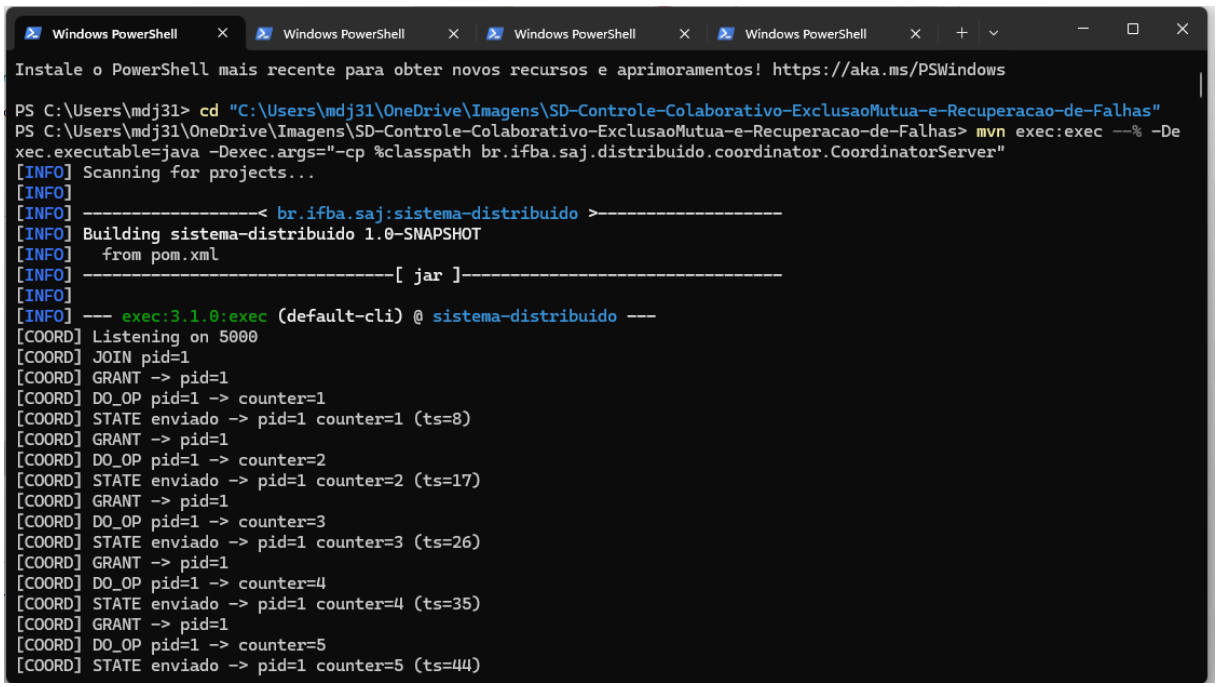
- Coordenador (CoordinatorServer): atua como ponto central de decisão, controlando o acesso à seção crítica e propagando atualizações de estado.
- Nós (NodeClient): mantêm cópia local do recurso, solicitam acesso à seção crítica, aplicam operações e replicam alterações.

2.2 Estrutura de Pacotes

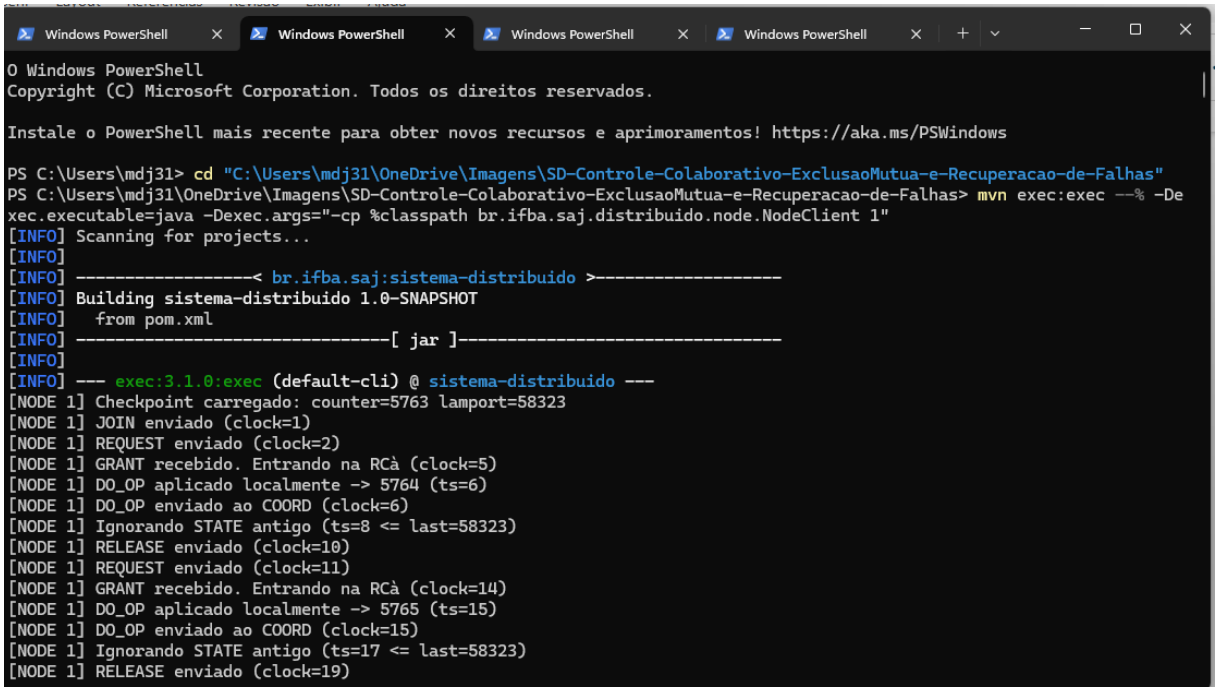
- br.ifba.saj.distribuido.node: NodeClient e NodeState– execução de operações críticas, checkpoint, rollback e simulação de atraso/falha.
- br.ifba.saj.distribuido.coordinator: CoordinatorServer – fila de prioridades, controle de acesso e propagação de STATE.
- br.ifba.saj.distribuido.model: Message, MessageType e LamportClock – padronização de comunicação e ordenação de eventos.

2.3 Logs do Sistema:

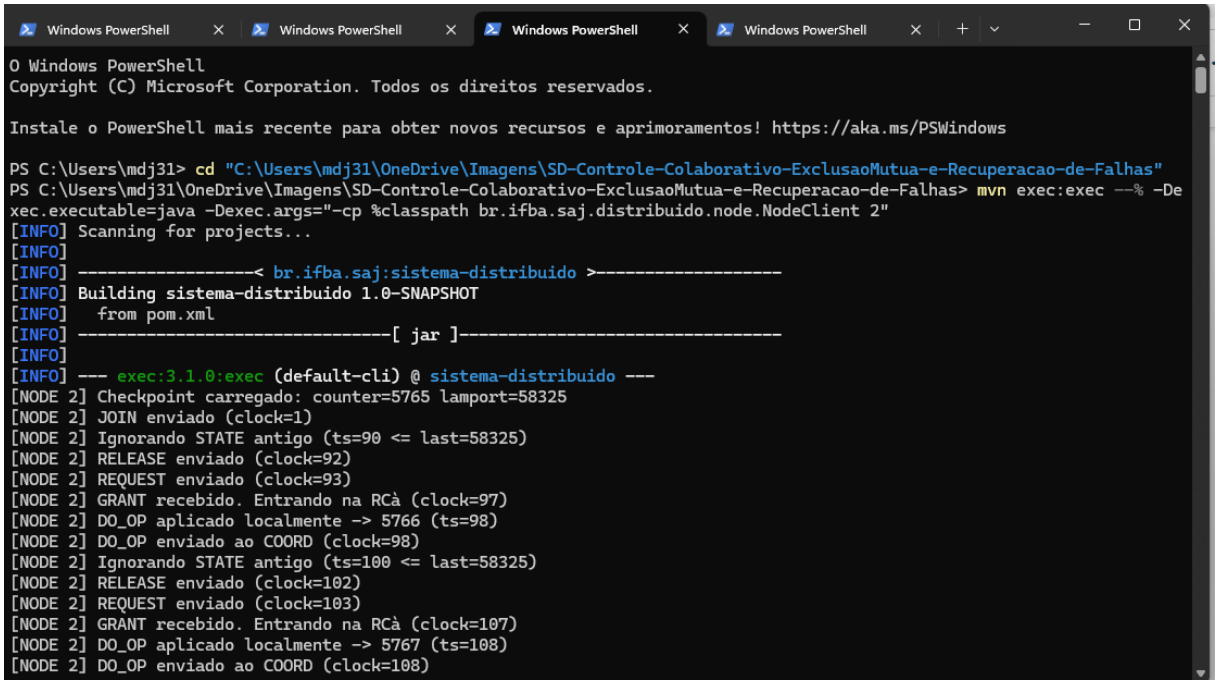
Exemplo de Log do Coordenador:



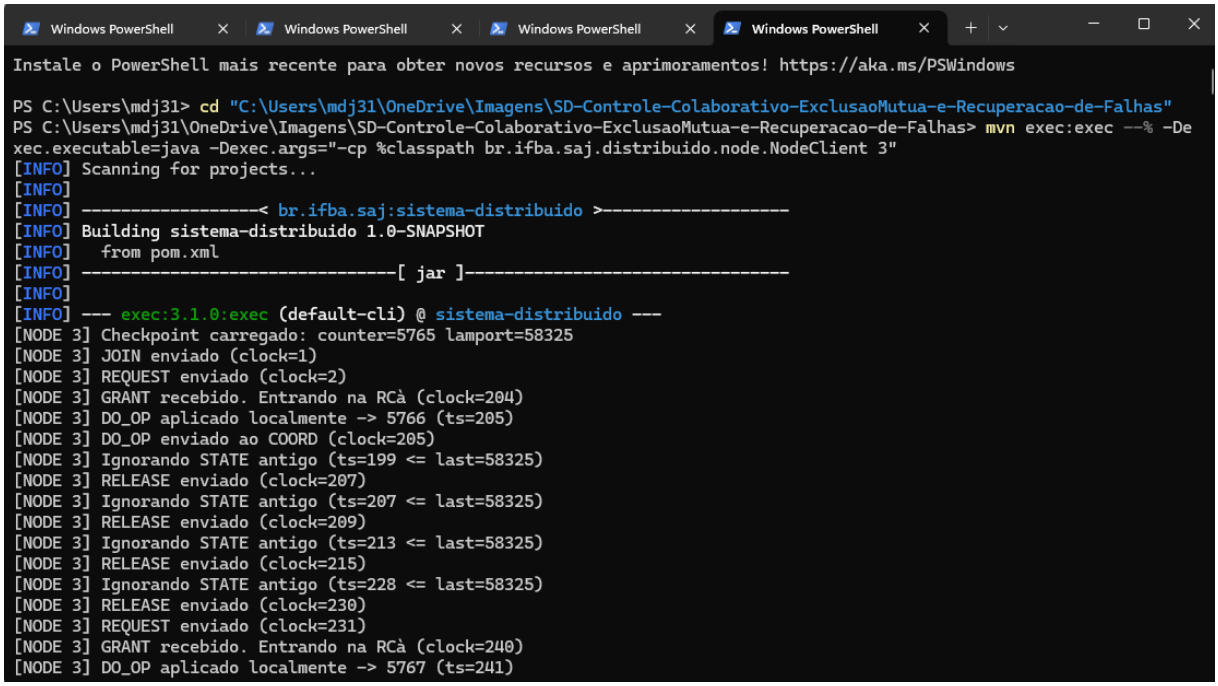
Exemplo de Log do Nó 1:



Exemplo de Log do Nó 2:



Exemplo de Log do Nó 3:



3. ALGORITMO DE EXCLUSÃO MÚTUA

3.1 Justificativa

O Sistema utiliza um algoritmo centralizado de exclusão mútua, no qual o servidor atua como coordenador para controlar o acesso à seção crítica, mantendo uma fila de requisições (PriorityBlockingQueue) priorizando pelo Time Stamp de Lamport. Cada Nó envia Request,, espera Grant, executa DO_OP e envia RELEASE, além de garantir um único nó na seção crítica por vez.

Segue abaixo pontos que justificam essa escolha:

- **Simplicidade de implementação:** o coordenador central decide a ordem de acesso, evitando conflitos.
- **Menor quantidade de mensagens:** cada nó precisa apenas enviar pedidos e receber permissões do servidor, reduzindo a sobrecarga de comunicação.
- **Evita deadlocks complexos:** como há apenas um ponto de decisão, a lógica de controle é direta e confiável.

Embora fosse possível usar algoritmos distribuídos como Ricart-Agrawala ou Bully, optamos pelo centralizado devido à facilidade de implementação e eficiência em sistemas pequenos, onde o ponto único de falha não é crítico para o experimento.

4. RELOGIO DE LAMPORT

Cada processo mantém um relógio lógico de Lamport (LamportClock) para ordenar eventos e garantir consistência, o relógio é incrementado a cada evento local e atualizado ao receber mensagens de outros processos e no servidor, o relógio de Lamport define a prioridade da fila de requisições, garantindo que os pedidos sejam atendidos em ordem causal permitindo coerência temporal mesmo em um sistema distribuído assíncrono, evitando que dois processos acessem a seção crítica simultaneamente. O coordenador usa o timeStamp para ordenar a fila de pedidos, garantindo dessa forma consistência causal e evitando que múltiplos nós entrem na RC simultaneamente, mesmo tendo atrasos.

5. REPLICAÇÃO E CONSISTÊNCIA EVENTUAL

O sistema implementa replicação do recurso compartilhado em cada nó, garantindo que o estado seja mantido de forma consistente, mesmo em ambiente distribuído e assíncrono. Cada nó mantém internamente um estado local representado pela classe NodeState, que armazena o contador (counter) e o timestamp lógico do último evento (lastLamportTs). Para garantir tolerância a falhas, são realizados checkpoints periódicos em arquivos no formato node-{id}-checkpoint.json, a cada 10 segundos. Antes de qualquer operação crítica (DO_OP), o nó cria uma pré-imagem (StateSnapshot) do estado atual, permitindo rollback em caso de falhas.

Quando uma operação é aplicada localmente, o nó envia a operação ao coordenador, que, por sua vez, propaga o novo estado (STATE) para todos os nós replicados. Cada nó aplica o estado recebido apenas se o timestamp de Lamport da operação for maior que o último registado (lastLamportTs), garantindo uma consistência monotônica do recurso, mesmo que mensagens STATE cheguem fora de ordem. Para simular atrasos típicos de rede ou processamento, uma flag de simulação (--delay) provoca um atraso de 5 segundos antes de aplicar a operação crítica, permitindo testar o comportamento do sistema em condições de consistência eventual.

6. RECUPERAÇÃO DE FALHAS

O sistema incorpora mecanismos robustos para recuperação de falhas. A mensagem ROLLBACK permite que o nó restaure seu último checkpoint, garantindo que o estado local seja consistente antes de qualquer operação subsequente. A pré-imagem salva antes de cada operação crítica possibilita a recuperação antes de confirmar alterações ao coordenador, evitando que falhas comprometam o progresso global.

Além disso, a flag --crash simula a queda abrupta de um nó durante a execução de uma operação crítica. Neste caso, a pré-imagem é salva em arquivo (node-{id}-precrash.json) para fins de diagnóstico. Após o reinício, o nó carrega automaticamente o último checkpoint disponível, evitando perda de dados e garantindo continuidade do processamento. O sistema lida com cenários de falha típicos, como falhas durante a operação crítica, mensagens STATE atrasadas ou fora de ordem e reinício de nós, aplicando rollback automático ou ignorando estados antigos, de acordo com os timestamps de Lamport.

Dessa forma, o mecanismo de checkpoints e rollback assegura tolerância a falhas e consistência eventual, permitindo que cada nó mantenha seu estado corretamente sincronizado com os demais, mesmo em condições adversas.

7. ANALISE E EFICIÊNCIA

A análise de eficiência do sistema considera tanto o número de mensagens trocadas quanto o tempo de resposta das operações críticas em um ambiente distribuído. Cada operação crítica segue um fluxo bem definido de comunicação: o nó envia um REQUEST ao coordenador, que responde com um GRANT autorizando a entrada na seção crítica. Em seguida, o nó aplica localmente a operação (DO_OP) e envia os resultados ao coordenador, que propaga o STATE atualizado para todos os nós replicados. Por fim, o nó envia a mensagem RELEASE, indicando que liberou a seção crítica.

O tempo de resposta das operações críticas é impactado por múltiplos fatores, como atrasos intencionais simulados por meio da flag --delay, o número de nós ativos e a ordem de chegada das requisições. Observações dos testes realizados mostraram que, mesmo com atrasos e falhas simuladas, o sistema manteve consistência monotônica e coerência temporal entre os nós.

Os logs e a contagem de operações confirmam que tanto a replicação quanto o mecanismo de rollback funcionam corretamente. A contagem de operações DO_OP e mensagens STATE evidencia que todas as alterações são propagadas e aplicadas em cada nó, mesmo quando ocorrem falhas ou reinícios. A comparação de execuções mostra que, embora atrasos aumentem a latência do sistema, eles não comprometem a consistência global, validando a robustez da estratégia de replicação e recuperação implementada.

7.1 Comparação em Tabela

Cenário	Nº de nós	Nº médio de mensagens por operação crítica	Tempo médio de resposta (LamportClock)	Observações
Execução inicial	1	5	9	Apenas um nó; operação simples, sem concorrência.
Concorrência baixa	2	6	9	Dois nós; replicação do estado via STATE; leve aumento de latência.
Concorrência média	3	7	10	Três nós; maior número de mensagens STATE para manter consistência; aumento moderado do tempo de resposta.

Explicação:

- **Nº médio de mensagens por operação crítica:** inclui REQUEST, GRANT, DO_OP, STATE (para cada nó replicado) e RELEASE.
- **Tempo médio de resposta:** diferença aproximada entre o momento em que o nó envia REQUEST e finaliza RELEASE.
- **Observações:** comentam o impacto da quantidade de nós na latência e nas mensagens.

8. CONCLUSÃO

O sistema desenvolvido atende integralmente aos requisitos propostos na atividade. A implementação da exclusão mútua centralizada garante que apenas um nó acesse a seção crítica por vez, mantendo a integridade do recurso compartilhado. O uso dos relógios de Lamport assegura a ordenação causal dos eventos, permitindo consistência temporal entre os nós mesmo em um ambiente assíncrono. A estratégia de replicação eventual implementada assegura consistência monotônica, de forma que todas as operações críticas são propagadas corretamente para os nós replicados. Os checkpoints periódicos, combinados com o mecanismo de rollback, permitem a recuperação eficiente de falhas, assegurando que nenhuma operação seja perdida mesmo em casos de crashes simulados.

A simulação de atrasos e falhas possibilitou avaliar o comportamento do sistema sob condições adversas, confirmando sua robustez e a eficácia dos mecanismos de controle e recuperação. A arquitetura adotada proporciona um ambiente adequado para estudo de cenários reais de sistemas distribuídos e de estratégias de recuperação de falhas, servindo como base sólida para análises de desempenho, consistência e confiabilidade.

9. LINK REPOSITORIO

- https://github.com/Ronaldo-Correia/SD-Controle-Colaborativo-ExclusaoMutua-e-Recuperacao-de-Falhas.git

10. REFERÊNCIAS

- TANENBAUM, A. S.; VAN STEEN, M. Distributed Systems: Principles and Paradigms. 3. ed. Pearson, 2017.
- LAMPORT, L. Time, Clocks, and the Ordering of Events in a Distributed System. Communications of the ACM, v. 21, n. 7, p. 558–565, 1978.
- CONTROLE.NET. Cliente-servidor: uma estrutura para a computação centralizada. Disponível em: https://www.controle.net/faq/cliente-servidor-uma-estrutura-para-a-computacao-centralizada. Acesso em: 28 ago. 2025.
- DEVMEDIA. Java Sockets: criando comunicações em Java. Disponível em: https://www.devmedia.com.br/java-sockets-criando-comunicacoes-em-java/9465. Acesso em: 28 ago. 2025.
- CONTROLE NET. *Cliente-servidor: uma estrutura para a computação centralizada*. Disponível em: https://www.controle.net/faq/cliente-servidor-uma-estrutura-para-a-computacao-centralizada. Acesso em: 28 ago. 2025.
- RANIDO, J. *Algoritmos para exclusão mútua*. MO441 – Computação Distribuída. Instituto de Computação, Universidade Estadual de Campinas. Disponível em: https://www.ic.unicamp.br/~ranido/mo441/slides/ExclusaoMutuaCompleto.pdf. Acesso em: 28 ago. 2025.