Relatório Técnico — Sistema Distribuído de Impressão com Exclusão Mútua (Ricart–Agrawala)

Pedro Rodrigues Alves
Lucas Gualtieri Firace Evangelista
Gabriel Felipe Quaresma de Oliveira

Orientador: Prof. Matheus Barros Pereira

31 de outubro de 2025

1 Arquitetura e Funcionamento do Sistema

O sistema desenvolvido é composto por dois tipos de processos principais:

- **Servidor de Impressão (burro):** Responsável apenas por receber solicitações de impressão e exibir as mensagens recebidas no terminal. Não realiza controle de concorrência nem possui estado compartilhado.
- **Clientes distribuídos:** Representam processos autônomos que desejam imprimir. Antes de enviar uma requisição de impressão, precisam obter acesso exclusivo à seção crítica, utilizando o **algoritmo de Ricart–Agrawala**.

1.1 Componentes Principais

Componente	Descrição
`distributed_printing.proto`	Definição dos serviços e mensagens gRPC.
`printer_server.py`	Servidor burro de impressão.
`printing_client.py`	Cliente com controle de exclusão mútua.
`run_all.bat`	Script de execução automática no Windows.

1.2 Fluxo de Execução

1. Cada cliente inicia um servidor gRPC local, apto a receber mensagens de outros clientes.
2. Quando um cliente deseja imprimir, ele incrementa seu relógio de Lamport e envia mensagens `RequestAccess` a todos os peers.
3. O cliente aguarda os `ACKs` de todos os peers antes de entrar na seção crítica.
4. Dentro da seção crítica, envia uma mensagem `PrintRequest` ao servidor burro.

5. Após imprimir, envia `ReleaseAccess` a todos os peers, liberando o recurso.

Essa arquitetura é totalmente descentralizada, não havendo um coordenador central.

2 Análise do Algoritmo de Ricart–Agrawala

O algoritmo de Ricart–Agrawala é uma solução de exclusão mútua distribuída baseada na troca de mensagens e ordenação lógica via relógios de Lamport.

2.1 Funcionamento

Cada cliente que deseja acessar a seção crítica:

- Envia uma mensagem `RequestAccess` a todos os outros processos.
- Espera por respostas positivas (`ACK`) de todos os peers.
- Somente após receber todas as respostas, entra na seção crítica.

Ao receber uma solicitação:

1. Se o cliente não está na seção crítica nem a requisitou, concede acesso imediatamente.
2. Se está requisitando e tem prioridade (menor timestamp), adia a resposta.
3. Caso contrário, concede acesso de imediato.

2.2 Versão Implementada

Foi implementada a versão **bloqueante** do algoritmo:

- As chamadas RPC `RequestAccess` permanecem bloqueadas até o peer liberar o recurso.
- Isso simplifica a implementação, evitando listas de pedidos adiados.

2.3 Propriedades Garantidas

Propriedade	**Como é Garantida**
Exclusão mútua	Apenas entra na seção crítica quem recebe todos os ACKs.
Ausência de deadlock	Pedidos ordenados por timestamps garantem progresso.
Justiça	Ordem de atendimento segue o tempo lógico de Lamport.

3 Resultados dos Testes

3.1 Cenário 1 — Um Cliente

```
python3 printer_server.py --port 50051
python3 printing_client.py --id 1 --server localhost:50051 --port 50052 --c
```

Resultado: O cliente entra diretamente na seção crítica e envia tarefas ao servidor. O servidor imprime corretamente, sem bloqueios ou trocas de mensagens entre peers.

3.2 Cenário 2 — Três Clientes Concorrentes

Executado via `run_all.bat`.

Comportamento:

- Quando dois clientes solicitam simultaneamente, o de menor timestamp entra primeiro.
- Após liberar, o próximo cliente é notificado e entra em sequência.

Logs:

```
[Cliente 1] Solicitando acesso (ts=12)
[Cliente 1] Ack recebido de localhost:50053
[Cliente 1] Entrou na seção crítica.
[Cliente 1] Resposta da impressora: Impressão concluída
[Cliente 1] Saindo da seção crítica.
```

Conclusão: Exclusão mútua garantida, sem deadlocks, mantendo coerência de timestamps.

3.3 Cenário 3 — Peer Offline

```
[Cliente 1] Falha RequestAccess em localhost:50053: StatusCode.UNAVAILABLE
[Cliente 1] Não obteve ack de todos os peers (1/2). Abortando pedido e tentando depoi
```

O sistema permanece funcional, ignorando temporariamente o peer inativo.

4 Dificuldades e Soluções Adotadas

Dificuldade	Solução
Sincronização de threads e relógio Lamport	Utilização de `threading.Lock()` para acesso seguro.
Erros na geração de stubs gRPC	Padronização de mensagens no arquivo `.proto`.
Falhas de comunicação e timeouts	Tratamento com `grpc.RpcError` e timeout de 10s.
Bloqueios em chamadas RPC	Uso de `Condition Variable` para liberação controlada.
Visualização do fluxo de mensagens	Padronização de logs com emojis e timestamps.
Execução multiplataforma	Criação de script `run_all.bat` para automação no Windows.

5 Conclusão

O sistema desenvolvido cumpre todos os objetivos propostos:

- Garante exclusão mútua distribuída usando apenas comunicação entre pares;

- Mantém ordenação causal via relógios de Lamport;
- Funciona de forma estável em cenários concorrentes e tolera falhas parciais;
- Implementa de maneira didática o algoritmo de Ricart–Agrawala.

O resultado final demonstra um sistema funcional, seguro e escalável, ideal para estudos práticos de sincronização e comunicação em sistemas distribuídos.