# Estudo prático do algoritmo de Ricart-Agrawala

**Jerson Vitor de Paula Gomes e Wallace Freitas Oliveira**

Departamento de Ciência da Computação
Pontifícia Universidade Católica de Minas Gerais
Belo Horizonte -- MG -- Brasil

## Introdução

Um dos problemas clássicos ao lidarmos com sistemas paralelos ou distribuídos é o princípio da exclusão mútua, segundo o qual diferentes processos ou threads, ao requisitarem acesso concorrente a um recurso compartilhado (denominado seção crítica), devem se coordenar para decidir quem terá permissão de acesso, evitando, assim, situações de acesso simultâneo que podem causar inconsistências no sistema.

Dentre os diversos algoritmos propostos para resolver esse problema, destaca-se o algoritmo de Ricart-Agrawala, que é o foco deste relatório. Por meio de um estudo prático, explora-se seu funcionamento, baseado na troca de mensagens entre os diferentes nós para coordenar o acesso à seção crítica, eliminando a necessidade de um coordenador central. São analisados aspectos importantes da computação distribuída, como a sincronização entre processos, o gerenciamento de requisições e respostas, bem como os desafios inerentes ao processamento simultâneo de mensagens em sistemas de comunicação baseados em TCP/IP.

## Cenário Explorado

O cenário explorado nesse trabalho representa uma simulação de um ambiente distribuido de impressão, e tem seu funcionamento baseado em processos, sendo esses divididos em dois grupos: processos do tipo **cliente**, que irão requisitar a impressão de seus dados de tempos em tempos, e um processo do tipo **servidor de impressão**, que irá apenas exibir os dados recebidos.

A concorrência por acesso a ao **servidor de impressão**, gera nos processos **clientes** a necessidade de tratamento de exclusão mútua, a ser realizada por meio do algoritmo de **Ricart-Agrawala**.

### Requisitos do Cenário Explorado

1. **Arquitetura distribuída**
    - Não deve haver um coordenador centralizado, todos os processos são iguais e trocam mensagens diretamente entre si.

2. **Algoritmo de Ricart–Agrawala (1981)**
    - Coordena o acesso à seção crítica via troca de mensagens `{timestamp, id}` e `{ok, id}`.

3. **Dimensão do sistema**
    - O sistema deve comportar **6 nós** iniciais, no entanto deve ser escalável para novos nós **clientes**.

4. **Decisão de acesso**
    - A cada **2s**, cada nó **cliente** gera um número pseudoaleatório em `[0, 1]`;
    - Se o valor for `> 0.5`, envia requisição de acesso ao **servidor de impressão**.

5. **Comunicação**
    - Todas as trocas ocorrem sobre **TCP/IP**, usando sockets padrão.

6. **Registro das impressões**
    - O log do sistema deve conter o **timestamp** e o **identificador** do processo responsável.

# Implementação

Para esse projeto, o uso da linguagem de programação **JAVA** se demonstrou a mais adequada por sua facilidade na implementação de **sistemas distribuidos**, seu suporte nativo a comunicação via **Sockets TCP** e seus recursos para manipulação de **threads** e **sinconização**.

## Estrutura do Projeto

O projeto encontra-se dividido em 5 pacotes, sendo eles pacote de comunicação (**Network**), computação distribuida (**Distributed**) e recursos auxiliares (**Enums**, **Interfaces**, **Uteis**), além de dois arquivos **.txt** de definições de inicio do sistema.

```
src/
├── Distributed/
```

```
│ ├── NodeDistributed.java
│ ├── NodeManager.java
│ ├── NodePrinterService.java
│ ├── RicartAgrawala.java
├── Enums/
│ └── MessageType.java
├── Interfaces/
│ └── DistributedNodeInterface.java
├── Network/
│ ├── Message.java
│ ├── NodeInfo.java
│ ├── NodeTCP.java
├── Uteis/
│ ├── LogCat.java
│ ├── Uteis.java
├── Main.java
├── serviceNode.txt
└── configNodes.txt
```

## Classes Principais

`NodeTCP`

Classe abstrata que define a infraestrutura básica de um nó **TCP**, responsável por iniciar um servidor que escuta conexões de entrada, receber mensagens, e enviá-las a outros nós. Serve como base para nós especializados, como o **NodeDistributed**, e abstrai detalhes da comunicação via **TCP/IP**, permitindo foco na lógica de exclusão mútua.

```
public abstract class NodeTCP {

    protected NodeInfo nodeInfo;
    protected ServerSocket serverSocket;
    protected Executor executor;

    public abstract void start();
    public void startServer();
    private void listenForConnections();
    protected void handleIncomingConnection(Socket socket);
    protected void handleMessage(Message message, String senderIp);
    public void sendMessage(String targetIp, int targetPort, Message message);

}
```

`NodeDistributed`

Classe que estende **NodeTCP** e implementa a lógica de um nó participante na rede distribuída, coordenando-se com outros nós através do algoritmo de **Ricart-Agrawala**. Gerencia a entrada e saída da rede, bem como as trocas de mensagens de controle **(JOIN, LEAVE, REQUEST, REPLY)**. Integra-se ao sistema como o principal agente que executa a exclusão mútua distribuída e interage com o **RicartAgrawala**.

```
public class NodeDistributed extends NodeTCP
implements DistributedNodeInterface {

    private final NodeManager manager;
    private final RicartAgrawala ricartAgrawala;
    private CountDownLatch latchJoinConnect;

    private CountDownLatch latchCriticalSection;
    private CountDownLatch latchCritialSectionToServer;

    private NodeInfo serviceInfo;
    public void start();
    public void joinNetworkIfNeeded();
    private void run()
    protected void handleMessage(Message message, String senderIp);
    public void requestBroadCast(Integer timestamp);
    public void reply(NodeInfo toNode, Integer timestamp);
    public void release(NodeInfo toNode, Integer timestamp);
    public void sendMessage(NodeInfo toNode, Message message);
    public void runCriticalSection();
    public List<NodeInfo> getKnownNodes();

}
```

### `RicartAgrawala`

Classe que implementa a lógica do algoritmo de **Ricart-Agrawala**, responsável por coordenar a entrada do nó na seção crítica de forma segura e ordenada. Mantém o controle sobre o estado do nó (solicitando, em seção crítica ou livre), gerencia relógio lógico e lista de respostas pendentes. Integra-se ao sistema como um módulo auxiliar usado por **NodeDistributed** para gerenciar o acesso concorrente aos recursos compartilhados.

```
public class RicartAgrawala {

    private DistributedNodeInterface delegate;

    private boolean requestingCriticalSection = false;
    private boolean inCriticalSection = false;
    private Integer timestamp = 0;

    private HashSet<NodeInfo> pendingReplies = new HashSet<>();
    private final List<NodeInfo> deferredNodes = new CopyOnWriteArrayList<>();
    public synchronized void requestCriticalSection();
    public synchronized void onRelease(Message messageRelease);
    public synchronized void onRequest(Message messageRequest, String currentID);
    public synchronized void onReply(Message messageReply);
    public synchronized void releaseCriticalSection();
    public synchronized void toggleInCriticalSection();
    public synchronized boolean isInCriticalSection();
    public synchronized boolean isRequestingCriticalSection();
    public synchronized void updateClock(Integer receveidTimestamp);
    public synchronized Integer getTimestamp();
```

```
}
```

## Como Executar

### 1. Pré-requisitos

- **Java Development Kit (JDK):** versão 11 ou superior
- **Git**

### 2. Clonando o repositório

```
git clone https://github.com/Olivwallace/Ricart-Agrawala.git
cd Ricart-Agrawala
```

### 3. Compilação

Para compilar a solução basta acessar via terminal a hieraquia da pasta **src/** e executar o seguinte comando.

```
javac Main.java
```

### 3. Execução do código

Em cada terminal que deseja fazer parte do sistema, seja como o **servidor de impressão** ou como **cliente**, execute o seguinte comando.

```
java Main.java "nome_do_processo" "ip-da-maquina" porta_do_processo "é_servidor"
```

#### Parâmetros do comando

| Parâmetro | Descrição | Exemplo |
|---|---|---|
| `nome_do_processo` | Identificador único do nó (cliente ou servidor). | `Cliente-1` |
| `ip-da-maquina` | Endereço IP da máquina onde o nó está rodando. | `192.168.0.10` |
| `porta_do_processo` | Porta TCP em que o nó irá escutar. | `5001` |
| `é_servidor` | Flag que indica se é servidor ( `true` ) ou cliente ( `false` ). | `false` / `true` |

`Exemplo: iniciando servidor de impressão`

```
java Main.java "Process0" "192.168.0.1" 5000 true
```

`Exemplo: iniciando cliente`

```
java Main.java "Process0" "192.168.0.1" 5001 false
```

**OBS**

Baseando-se na ideia de computação distribuída, o nosso código foi feito pensado na execução em vários terminais, simulando processos de **clientes** e o de **servidor**, e está configurado para ser escalável, suportando a inserção e saída de nós **clientes** do sistema em tempo de execução. Para isso, é necessário atentar-se para as seguintes definições a serem realizadas:

- **Primeiro Cliente:**
  Para o primeiro processo **cliente** do sistema, o arquivo de difinição ***configNodes.txt*** é opcional. Sendo necessário apenas as definições do ***serviceNode.txt***, conforme demonstradas abaixo:

  `serviceNode.txt`

  ```
  Service,192.168.0.2,5000
  ```

- **Demais Clientes:**
  Um processo só pode ingressar no sistema se o mesmo conhecer o **servidor de impressão** e um ou mais nós **clientes** que pertece ao sistema e encontra-se ativo, para isso, os arquivos de ***configNodes.txt*** e ***serviceNode.txt*** devem está preenchidos no seguinte formato:

  `configNodes.txt`

  ```
  Process0,192.168.0.2,500x
  ProcessX,192.168.0.x,xxxx
  ```

  `serviceNode.txt`

  ```
  Service,192.168.0.2,5000
  ```

## Desafios

O principal desafio deste trabalho foi a implementação e validação, do algoritmo de exclusão mútua de Ricart-Agrawala com pensamento além da teoria pensando na forma distribuída, garantindo a correta coordenação entre múltiplos nós sem a presença de um coordenador central. Isso envolveu superar dificuldades relacionadas à sincronização de processos, ao tratamento de mensagens concorrentes via sockets TCP/IP e o gereciamento e controle preciso do relógio lógico na idéia de Lamport e das filas de requisições pendentes.

## Conclusão

Neste trabalho, implementamos o algoritmo de Ricart–Agrawala em Java, usando sockets TCP/IP e relógios de Lamport para coordenar o acesso exclusivo de múltiplos nós a um servidor de impressão. Testes com seis nós iniciais e a inclusão dinâmica de novos clientes confirmaram que o algorítmo proposto no trabalho mostrou-se eficaz na comunicação por meio do protocolo TCP/IP e a sua capacidade de evitar acessos simultâneos à seção crítica.

## Referências


Ricart, G. and Agrawala, A. K. (1981). An optimal algorithm for mutual exclusion in computer networks. Commun. ACM, 24(1):9–17.

TRUNC8. ricart-agrawala-algorithm. GitHub. Disponível em: https://github.com/trunc8/ricart-agrawala-algorithm/tree/master. Acesso em: 24 maio 2025.

HALYAL, Nisha. Ricart-Agrawala-for-distributed-mutual-exclusion-: Proj.java. GitHub. Disponível em: https://github.com/NishaHalyal/Ricart-Agrawala-for-distributed-mutual-exclusion-/blob/master/src/sockets/Proj.java. Acesso em: 24 maio 2025.