

# Plataforma Distribuída de Processamento Colaborativo de Tarefas

**ProjetoFinal - SistemasDistribuídos**

IFBA - Campus Santo Antônio de Jesus

Apresentado por:

Hildemar/Thiago/Kleber

Professor: Felipe



# Agenda da Apresentação

01

---

## **Introdução e Objetivos**

Contextualização do projeto e metas estabelecidas

02

---

## **Fundamentação Teórica**

Conceitos de sistemas distribuídos aplicados

03

---

## **Arquitetura do Sistema**

Componentes e estrutura da plataforma

04

---

## **Implementação Técnica**

Decisões de desenvolvimento e tecnologias

05

---

## **Resultados e Análise**

Evidências de funcionamento e avaliação crítica

# Introdução ao Projeto

Este trabalho apresenta o desenvolvimento de uma **Plataforma Distribuída de Processamento Colaborativo de Tarefas**, implementada como projeto final da disciplina Sistemas Distribuídos.

## Objetivos Principais

- Receber e distribuir tarefas entre múltiplos nós
- Manter estado global consistente
- Garantir recuperação automática de falhas
- Implementar orquestração com backup



**Tecnologia:** Java com sockets TCP/UDP puros, sem frameworks externos ou ferramentas REST.

# Fundamentos de Sistemas Distribuídos

"Um sistema distribuído consiste em um conjunto de computadores independentes que se apresentam ao usuário como um sistema único e coerente" - Tanenbaum & Van Steen

## Transparência

O sistema deve apresentar-se como uma unidade coesa, ocultando a complexidade da distribuição dos usuários

## Compartilhamento

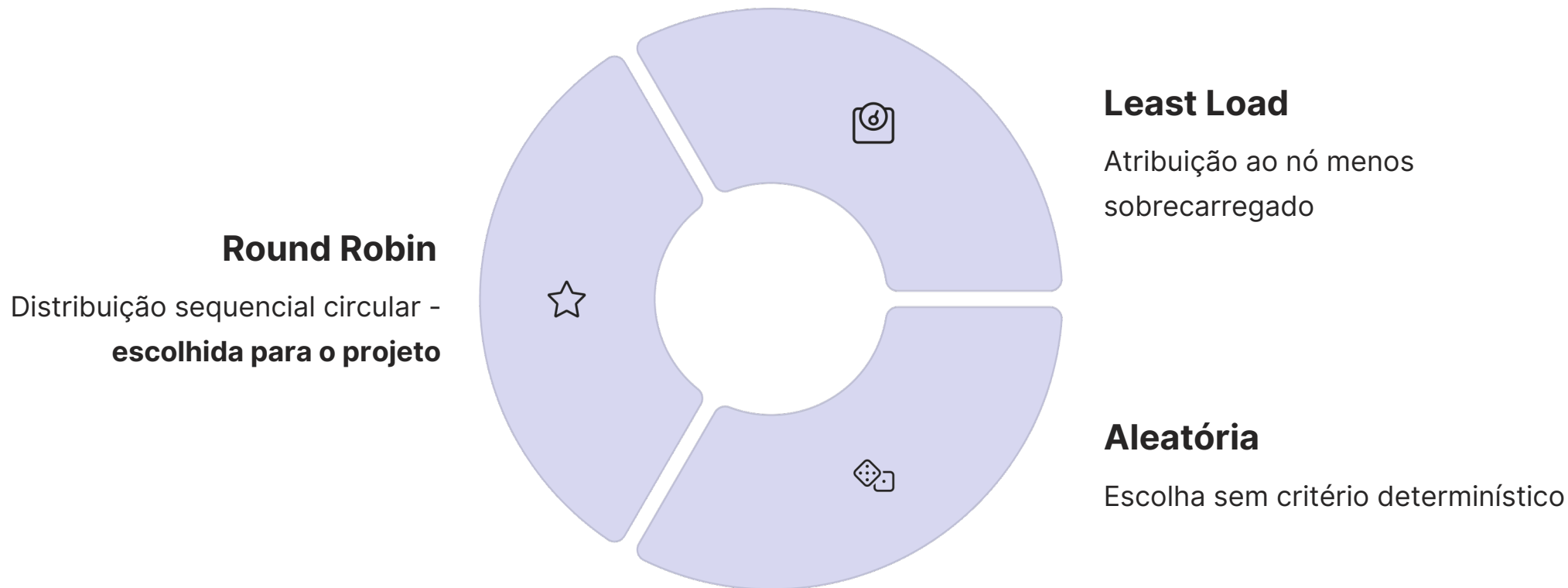
Recursos computacionais são compartilhados eficientemente entre múltiplos componentes

## Tolerância a Falhas

O sistema mantém funcionamento mesmo quando componentes individuais falham



# Estratégias de Balanceamento de Carga



A política **Round Robin** foi adotada por sua simplicidade, previsibilidade e capacidade de garantir distribuição equitativa entre workers ativos.





# Mecanismos de Tolerância a Falhas



## Heart beat

Workersenviam sinais periódicos confirmando atividade ao orquestrador principal



## Failover Automático

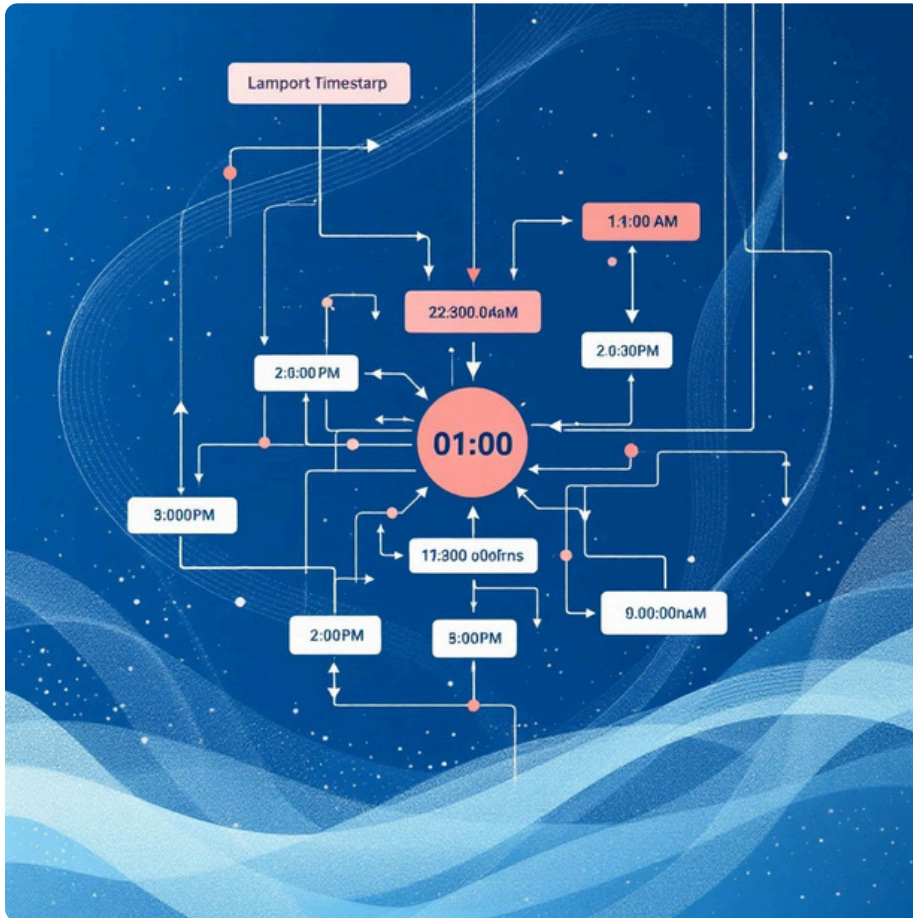
Orquestrador backupassume controle automaticamente em caso de falha do principal



## Redistribuição

Tarefas inacabadas são reatribuídas a outros nós ativos quando um worker falha

# Relógios Lógicos de Lamport



Em sistemas distribuídos, a ausência de relógio global consistente exige mecanismos especiais de ordenação temporal.

## Aplicação no Projeto

- Ordenação de submissões de tarefas
- Sincronização de atualizações de estado
- Garantia de consistência em operações de failover

**i** Os timestamps lógicos asseguram relação causal entre eventos, prevenindo inconsistências durante redistribuição e failover.

# Arquitetura do Sistema



## Cliente

Submetetarefas e consulta status via TCP com o orquestrador principal



## Orquestrador Principal

Coordena o sistema, balanceia carga e monitora workers



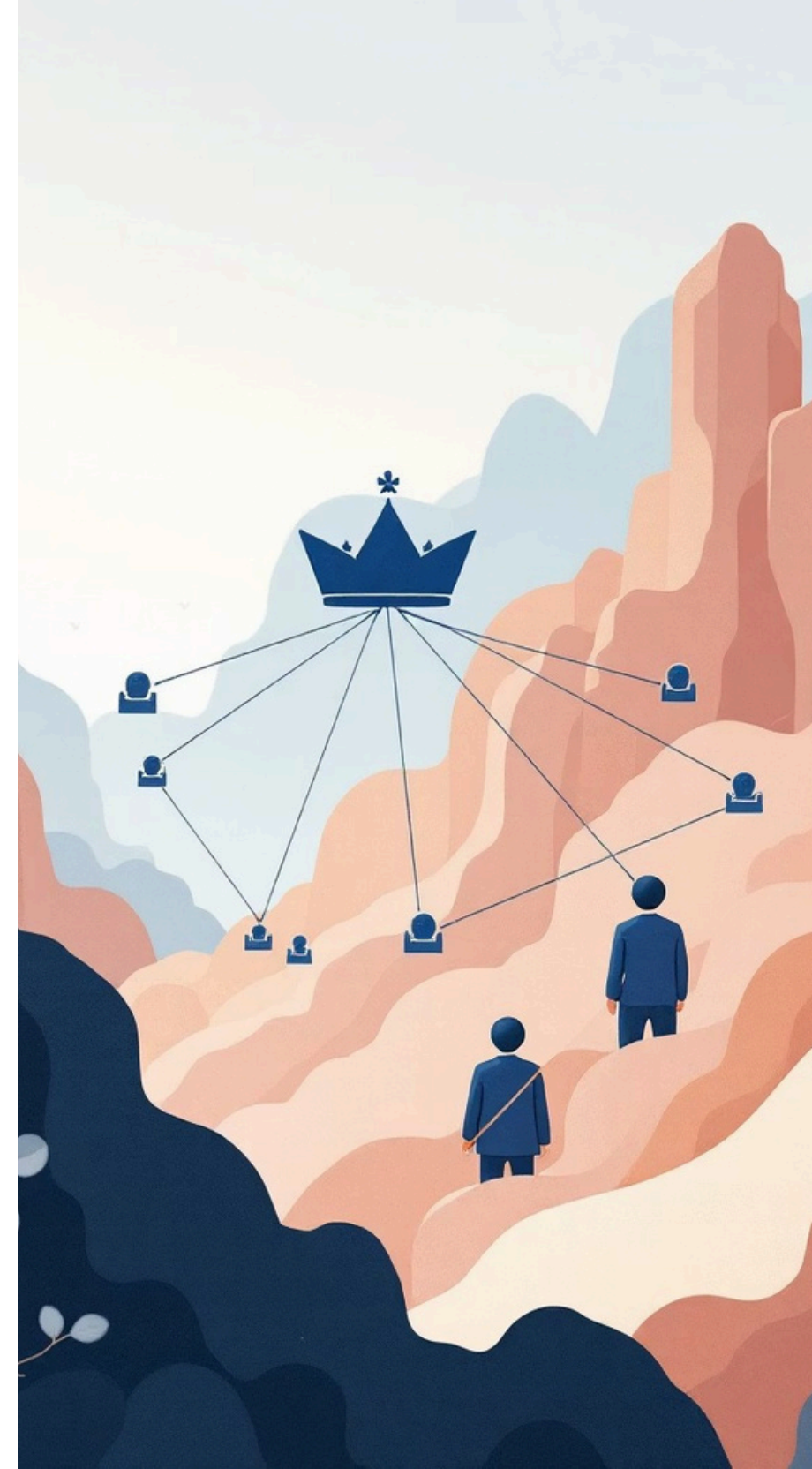
## Orquestrador Backup

Mantém cópia do estado via UDP multicast, assume controle em falhas



## Workers

Executam tarefas atribuídas e enviam heart beats periódicos





# Arquitetura do Sistema

## Estrutura do Projeto

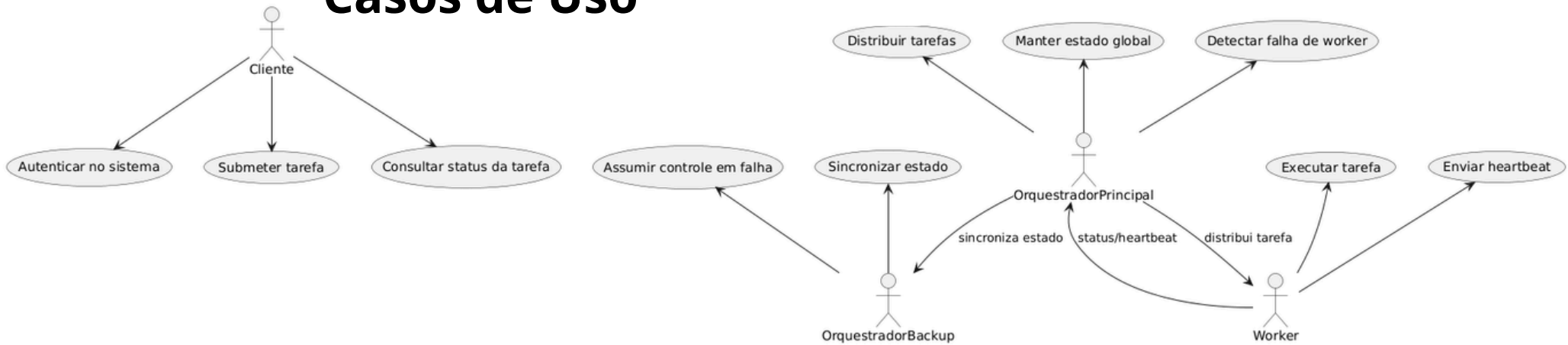
Organizada em pacotes que separam responsabilidades:

- **cliente** – interação com o usuário
- **modelo** – entidades e estados do sistema
- **orquestrador** – coordenação da execução distribuída
- **worker** – execução das tarefas
- **util** – funções auxiliares
- **scripts** – automação de compilação e execução  
Tudo foi pensado para manter clareza, modularidade e facilitar a manutenção.

```
PLATAFORMADISTRIBUIDA
|
|─ bin
|
|─ src
|   |─ cliente
|   |   └─ Cliente.java
|   |
|   |─ modelo
|   |   └─ EstadoGlobal.java
|   |   └─ Tarefa.java
|   |   └─ Usuario.java
|   |
|   |─ orquestrador
|   |   └─ OrquestradorBackup.java
|   |   └─ OrquestradorPrincipal.java
|   |
|   |─ util
|   |   └─ RelogioLamport.java
|   |
|   └─ worker
|       └─ Worker.java
|
|─ compile_and_run.sh
|─ README.md
```

# Diagramas UML

## Casos de Uso

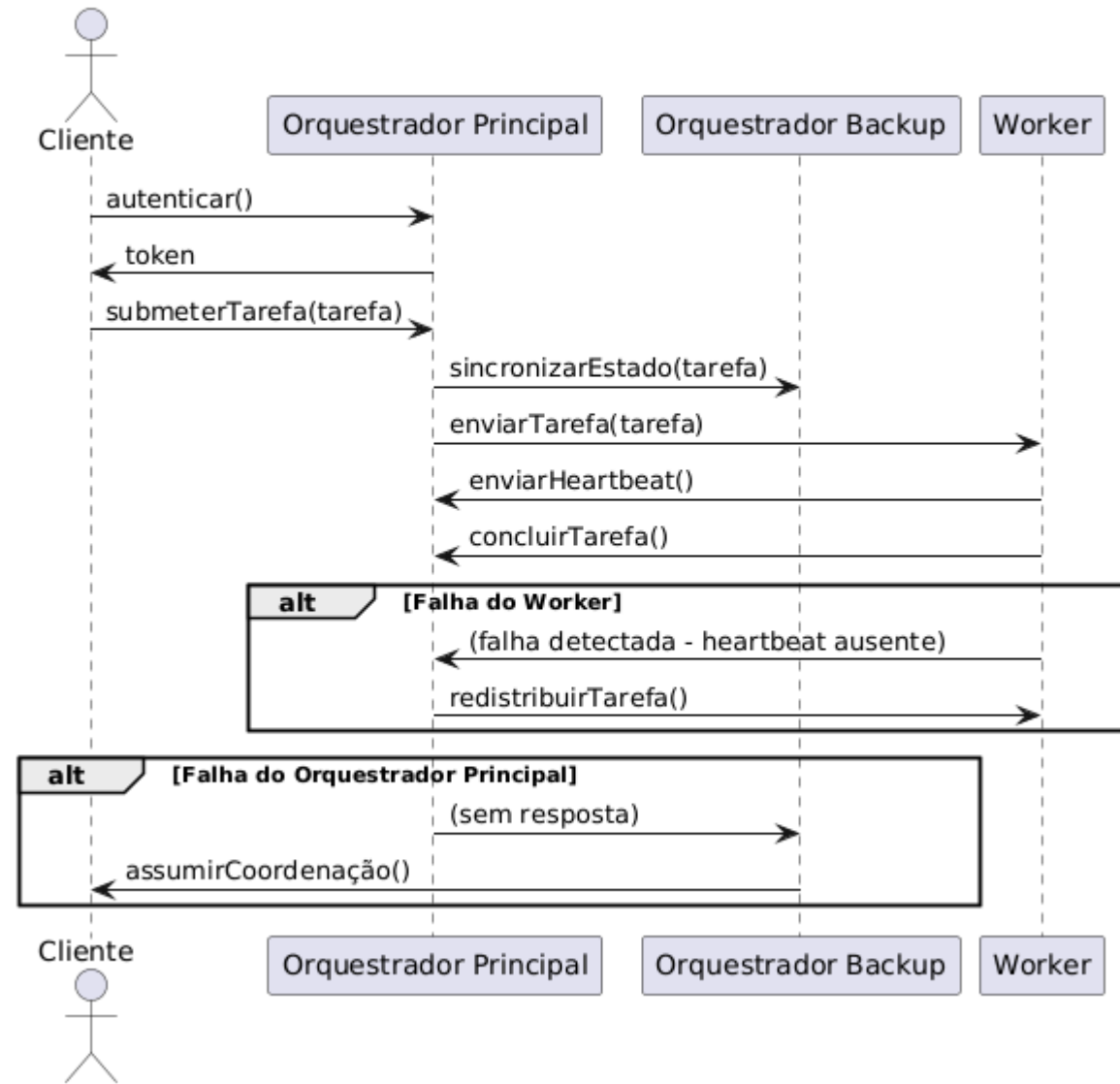
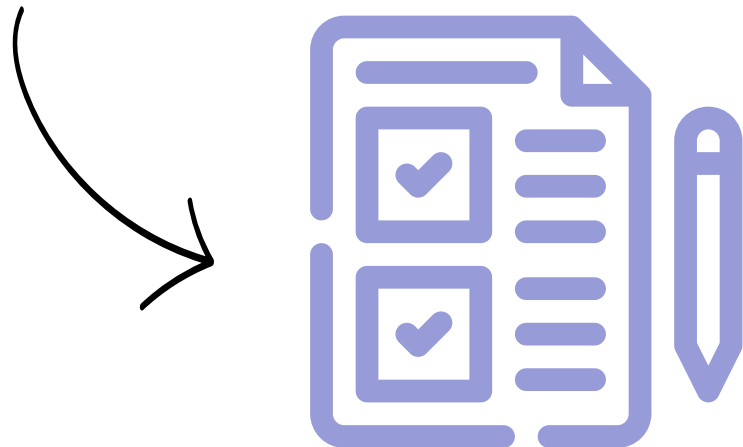


→ A ideia desse diagrama é mostrar de forma simples e clara quais são os papéis no sistema e o que cada um pode fazer. Ele ajuda a ter uma visão de alto nível, sem entrar nos detalhes técnicos, mas já deixando evidente que o sistema não depende apenas de um único ponto, já que existe a redundância com o orquestrador de backup e o mecanismo de detecção de falhas dos workers.

# Diagramas UML

## Sequencia

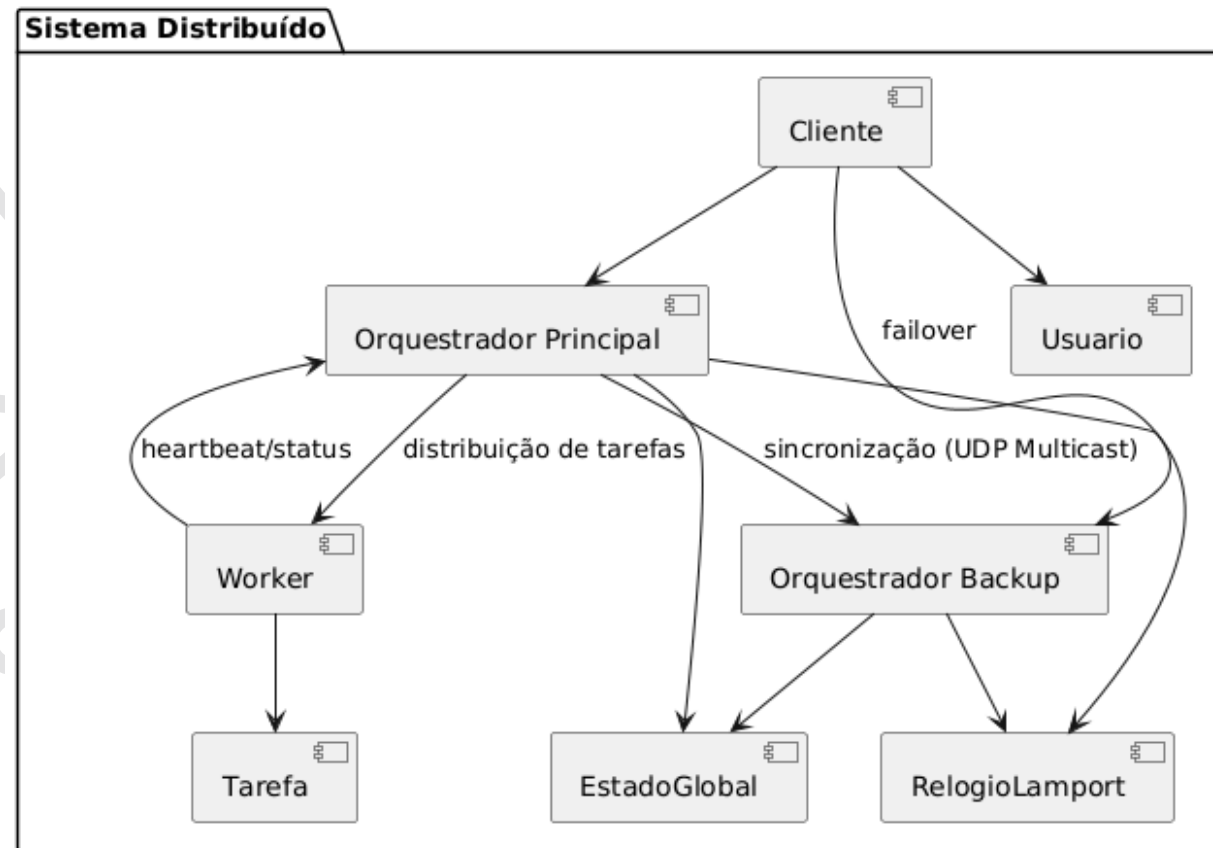
Nesse diagrama também aparecem os cenários de falha. Se um worker parar de responder, o orquestrador principal percebe pela ausência de heartbeat, marca esse worker como inativo e redistribui as tarefas dele para outro worker disponível. Já no caso do orquestrador principal falhar, o backup detecta a falta de comunicação, assume o papel de principal e continua o funcionamento sem que o cliente perca as informações das tarefas. Esse diagrama é importante porque mostra não só o funcionamento normal, mas também como o sistema reage em situações de erro.



# Diagramas UML

## Componentes

Esse diagrama de componentes mostra a arquitetura do sistema, ou seja, quais são as partes que compõem ele e como essas partes se relacionam. O cliente se comunica com o orquestrador principal por meio de gRPC ou sockets TCP. O orquestrador principal e o backup se comunicam entre si usando UDP multicast para manter o estado sincronizado. O orquestrador principal também é quem envia as tarefas para os workers e recebe de volta os heartbeats e os resultados das execuções.





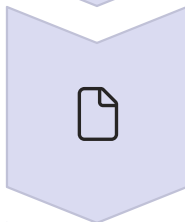


# Padrões de Comunicação



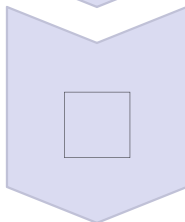
## **Cliente    Orquestrador Principal**

**TCP:** Garante confiabilidade na submissão e consulta de tarefas



## **Orquestrador Principal    Backup**

**UDPMulticast:** Sincronização eficiente do estado global



## **Orquestrador    Workers**

Atribuição de tarefas e recebimento de heart beats

# Padrões de Comunicação

## Relatos de Execuções

1

```
[OrquestradorPrincipal] TCP server started on port 5000
[OrquestradorPrincipal] Worker registrado: worker1
[OrquestradorPrincipal] Worker registrado: worker2
[OrquestradorPrincipal] Worker registrado: worker3
[OrquestradorPrincipal] Cliente autenticado: junior
[OrquestradorPrincipal] Recebeu tarefa: t1,junior,10,PENDING
[OrquestradorPrincipal] Enviou tarefa t1 para worker worker1
[OrquestradorPrincipal] Recebeu tarefa: t2,junior,10,PENDING
[OrquestradorPrincipal] Enviou tarefa t2 para worker worker2
[OrquestradorPrincipal] Tarefa concluída: t1,junior,10,PENDING por worker worker1
[OrquestradorPrincipal] Recebeu tarefa: t3,junior,10,PENDING
[OrquestradorPrincipal] Enviou tarefa t3 para worker worker3
[OrquestradorPrincipal] Tarefa concluída: t2,junior,10,PENDING por worker worker2
[OrquestradorPrincipal] Tarefa concluída: t3,junior,10,PENDING por worker worker3
```

Cliente efetuando login para funções

2

Orquestrador principal dividindo tarefas entre Workers

```
Backup] Estado recebido via multicast: t1,junior,10,PENDING:worker1:RUNNING
Backup] Estado recebido via multicast: t1,junior,10,PENDING:worker1:RUNNING
Backup] Estado recebido via multicast: t1,junior,10,PENDING:worker1:RUNNING
Backup] Estado recebido via multicast: t1,junior,10,PENDING:worker1:RUNNING
Backup] Estado recebido via multicast: t2,junior,10,PENDING:worker2:RUNNING;t1,junior,10,PENDING:worker1:RUNNING
Backup] Estado recebido via multicast: t2,junior,10,PENDING:worker2:RUNNING
Backup] Estado recebido via multicast: t2,junior,10,PENDING:worker2:RUNNING
Backup] Estado recebido via multicast: t2,junior,10,PENDING:worker2:RUNNING;t3,junior,10,PENDING:worker3:RUNNING
Backup] Estado recebido via multicast: t3,junior,10,PENDING:worker3:RUNNING
Backup] Estado recebido via multicast: t3,junior,10,PENDING:worker3:RUNNING
Backup] Estado recebido via multicast: t3,junior,10,PENDING:worker3:RUNNING
Backup] Estado recebido via multicast: t3,junior,10,PENDING:worker3:RUNNING
Backup] Estado recebido via multicast: t3,junior,10,PENDING:worker3:RUNNING
```

```
Backup] Não recebeu estado do principal recentemente. Assumindo papel de Orquestrador Principal...
Backup-as-Primary] TCP server started on port 5000
```

3

Caso o orquestrador principal cair, backup entra em ação

Backup de todo o processo

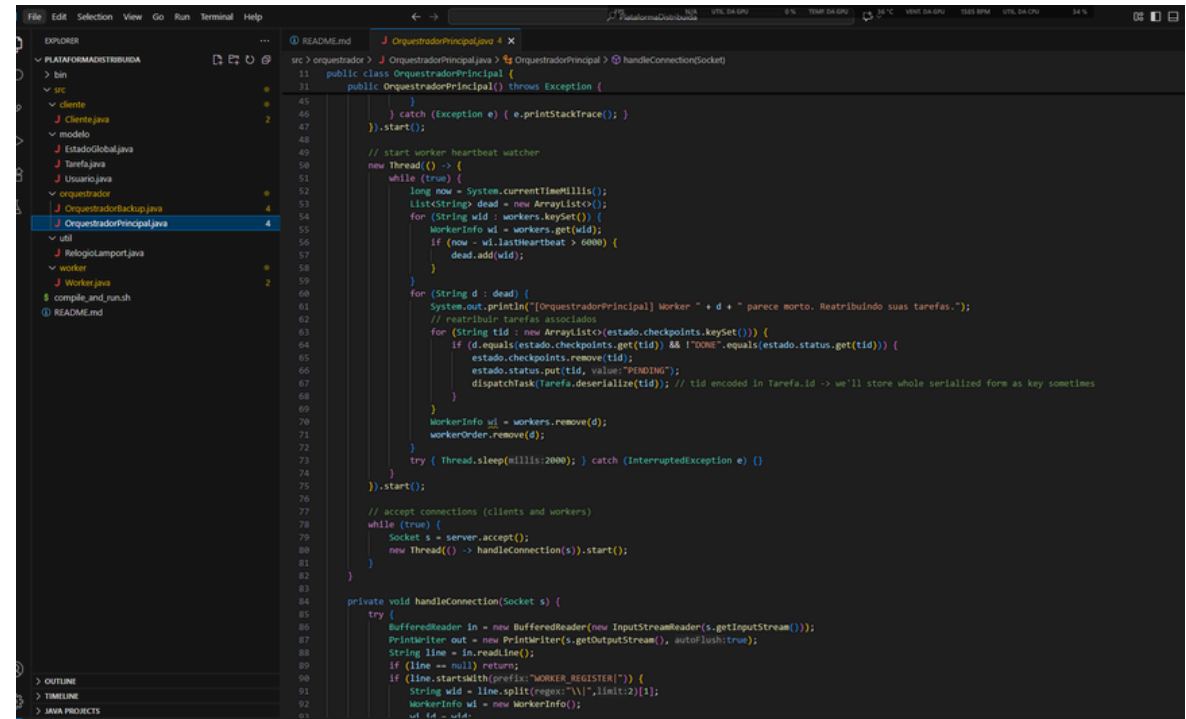
# Conclusões e Perspectivas

## Objetivos Alcançados

- Plataforma funcional com balanceamento Round Robin
- Tolerância a falhas com heartbeat e failover
- Autenticação básica implementada
- Uso efetivo de relógios lógicos de Lamport

## Melhorias Futuras

- Balanceamento dinâmico baseado em métricas reais
- Criptografia TLS para comunicação segura
- Persistência em banco distribuído
- Containerização com Docker



```
11 public class OrquestradorPrincipal {
12     public OrquestradorPrincipal() throws Exception {
13     }
14     public void start() {
15         // start worker heartbeat watcher
16         new Thread(() -> {
17             while (true) {
18                 long now = System.currentTimeMillis();
19                 List<String> dead = new ArrayList<>();
20                 for (String wid : workers.keySet()) {
21                     WorkerInfo wi = workers.get(wid);
22                     if (now - wi.lastHeartbeat > 60000) {
23                         dead.add(wid);
24                     }
25                 }
26                 for (String d : dead) {
27                     System.out.println("[OrquestradorPrincipal] Worker " + d + " parece morto. Reatribuindo suas tarefas.");
28                     // reatribuir tarefas associados
29                     for (String tid : new ArrayList<>(estado.checkpoints.keySet())) {
30                         if (d.equals(estado.checkpoints.get(tid)) && !"DONE".equals(estado.status.get(tid))) {
31                             estado.checkpoints.remove(tid);
32                             estado.status.put(tid, "PENDING");
33                             dispatchTask(Tarefa.deserialize(tid)); // tid encoded in Tarefa.id -> we'll store whole serialized form as key sometimes
34                         }
35                     }
36                     WorkerInfo wi = workers.remove(d);
37                     workerOrder.remove(d);
38                 }
39                 try { Thread.sleep(1000); } catch (InterruptedException e) {}
40             }
41         }).start();
42         // accept connections (clients and workers)
43         while (true) {
44             Socket s = server.accept();
45             new Thread(() -> handleConnection(s)).start();
46         }
47     }
48     private void handleConnection(Socket s) {
49         try {
50             BufferedReader in = new BufferedReader(new InputStreamReader(s.getInputStream()));
51             PrintWriter out = new PrintWriter(s.getOutputStream(), autoFlush);
52             String line = in.readLine();
53             if (line == null) return;
54             if (line.startsWith(prefix + "WORKER_REGISTER")) {
55                 String wid = line.split(regex)[1];
56                 WorkerInfo wi = new WorkerInfo();
57                 wi.id = wid;
58             }
59         } catch (IOException e) {}
60     }
61 }
```

A plataforma demonstra com sucesso os conceitos fundamentais de sistemas distribuídos, servindo como base sólida para implementações mais complexas.