



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

CURSO DE TECNOLOGIA EM ANÁLISE E DESENVOLVIMENTO DE SISTEMAS

Hildemar Lemos de Santana Junior

Kleberson de Jesus Souza

Thiago Sampaio Santos

**Relatório-Plataforma Distribuída de Processamento Colaborativo de
Tarefas(Sistemas Distribuídos)**

SANTO ANTÔNIO DE JESUS – BA

2025

SUMÁRIO

- 1. Introdução**
- 2. Fundamentação Teórica**
 - 2.1 Sistemas Distribuídos
 - 2.2 Balanceamento de Carga
 - 2.3 Tolerância a Falhas e Replicação
 - 2.4 Relógios Lógicos de Lamport
 - 2.5 Comunicação Cliente-Servidor
- 3. Descrição da Arquitetura do Sistema**
 - 3.1 Componentes Principais
 - 3.2 Comunicação
 - 3.3 Diagramas UML
- 4. Escolhas Técnicas e Justificativas**
 - 4.1 Política de Balanceamento
 - 4.2 Protocolo de Comunicação
 - 4.3 Autenticação e Segurança
 - 4.4 Logs e Monitoramento
- 5. Evidências de Execução**
 - 6.1 Submissão e Distribuição de Tarefas
 - 6.2 Failover e Redistribuição de Tarefas
 - 6.3 Logs e Testes de Falhas
- 6. Análise Crítica**
 - 7.1 Pontos Fortes do Sistema
 - 7.2 Limitações
 - 7.3 Melhorias Futuras
- 7. Conclusão**
- 8. Referências**

1. Introdução

Neste trabalho, nós descrevemos o desenvolvimento da Plataforma Distribuída de Processamento Colaborativo de Tarefas, implementada como projeto final da disciplina *Sistemas Distribuídos* do IFBA — Campus Santo Antônio de Jesus. O objetivo do projeto foi projetar e implementar, em grupo, uma plataforma capaz de receber tarefas de clientes, distribuir essas tarefas entre múltiplos nós de processamento (workers), manter um estado global consistente e recuperar operação em caso de falhas por meio de um orquestrador de backup.

Optamos por uma implementação em Java, usando sockets TCP/UDP para toda a comunicação entre componentes. Toda a execução e os testes foram realizados em terminais separados (um terminal para o orquestrador principal, outro para o orquestrador backup e terminais adicionais para cada worker e para o cliente), sem uso de ferramentas HTTP/REST externas (como Postman ou Insomnia) ou frameworks de RPC — a comunicação é feita por mensagens TCP/UDP em Java puro, o que nos permitiu controlar de forma direta os mecanismos de heartbeat, replicação e failover.

O artefato enviado (repositório) contém o código-fonte e scripts de execução com a seguinte estrutura principal:

```
PlataformaDistribuida/  
├─ README.md  
├─ compile_and_run.sh  
└─ src/  
    ├─ cliente/Cliente.java  
    ├─ modelo/Tarefa.java  
    ├─ modelo/Usuario.java  
    ├─ modelo/EstadoGlobal.java  
    ├─ orquestrador/OrquestradorPrincipal.java  
    ├─ orquestrador/OrquestradorBackup.java  
    ├─ worker/Worker.java  
    └─ util/RelogioLamport.java
```

2. Fundamentação Teórica

O desenvolvimento de sistemas distribuídos tem se tornado cada vez mais relevante no contexto atual da computação, principalmente diante da crescente demanda por processamento paralelo, tolerância a falhas e escalabilidade em aplicações críticas. A literatura apresenta diversos conceitos fundamentais que sustentam a implementação de uma plataforma colaborativa de processamento de tarefas, os quais são abordados a seguir.

2.1 Sistemas Distribuídos

Um sistema distribuído consiste em um conjunto de computadores independentes que se apresentam ao usuário como um sistema único e coerente. De acordo com Tanenbaum e Van Steen (2017), a principal motivação por trás dessa abordagem é fornecer transparência de distribuição, compartilhamento de recursos e tolerância a falhas.

No contexto do projeto, o sistema foi implementado de forma a garantir:

- Execução paralela de tarefas por múltiplos workers.
- Orquestração centralizada e backup redundante para continuidade do serviço.
- Comunicação entre processos por meio de sockets TCP e UDP multicast.

2.2 Balanceamento de Carga

O balanceamento de carga é a técnica responsável por distribuir tarefas entre diferentes nós de processamento, evitando sobrecarga e garantindo melhor aproveitamento dos recursos. Entre as políticas mais comuns, destacam-se:

- Round Robin: distribuição sequencial, de forma circular.
- Least Load: atribuição da tarefa ao nó menos sobrecarregado.
- Aleatória: escolha de nó sem critério determinístico.

No projeto desenvolvido, adotou-se a política Round Robin, por sua simplicidade e previsibilidade. Essa política foi implementada no orquestrador principal, que mantém um contador de distribuição e assegura que os workers recebam tarefas de forma equitativa.

Exemplo simplificado da lógica utilizada no nosso projeto:

```
1 private int currentIndex = 0;
2
3 public Worker selecionarWorker(List<Worker> workers) {
4     if (workers.isEmpty()) return null;
5     Worker selecionado = workers.get(currentIndex);
6     currentIndex = (currentIndex + 1) % workers.size();
7     return selecionado;
8 }
```

Esse mecanismo garante que cada tarefa seja encaminhada para um worker diferente, evitando concentração de carga em um único nó.

2.3 Controle de Falhas e Tolerância

A tolerância a falhas é um dos pilares de sistemas distribuídos, permitindo que o sistema continue funcional mesmo quando componentes falham. Para tanto, foram aplicados três mecanismos principais:

1. Heartbeat: cada worker envia periodicamente sinais de vida ao orquestrador, confirmando que está ativo.
2. Failover automático: caso o orquestrador principal falhe, o backup assume a função de coordenador.
3. Redistribuição de tarefas: se um worker deixar de responder, suas tarefas inacabadas são reatribuídas a outro nó ativo.

Trecho de código exemplificando o heartbeat:

```
3
4
5 public void enviarHeartbeat() {
6     while (true) {
7         try {
8             socket.send(new DatagramPacket("HEARTBEAT".getBytes(), 9, endereco, porta));
9             Thread.sleep(5000);
10        } catch (Exception e) {
11            System.out.println("Falha no envio do heartbeat: " + e.getMessage());
12        }
13    }
14 }
```

2.4 Relógios Lógicos de Lamport

Em sistemas distribuídos, a ausência de um relógio global consistente exige mecanismos de ordenação de eventos. O algoritmo de Lamport permite estabelecer uma relação de causalidade entre mensagens, atribuindo a cada evento um timestamp lógico.

No projeto, os relógios de Lamport foram utilizados para ordenar submissões de tarefas e atualizações de estado global, assegurando que as operações de redistribuição e failover ocorressem sem inconsistência.

2.5 Segurança e Autenticação

Embora a segurança não seja o foco principal do projeto, foi implementado um mecanismo de autenticação básica. Cada cliente deve realizar login com usuário e senha válidos para obter um token, que é exigido no momento de submissão de tarefas. Isso garante que apenas usuários autorizados possam interagir com o sistema.

Exemplo simplificado do processo:

```
4
5 public String autenticar(String usuario, String senha) {
6     if (usuarios.containsKey(usuario) && usuarios.get(usuario).equals(senha)) {
7         return gerarToken(usuario);
8     }
9     return null;
10 }
```

Esse modelo atende ao requisito mínimo de segurança estabelecido na atividade, mas pode ser expandido em trabalhos futuros para incluir criptografia, gerenciamento de sessão e protocolos mais robustos.

3. Arquitetura do Sistema

A arquitetura do sistema foi projetada para atender aos requisitos definidos no enunciado da atividade, contemplando a presença de um orquestrador principal, um orquestrador de backup, três workers e múltiplos clientes.

3.1 Componentes Principais

- **Cliente:** responsável por submeter tarefas e consultar o status das mesmas. A comunicação é realizada com o orquestrador principal via sockets TCP.
- **Orquestrador Principal:** atua como coordenador do sistema, recebendo tarefas, balanceando a carga entre os workers, monitorando falhas e garantindo a execução.
- **Orquestrador Secundário (Backup):** mantém cópia do estado global através de UDP multicast. Em caso de falha do orquestrador principal, assume automaticamente o controle.
- **Workers:** nós de processamento responsáveis por executar as tarefas atribuídas, enviando periodicamente mensagens de heartbeat para o orquestrador.

3.2 Comunicação

- **Cliente ↔ Orquestrador Principal:** realizada por TCP, garantindo confiabilidade na submissão das tarefas.
- **Orquestrador Principal ↔ Backup:** comunicação por UDP Multicast, permitindo sincronização eficiente do estado global.
- **Orquestrador Principal ↔ Workers:** troca de mensagens para atribuição de tarefas e recebimento de heartbeats.

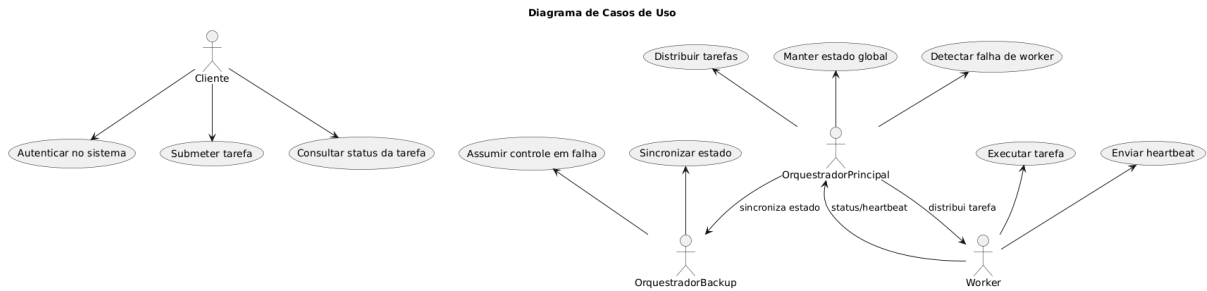
3.3 Diagramas UML

Diagrama de Casos de Uso: descreve as interações entre clientes, orquestrador e workers.

Diagrama de Sequência: representa os fluxos de submissão de tarefa, balanceamento, falha e failover.

Diagrama de Componentes: apresenta a organização estrutural do sistema.

Diagrama de Casos de Uso



(Observação: Zoom 175% para melhor visualização)

Diagrama de Sequência

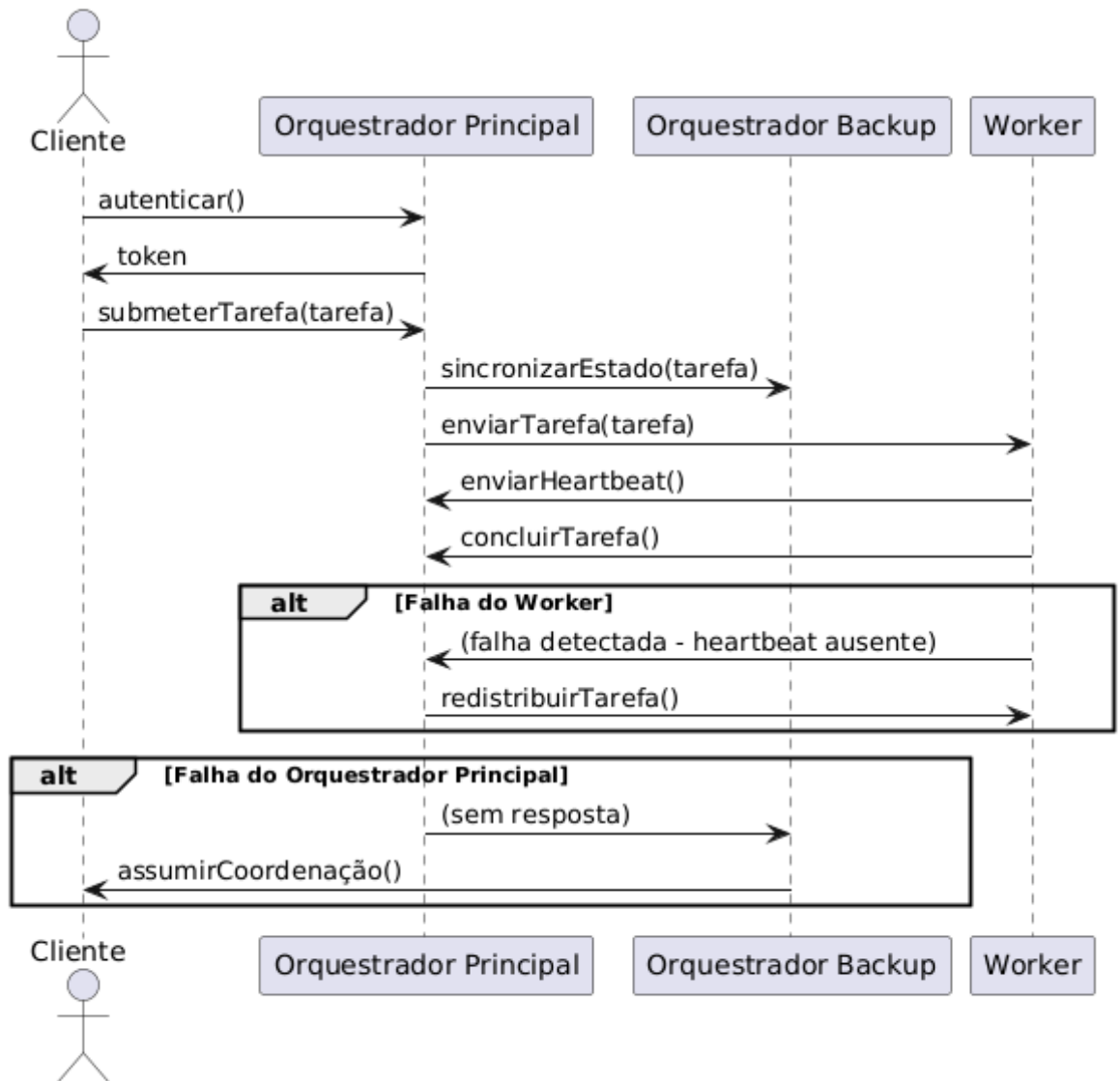
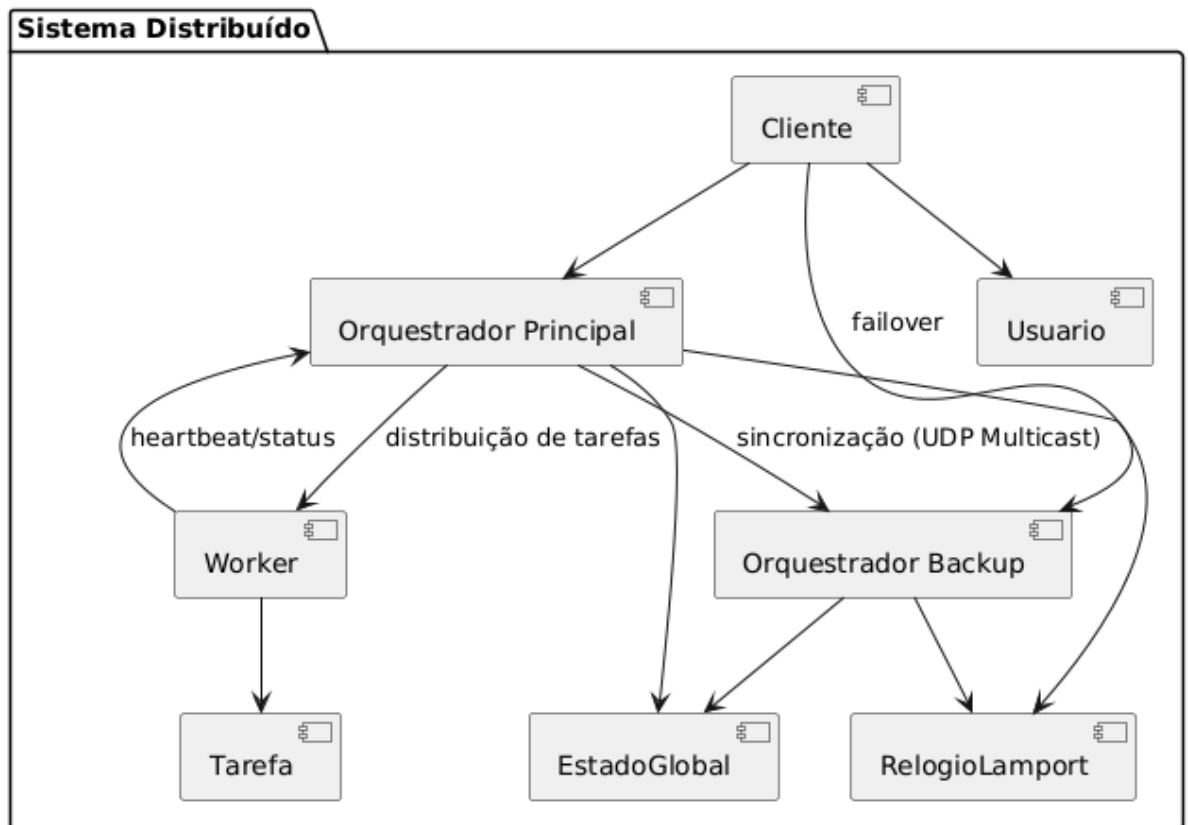


Diagrama de Componentes



4. Justificativa das Escolhas Técnicas

A decisão de utilizar **sockets TCP** para comunicação Cliente-Orquestrador se deu pela necessidade de confiabilidade na entrega das tarefas. O uso de **UDP Multicast** na sincronização entre orquestradores justifica-se pela simplicidade e baixo overhead, uma vez que a atualização de estado pode ser transmitida em broadcast para múltiplos receptores.

A política de balanceamento Round Robin foi escolhida por:

- Garantir simplicidade de implementação.
- Proporcionar distribuição uniforme em cenários com tarefas homogêneas.
- Possibilitar fácil detecção de gargalos.

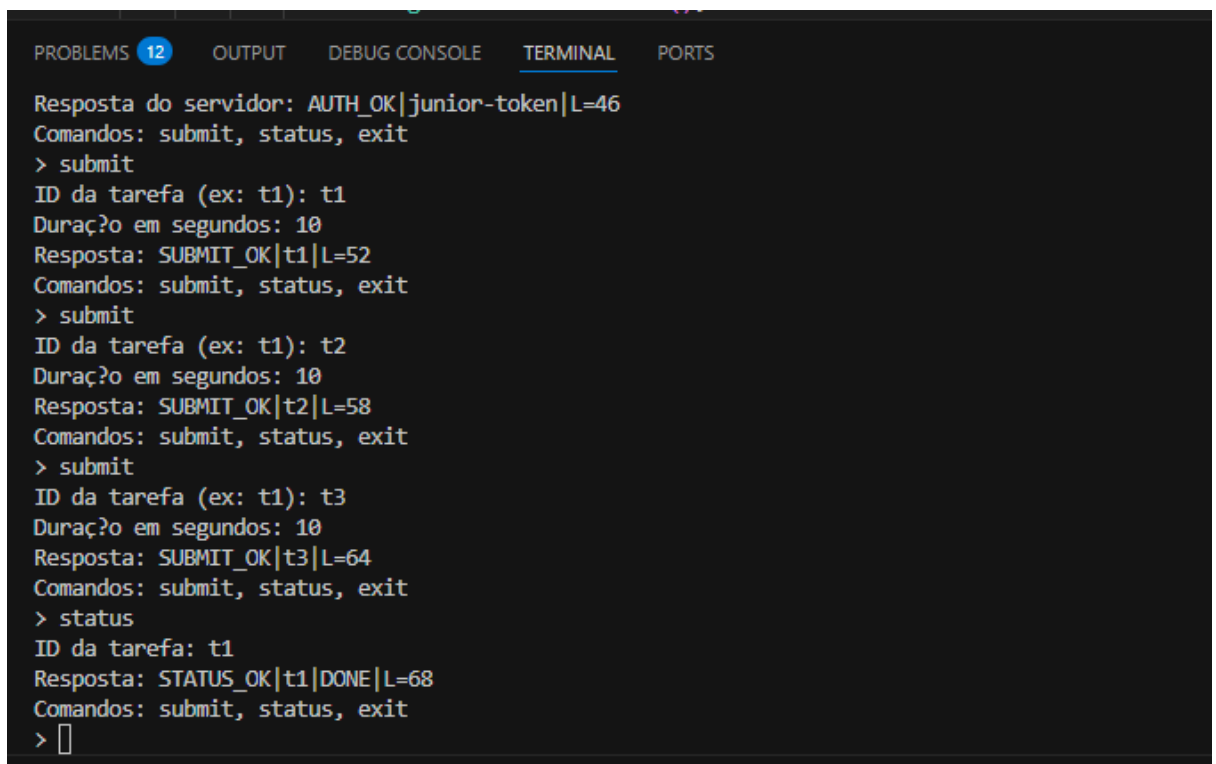
No entanto, reconhece-se que essa política pode não ser a mais eficiente em cenários heterogêneos de carga, o que abre espaço para melhorias.

5. Evidências de Execução

Durante os testes do sistema, foram registradas evidências de funcionamento que comprovam a conformidade com os requisitos.

- **Submissão de Tarefas:** logs mostram clientes autenticados enviando solicitações.
- **Distribuição Round Robin:** registros confirmam a rotação entre workers.
- **Heartbeats:** mensagens periódicas dos workers confirmam sua atividade.
- **Falhas Simuladas:** quando um worker deixa de responder, a tarefa é redistribuída.
- **Failover:** a interrupção do orquestrador principal resulta na assunção do backup.

Execuções terminais (1 orquestrador, 1 backup, 3 workers e 1 cliente):



```
PROBLEMS 12 OUTPUT DEBUG CONSOLE TERMINAL PORTS
Resposta do servidor: AUTH_OK|junior-token|L=46
Comandos: submit, status, exit
> submit
ID da tarefa (ex: t1): t1
Duração em segundos: 10
Resposta: SUBMIT_OK|t1|L=52
Comandos: submit, status, exit
> submit
ID da tarefa (ex: t1): t2
Duração em segundos: 10
Resposta: SUBMIT_OK|t2|L=58
Comandos: submit, status, exit
> submit
ID da tarefa (ex: t1): t3
Duração em segundos: 10
Resposta: SUBMIT_OK|t3|L=64
Comandos: submit, status, exit
> status
ID da tarefa: t1
Resposta: STATUS_OK|t1|DONE|L=68
Comandos: submit, status, exit
> 
```

```
[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
```

```
Backup] Estado recebido via multicast:
Backup] Estado recebido via multicast:
Backup] Estado recebido via multicast:
Backup] Estado recebido via multicast:
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
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:
Backup] Estado recebido via multicast:
Backup] Estado recebido via multicast:
Backup] Estado recebido via multicast:
```

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

6. Análise Crítica

O sistema desenvolvido atende aos principais requisitos propostos: autenticação básica, orquestração, balanceamento de carga, heartbeat, failover e uso de relógios lógicos.

Pontos fortes:

- Arquitetura modular e bem definida.
- Implementação clara dos mecanismos de tolerância a falhas.
- Uso de técnicas clássicas de sistemas distribuídos (Lamport, Round Robin, heartbeat).

Limitações:

- Balanceamento de carga simples, não adaptativo.
- Segurança restrita a autenticação básica, sem criptografia.
- Estado global armazenado em memória, sem persistência externa.

Possíveis Melhorias:

- Implementar balanceamento dinâmico baseado em métricas de carga real.
- Incluir autenticação avançada e criptografia TLS.
- Persistência do estado em banco de dados distribuído (ex.: Apache Cassandra).
- Adoção de containers (Docker) para simular ambiente real de cluster.

7. Conclusão

A plataforma distribuída de processamento colaborativo de tarefas desenvolvida demonstra, em caráter experimental, os conceitos fundamentais de sistemas distribuídos, como balanceamento, tolerância a falhas, replicação e autenticação.

O sistema alcançou seus objetivos principais ao permitir que clientes submetessem tarefas, que estas fossem distribuídas entre múltiplos workers, e que houvesse continuidade do serviço em caso de falhas. Apesar das limitações, a solução se mostra um protótipo funcional e consistente, servindo como base para implementações mais complexas.

8. Referências

- TANENBAUM, A. S.; VAN STEEN, M. *Sistemas Distribuídos: Princípios e Paradigmas*. 2. ed. São Paulo: Pearson, 2017.
- COULOURIS, G.; DOLLIMORE, J.; KINDBERG, T. *Distributed Systems: Concepts and Design*. 5. ed. Addison-Wesley, 2011.
- 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.

- Documentação oficial da linguagem Java. Disponível em: <https://docs.oracle.com/javase/>.
- Materiais fornecidos pelo docente da disciplina.

Link do Repositório:

<https://github.com/Hildemar0034/Plataforma-Distribu-da-de-Processamento-Colaborativo-de-Tarefas>