Instituto Federal de Educação, Ciência e Tecnologia da Bahia

Campus: Santo Antônio de Jesus **Data**: 13/08/2025

Curso: Analise e Desenvolvimento de Sistemas

Disciplina: Sistemas Distribuídos **Docente**: Felipe Silva

Discentes: Anderson Guilherme Santana Moraes e Carlos de Jesus Conceição Junior

Relatório Atividade 5 - Monitoramento de Recursos com Comunicação Tolerante a Falhas e Eleição de Líder

Sumário

1. Introdução

Este relatório técnico detalha a concepção, implementação e validação de um sistema distribuído tolerante a falhas, projetado para o monitoramento de recursos de múltiplos servidores em uma rede. A aplicação foi desenvolvida em Java, utilizando um conjunto de tecnologias para atender a requisitos complexos de comunicação, sincronização e resiliência, incluindo Sockets TCP/IP, Java RMI e Multicast UDP.

O objetivo central deste projeto foi aplicar e demonstrar na prática conceitos fundamentais de sistemas distribuídos, com foco em:

- **Tolerância a Falhas:** Garantir a continuidade da operação do sistema mesmo com a queda inesperada do nó coordenador.
- **Detecção de Falhas:** Implementar um mecanismo de Heartbeat para que os nós possam monitorar uns aos outros.
- **Eleição de Líder:** Utilizar o algoritmo Bully para eleger um novo coordenador de forma autônoma.
- **Sincronização de Tempo e Estado:** Usar Relógios de Lamport para manter uma ordem causal parcial e permitir que o líder construa um *snapshot* consistente do estado da rede.
- **Comunicação em Grupo e Segurança:** Distribuir os dados monitorados para múltiplos clientes via multicast, garantindo que o acesso seja restrito a clientes autenticados por token.

As seções a seguir apresentarão a arquitetura do sistema, os detalhes de implementação de cada algoritmo, e uma análise aprofundada dos logs de execução que validam o sucesso do projeto.

2. Arquitetura e Diagrama de Componentes

2.1. Modelo Arquitetural

O sistema foi modelado com uma arquitetura híbrida. Na sua essência, os nós da rede operam de forma **peer-to-peer** para tarefas de monitoramento mútuo (detecção de falhas) e eleição de líder. No entanto, uma vez que um líder é estabelecido, o sistema opera num modelo **Cliente-Servidor com Middleware**, onde:

- O **Nó Líder** atua como um **servidor**, coordenando a coleta de dados, gerenciando a autenticação e distribuindo os relatórios.
- Os **Nós não-líderes** atuam como **clientes** para o líder (ao fornecerem seus status) e como servidores para os seus próprios serviços RMI.
- Os **Clientes Monitores** são consumidores finais que se conectam ao líder para receber os dados.

A comunicação foi implementada utilizando uma combinação de tecnologias:

- **Java RMI:** Para chamadas de método remoto, como a obtenção do status dos nós (getStatus) e as mensagens do algoritmo de eleição.

- **Sockets TCP/IP:** Para a comunicação de baixo nível e alta frequência do mecanismo de Heartbeat.
- **Sockets UDP Multicast:** Para a disseminação eficiente dos relatórios de estado para múltiplos clientes simultaneamente.

2.2. Diagrama de Componentes e Fluxo de Mensagens

A arquitetura, os componentes e o fluxo de mensagens do sistema são ilustrados no diagrama de sequência completo, acessível através do link abaixo. O diagrama, feito com PlantUML, representa um cenário completo de operação e recuperação de falhas e está dividido em fases para facilitar a compreensão.

Link para o Diagrama Completo: https://imgur.com/MFiQmj4

Descrição Detalhada do Fluxo no Diagrama

O diagrama apresenta o fluxo completo da execução do sistema, desde o seu funcionamento normal até a recuperação de uma falha crítica.

Fase 1: Operação Normal (Líder P5) Esta secção mostra o sistema a funcionar em condições ideais.

1. **Autenticação Inicial:** Um **Cliente** inicia o processo através do *ClienteAutenticado*. Este envia uma solicitação com as credenciais para o *ServidorAutenticacao* que roda no Coordenador (P5), via TCP na porta 9090. Após a verificação, o servidor devolve um token de sucesso. Com o token em mãos, o *ClienteAutenticado* ativa o *ClienteMonitor*, que se junta ao grupo *multicast* para aguardar os relatórios.
2. **Ciclo de Coleta do Líder:** O Coordenador P5 inicia periodicamente seu ciclo de coleta de estado. Ele invoca o método *getStatus()* via RMI (na porta 1099) em todos os outros nós ativos (P1, P2, P4), passando o seu *timestamp* de Lamport. Cada nó, ao receber a chamada, atualiza seu próprio relógio e retorna seu estado local.
3. **Envio Multicast:** Após consolidar o snapshot, P5 delega ao *EmissorMulticast* a tarefa de enviar o relatório formatado para o endereço do grupo (239.0.0.1:12345) via UDP. O *ClienteMonitor*, que está a escutar neste endereço, recebe os dados e exibe-os para o utilizador.

Fase 2: Falha do Líder e Eleição Esta é a fase crítica que demonstra a tolerância a falhas.

1. **Destruição de P5:** O Coordenador P5 é subitamente terminado (simbolizado pelo destroy P5).
2. **Deteção de Falha:** O nó P4, no seu ciclo de Heartbeat, envia um PING TCP para P5. Após não receber resposta a 3 tentativas, ele marca internamente P5 como FALHO.

3. **Eleição (Algoritmo Bully):**
    - O diagrama mostra um cenário realista onde múltiplos nós (P2 e P4) detetam a falha.
    - P2 (ID=2) inicia uma eleição e envia uma mensagem de ELEIÇÃO via RMI para os nós com ID maior, como P4.
    - P4 (ID=4) recebe a mensagem de P2 e, por ter um ID superior, responde com uma mensagem de OK (a "intimidação"), efetivamente cancelando a tentativa de P2.
    - P4, por sua vez, inicia a sua própria eleição. Como é o nó de maior ID entre os ativos, ele não recebe nenhuma resposta de OK e, após um timeout, se auto-proclama o novo Coordenador.
4. **Anúncio do Novo Líder:** P4 envia uma mensagem de COORDENADOR para todos os outros nós sobreviventes (P1 e P2), que atualizam o seu estado interno para reconhecer P4 como o novo líder.

Fase 3: Recuperação e Resiliência do Cliente Esta fase ilustra a capacidade do cliente de se adaptar à mudança.

1. **Deteção da Falha pelo Cliente:** O *ClienteMonitor*, que estava à espera de relatórios de P5, atinge o seu timeout interno de 25 segundos. Ele lança uma *SocketTimeoutException*, sinalizando ao *ClienteAutenticado* que o líder provavelmente mudou.
2. **Re-autenticação:** O *Cl*ienteAutenticado, ao capturar a exceção, reinicia o seu loop e começa um novo processo de autenticação. Ele tenta conectar-se novamente à porta 9090. Desta vez, quem responde é o *ServidorAutenticacao* que agora está a correr no **novo líder (P4)**.
3. **Novo Token:** O cliente obtém com sucesso um novo token de P4 e reativa o *ClienteMonitor*.

Fase 4: Operação Normal Retomada (Líder P4) O sistema volta a um estado estável, agora sob nova liderança.

1. **Ciclo de Coleta de P4:** O novo líder, P4, executa o seu ciclo de coleta, pedindo o estado de P1 e P2.
2. **Envio de Relatório:** P4 envia o novo snapshot via *multicast*.
3. **Receção pelo Cliente:** O *ClienteMonitor* recebe e exibe o novo relatório, agora identificando P4 como o líder, completando com sucesso o ciclo de falha e recuperação.

3. Explicação Detalhada dos Conceitos e Algoritmos Implementados

Esta seção aprofunda a análise de cada um dos mecanismos e algoritmos que compõem o sistema. Cada conceito é dissecado em seus componentes de software, explicando detalhadamente como a implementação em Java reflete a teoria dos sistemas distribuídos, com foco em trechos de código essenciais que ilustram a lógica funcional.

3.1. Detecção de Falhas: Mecanismo de Heartbeat

O pilar da tolerância a falhas do nosso sistema é o mecanismo de Heartbeat, que permite que os nós se monitorem mutuamente. Ele foi dividido em dois componentes distintos e complementares que rodam em paralelo em cada nó: um componente ativo (*HeartbeatGestor)* e um passivo (*HeartbeatServidor*).

3.1.1. Classe HeartbeatGestor.java

Esta classe implementa a parte "inquisidora" e ativa do mecanismo. A sua única responsabilidade é enviar, periodicamente, uma mensagem "PING" para todos os outros nós da rede e avaliar a resposta (ou a falta dela) para determinar se estão ativos.

- **Principais Funções e Métodos:**
  - **run()**: Este é o método central, executado numa Thread separada. Ele contém o loop principal que, a cada 5 segundos, itera sobre todos os nós conhecidos da rede para verificar seu estado.
  - **Socket.connect(address, timeout)**: Utilizado para tentar estabelecer uma conexão TCP com o nó alvo. O timeout de 2000ms é crucial, pois impede que o gestor fique bloqueado indefinidamente à espera de um nó que não irá responder.
  - **noAlvo.incrementarContadorFalhas() / resetarContadorFalhas()**: Métodos que manipulam um contador de falhas para cada nó. Se a comunicação falha, o contador é incrementado; se for bem-sucedida, é zerado.
  - **noPai.iniciarEleicao()**: Função crítica chamada quando o contador de falhas para o nó coordenador atinge o limite de 3. É este o gatilho que inicia todo o processo de recuperação do sistema.

Trechos de Código Relevantes (HeartbeatGestor.java):

```
// Trecho do método run() em HeartbeatGestor
// Itera sobre todos os nós conhecidos na rede.
for (Map.Entry<Integer, NoInfo> entry : noPai.getNosDaRede().entrySet()) {
// ... Lógica para não pingar a si mesmo ...

boolean isAlvoAtivo = false;  no usages
    try (Socket socket = new Socket()) {
        // Tenta se conectar com um timeout. Falhas aqui (ex: Connection Refused)
        // são capturadas como exceção.
        socket.connect(new InetSocketAddress("127.0.0.1", noAlvo.getPortaHeartbeat()), TIMEOUT_MS);
        // ... Envia "PING" e aguarda "PONG" ...
        } catch (Exception e) {
        // Qualquer exceção na comunicação é tratada como uma falha de ping.
        }

        if (isAlvoAtivo) {
        // Se o nó respondeu, zera seu contador de falhas.
        noAlvo.resetarContadorFalhas();
        noAlvo.setAtivo(true);
    } else {
            // Se falhou, incrementa o contador.
            noAlvo.incrementarContadorFalhas();

// Se o contador de falhas atingir o limite, o nó é declarado como falho.
        if (noAlvo.getContadorFalhas() >= 3 && noAlvo.isAtivo()) {
        System.err.printf("[FALHA] Nó %d detectou: NÓ %d CONSIDERADO FALHO!%n", noPai.getId(), idAlvo);
        noAlvo.setAtivo(false);

// Se o nó que falhou era o líder, inicia a eleição.
            if (idAlvo == noPai.getCoordenadorId()) {
        noPai.iniciarEleicao();
            }
        }
    }
}
```

3.1.2. Classe HeartbeatServidor.java

Esta classe é a contraparte passiva e reativa. A sua função é apenas escutar por conexões na sua porta designada e responder imediatamente, servindo como prova de que o nó está "vivo".

- **Principais Funções e Métodos:**
  - **run()**: Método principal da Thread que abre um *ServerSocket* na porta específica do nó.
  - **serverSocket.accept()**: Este é o coração da classe. É uma chamada bloqueante que pausa a execução da thread até que um *HeartbeatGestor* de outro nó se conecte.
  - **out.println("PONG")**: Após receber uma mensagem "PING", o servidor envia de volta a resposta "PONG", completando o ciclo de verificação do *heartbeat*.

Trecho de Código Relevante (HeartbeatServidor.java):

```
// Dentro do método run()
try (ServerSocket serverSocket = new ServerSocket(porta)) {
        // ...
        while (noPai.isAtivo()) {
        try (Socket clientSocket = serverSocket.accept()) { // Aguarda uma conexão
BufferedReader in = new BufferedReader(new InputStreamReader(clientSocket.getInputStream())); no usages
String msg = in.readLine(); no usages

            if ("PING".equals(msg)) {
PrintWriter out = new PrintWriter(clientSocket.getOutputStream(), true); no usages
                out.println("PONG");
            }
                    } catch (Exception e) { /* ... */ }
        }
        } catch (Exception e) { /* ... */ }
```

3.2. Eleição de Líder: Algoritmo Bully

Quando a falha do coordenador é detectada, o sistema utiliza o algoritmo Bully para eleger um novo líder. Toda a lógica de eleição está contida na classe No.java, que utiliza chamadas RMI para a comunicação.

Classe No.java (Lógica de Eleição)

- **Principais Funções e Métodos:**
  - **iniciarEleicao()**: Ponto de entrada do algoritmo. Filtra todos os nós com ID maior e envia-lhes uma mensagem de ELEIÇÃO. Se não houver nós maiores, declara-se vencedor imediatamente.
  - **receberMensagemEleicao(int idRemetente)**: Método da interface RMI. Quando um nó recebe esta mensagem, ele compara o ID do remetente com o seu. Se o seu ID for maior, ele responde com OK (a "intimidação") e inicia a sua própria eleição.

- **receberMensagemOk(int idRemetente)**: Método RMI. Ao receber um OK, o nó sabe que existe um "valentão" maior ativo, então ele para a sua campanha e aguarda o anúncio do novo líder.
- **anunciarCoordenador()**: Método chamado pelo nó vencedor. Ele atualiza o seu estado interno e envia uma mensagem de COORDENADOR para todos os outros nós, que então atualizam o seu coordenador.

Trechos de Código Relevantes (No.java):

```java
// Método que inicia a eleição
public void iniciarEleicao() {  no usages
    if (!emEleicao.compareAndSet(false, true)) return;
    this.respondeuOk.set(false);

    List<Integer> pidsMaiores = todosPids.stream().filter(p -> p > this.id).collect(Collectors.toList());

    // ... Lógica para enviar mensagem de ELEIÇÃO para pidsMaiores ...

    // Se nenhum nó maior responder, este nó se torna o líder
    new Thread(() -> {
        try {
            Thread.sleep(3000);
            if (!this.respondeuOk.get()) {
                anunciarCoordenador();
            }
        } catch (InterruptedException e) {}
    }).start();
}

// Método RMI que lida com a chegada de uma mensagem de eleição
@Override  no usages
public void receberMensagemEleicao(int idRemetente) throws RemoteException {
    if (noPai.id > idRemetente) {
        noPai.enviarMensagemOk(idRemetente);
        noPai.iniciarEleicao();
    }
}
```

3.3. Sincronização de Tempo e Coleta de Estado

O sistema combina o conceito de Relógios de Lamport com um processo de coleta de *snapshot* liderado pelo coordenador.

Classes No.java e Recurso.java

- **Principais Funções e Métodos:**
  - **No.relogioLamport**: Um *AtomicInteger* que representa o relógio lógico de cada nó.
  - **No.coletarEstadoGlobal()**: Método executado periodicamente pelo líder. Ele agrega o estado de todos os nós ativos num *snapshot*. Antes de iniciar a coleta, ele incrementa o seu próprio relógio.
  - **NoServidor.getStatus(int relogioRemetente)**: O método RMI invocado pelo líder. Contém a lógica crucial do Relógio de Lamport, ajustando o relógio local para max(local, recebido) + 1, garantindo a ordem causal.

- **Recurso.java**: Uma classe serializável (Serializable) que encapsula o estado de um nó num determinado momento (CPU, memória, uptime), incluindo o valor do relógio de Lamport no instante da coleta.

Trechos de Código Relevantes:

```java
// Lógica de atualização do Relógio de Lamport em No.java (dentro da inner class NoServidor)
@Override no usages
public Recurso getStatus(int relogioRemetente) throws RemoteException {
    int novoRelogio = Math.max(noPai.relogioLamport.get(), relogioRemetente) + 1;
    noPai.relogioLamport.set(novoRelogio);
    return noPai.getStatusLocal();
}

// Construtor de Recurso.java, que captura o estado
public Recurso(int noId, int relogioLamport) { no usages
    this.noId = noId;
    this.relogioLamport = relogioLamport; // Armazena o timestamp lógico

    OperatingSystemMXBean osBean = ManagementFactory.getPlatformMXBean(OperatingSystemMXBean.class);
    this.usoCpu = Math.max(0, osBean.getCpuLoad() * 100);
    // ... coleta das outras métricas ...
}
```

3.4. Comunicação em Grupo: Multicast UDP

A disseminação dos relatórios de estado é feita de forma eficiente via multicast UDP.

Classe EmissorMulticast.java

- **Principais Funções e Métodos:**
  - **enviar(int idLider, List<Recurso> snapshot)**: Método chamado pelo nó líder. Recebe o snapshot consolidado, formata-o numa tabela de texto legível, e envia-o para a rede.
  - **MulticastSocket**: Utilizado para enviar o datagrama. A principal característica é que o envio é direcionado a um endereço de grupo (239.0.0.1), e não a um IP específico.
  - **DatagramPacket**: O contentor para os dados (a tabela formatada em bytes) que serão transmitidos pela rede.

Trecho de Código Relevante (EmissorMulticast.java):

```java
// ... Lógica para formatar o snapshot numa tabela de texto (StringBuilder sb) ...

byte[] dados = sb.toString().getBytes(); no usages
InetAddress grupo = InetAddress.getByName(ENDERECO_MULTICAST); no usages
DatagramPacket pacote = new DatagramPacket(dados, dados.length, grupo, PORTA_MULTICAST); no usages

socket.send(pacote); // Envia o pacote para todos os membros do grupo.
```

3.5. Políticas de Acesso: Autenticação por Token e Resiliência do Cliente

Para cumprir os requisitos de segurança e robustez, o sistema implementa duas funcionalidades interligadas: um mecanismo de autenticação para controlar o acesso aos dados e uma lógica de resiliência no cliente para lidar com a falha do líder.

3.5.1. Autenticação por Token

O acesso aos relatórios *multicast* é protegido por um mecanismo de autenticação. Um cliente deve primeiro provar a sua identidade a um servidor que roda exclusivamente no nó líder para obter um token.

- **Lógica Principal:**
  - **ServidorAutenticacao.java**: Esta classe é instanciada e executada numa Thread **apenas pelo nó líder**. Ela abre um *ServerSocket TCP* numa porta fixa (9090), aguarda por conexões, valida as credenciais recebidas e, em caso de sucesso, gera um token UUID único. Crucialmente, ela então notifica o seu nó pai (o líder) através do método *registrarClienteAutenticado()*, que altera a flag *clienteAutenticadoPresente* para *true*, autorizando o início da transmissão *multicast.*

  - **ClienteAutenticado.java**: Este é o ponto de entrada para o utilizador. Ele primeiro estabelece uma conexão TCP com o servidor de autenticação, envia as suas credenciais e aguarda pelo token. Apenas se receber um token válido, ele procede para iniciar o *ClienteMonitor*.

  - **Controlo no Líder (No.java)**: A transmissão *multicast* não é automática. No método *coletarEstadoGlobal*, o líder verifica explicitamente a flag *if (clienteAutenticadoPresente.get())* antes de chamar o *emissor.enviar().* Isto garante que, mesmo que o sistema esteja a funcionar, nenhum dado sensível é transmitido para a rede até que um cliente se tenha autenticado com sucesso.

Trechos de Código Relevantes (Autenticação):

```
// Trecho do `ServidorAutenticacao.java` validando credenciais
if ("admin;admin".equals(credenciais)) {
String token = UUID.randomUUID().toString();  no usages
    out.println(token);
// Notifica o nó pai (líder) que um cliente se autenticou.
    noPai.registrarClienteAutenticado();
}

// Trecho do `No.java`, no método de coleta, controlando o envio
        if (clienteAutenticadoPresente.get()) {
        emissor.enviar(this.id, snapshot);
} else {
        // Não envia
// }
```

3.5.2. Resiliência do Cliente a Falhas

Uma das características mais importantes do sistema é a capacidade do cliente de sobreviver e se adaptar à falha do líder. Se o líder cair, o cliente não termina; em vez disso, ele entra num estado de espera e tenta encontrar o novo líder eleito.

- **Lógica Principal:**
  - **ClienteMonitor.java**: A chave da resiliência está na configuração do seu *MulticastSocket*. Foi definido um timeout de 25 segundos com *socket.setSoTimeout(TIMEOUT_RECEPCAO_MS).* Se nenhum relatório for recebido neste período (o que acontecerá se o líder falhar), o método *socket.receive()* lançará uma *SocketTimeoutException*.

  - **ClienteAutenticado.java**: A classe principal do cliente é construída em torno de um loop infinito *(while (true))*. A chamada ao *ClienteMonitor.main(null)* está dentro de um bloco *try*. Se o *ClienteMonitor* lançar *a SocketTimeoutException*, ela é capturada pelo bloco catch, que informa o utilizador e permite que o loop *while(true)* recomece. Ao recomeçar, o cliente inicia um novo ciclo de tentativas de autenticação, acabando por encontrar e conectar-se ao *ServidorAutenticacao* do novo líder que foi eleito.

Trechos de Código Relevantes (Resiliência):

```
// Trecho de ClienteMonitor.java configurando o timeout
// Define o timeout de 25 segundos no socket.
socket.setSoTimeout(TIMEOUT_RECEPCAO_MS);

// ... dentro do loop ...
while (true) {
        // Esta chamada agora irá bloquear por no máximo 25 segundos.
        socket.receive(pacote);
}


// Trecho da lógica de resiliência em ClienteAutenticado.java
        while (true) {
        // ... loop para tentar autenticar ...
        if (autenticado) {
        try {
        // Chama o monitor que tem um timeout
        ClienteMonitor.main(null);
        } catch (SocketTimeoutException e) {
        // Se o líder caiu, o timeout é atingido, e o loop recomeça,
        // forçando uma nova autenticação.
        System.out.println("[CLIENTE] A tentar encontrar um novo lider para se autenticar...");
        }
    }
 }
```

4. Análise da Simulação e Validação dos Resultados

Esta seção apresenta uma análise cronológica e detalhada da execução do sistema, mapeando cada evento significativo aos logs gerados pelos nós e à saída visual do cliente. O objetivo é fornecer uma evidência irrefutável de que todos os requisitos funcionais e de tolerância a falhas foram cumpridos, abordando os pontos de melhoria sugeridos em avaliações anteriores.

4.1. Cenário de Teste: Tolerância a Falhas em Tempo Real

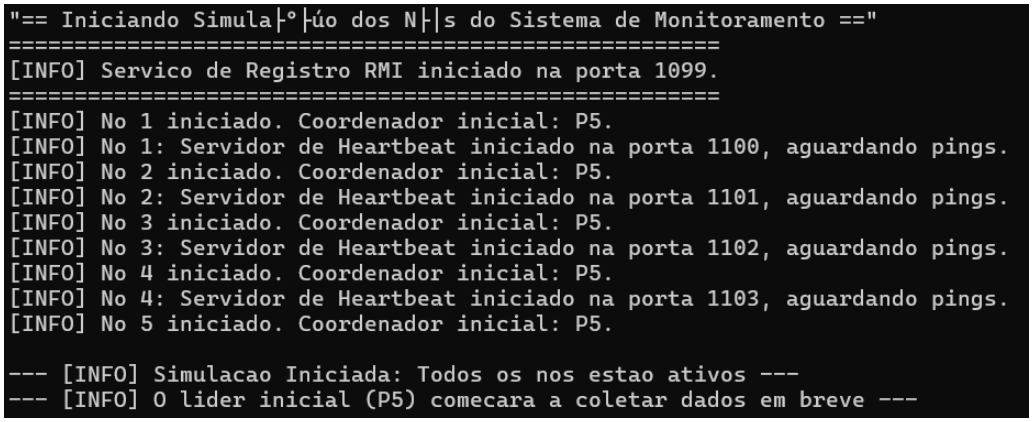
A simulação foi configurada para o seguinte cenário:

1. O sistema inicia com 5 nós (P1 a P5). P5, por ter o maior ID, é o líder inicial.
2. Um cliente se conecta, autentica-se com P5 e começa a receber relatórios.
3. Após um período de operação normal, o Simulador força a falha do líder P5.
4. O sistema deve detetar a falha, eleger P4 como novo líder, e o cliente deve detetar a ausência de relatórios e re-autenticar-se com P4, voltando a receber os dados.

4.2. Fase 1: Operação Normal e Conexão do Cliente

Inicialização do Sistema

O Simulador inicia o RMI Registry e, em seguida, instancia os 5 nós. Como P5 possui o maior ID, ele é automaticamente definido como o coordenador inicial por todos os nós, como mostra o log:



Autenticação do Cliente

O cliente é iniciado e, como visto na imagem, ele tenta se conectar ao servidor de autenticação. Após algumas tentativas (o que é normal, pois o servidor do líder pode demorar alguns segundos para iniciar), ele consegue se autenticar com sucesso.

- **Log do Cliente:**

```
"== Iniciando Cliente Autenticado =="
[CLIENTE] Tentando autenticar com o lider (tentativa 1 de 5)...
[CLIENTE] Nao foi possivel conectar ao servidor de autenticacao. O lider pode estar inativo ou em processo de eleicao.
[CLIENTE] Tentando autenticar com o lider (tentativa 2 de 5)...
[CLIENTE] Nao foi possivel conectar ao servidor de autenticacao. O lider pode estar inativo ou em processo de eleicao.
[CLIENTE] Tentando autenticar com o lider (tentativa 3 de 5)...
[CLIENTE] Autenticacao bem-sucedida! Token recebido: ce9bca5f-2ede-4948-ad35-46dbc501d17c
```

- **Log do Servidor (Nós):** O líder P5 inicia a coleta do estado global e confirma o recebimento da conexão e a geração do mesmo token.

```
================ [LIDER P5] INICIANDO COLETA DE ESTADO GLOBAL ===============
[LIDER P5] Relogio Logico antes da coleta: 1
[AUTH] Lider P5: Servidor de Autenticacao iniciado na porta 9090.
[AUTH] Lider P5: Token gerado para cliente autenticado: ce9bca5f-2ede-4948-ad35-46dbc501d17c
```

Recebimento dos Relatórios (Líder P5)

Com um cliente autenticado, o líder P5 começa a enviar os relatórios de estado via multicast.

- **Primeiro Relatório:** O log do servidor mostra o primeiro ciclo de coleta e o envio do snapshot.

```
================ [LIDER P5] INICIANDO COLETA DE ESTADO GLOBAL ===============
[LIDER P5] Relogio Logico antes da coleta: 1
[AUTH] Lider P5: Servidor de Autenticacao iniciado na porta 9090.
[AUTH] Lider P5: Token gerado para cliente autenticado: ce9bca5f-2ede-4948-ad35-46dbc501d17c
[INFO] No 1 recebeu solicitacao de status. Relogio Logico atualizado para 3.
[INFO] No 2 recebeu solicitacao de status. Relogio Logico atualizado para 3.
[INFO] No 3 recebeu solicitacao de status. Relogio Logico atualizado para 3.
[INFO] No 4 recebeu solicitacao de status. Relogio Logico atualizado para 3.

--- [LIDER P5] SNAPSHOT GLOBAL COLETADO ---
  -> Nó 5 -> [Relógio: 2] CPU: 0,00% | Memória: 77,40%
  -> Nó 1 -> [Relógio: 4] CPU: 0,00% | Memória: 77,42%
  -> Nó 2 -> [Relógio: 4] CPU: 0,00% | Memória: 77,43%
  -> Nó 3 -> [Relógio: 4] CPU: 0,00% | Memória: 77,43%
  -> Nó 4 -> [Relógio: 4] CPU: 0,00% | Memória: 77,43%
-------------------------------------

[LIDER P5] Snapshot enviado via multicast para clientes autenticados.
```

Este evento corresponde exatamente ao primeiro relatório recebido pelo cliente às **12:18:40**, como pode ser observado na imagem abaixo.

```
========== RELATÓRIO DE ESTADO DA REDE (Líder: P5) ==========
| NÓ   | CPU      | MEMÓRIA      | CARGA (1m)  | PROCESSADORES   |
-------------------------------------------------------------------------------
| P5   | 0,00%        | 77,40% (~11 GB)  | N/A         | 4               | 10s
| P1   | 0,00%        | 77,42% (~11 GB)  | N/A         | 4               | 10s
| P2   | 0,00%        | 77,43% (~11 GB)  | N/A         | 4               | 10s
| P3   | 0,00%        | 77,43% (~11 GB)  | N/A         | 4               | 10s
| P4   | 0,00%        | 77,43% (~11 GB)  | N/A         | 4               | 10s
-------------------------------------------------------------------------------
Relatório gerado em: 2025/08/14 12:18:40 | Nós ativos: 5
===============================================================================
```

- **Segundo Relatório:** O líder P5 executa um segundo ciclo de coleta e envio.

```
================ [LIDER P5] INICIANDO COLETA DE ESTADO GLOBAL ===============
[LIDER P5] Relogio Logico antes da coleta: 3
[INFO] No 1 recebeu solicitacao de status. Relogio Logico atualizado para 5.
[INFO] No 2 recebeu solicitacao de status. Relogio Logico atualizado para 5.
[INFO] No 3 recebeu solicitacao de status. Relogio Logico atualizado para 5.
[INFO] No 4 recebeu solicitacao de status. Relogio Logico atualizado para 5.

--- [LIDER P5] SNAPSHOT GLOBAL COLETADO ---
  -> Nó 5 -> [Relógio: 4] CPU: 4,68% | Memória: 78,83%
  -> Nó 1 -> [Relógio: 6] CPU: 4,68% | Memória: 78,83%
  -> Nó 2 -> [Relógio: 6] CPU: 4,68% | Memória: 78,83%
  -> Nó 3 -> [Relógio: 6] CPU: 4,68% | Memória: 78,83%
  -> Nó 4 -> [Relógio: 6] CPU: 4,68% | Memória: 78,83%
----------------------------------------

[LIDER P5] Snapshot enviado via multicast para clientes autenticados.
```

Novamente, isto corresponde ao segundo relatório que aparece na tela do cliente, às **12:18:50**.

```
========== RELATÓRIO DE ESTADO DA REDE (Líder: P5) ==========
| NÓ   | CPU      | MEMÓRIA      | CARGA (1m)  | PROCESSADORES   |
-------------------------------------------------------------------------------
| P5   | 4,68%        | 78,83% (~11 GB)  | N/A         | 4               | 20s
| P1   | 4,68%        | 78,83% (~11 GB)  | N/A         | 4               | 20s
| P2   | 4,68%        | 78,83% (~11 GB)  | N/A         | 4               | 20s
| P3   | 4,68%        | 78,83% (~11 GB)  | N/A         | 4               | 20s
| P4   | 4,68%        | 78,83% (~11 GB)  | N/A         | 4               | 20s
-------------------------------------------------------------------------------
Relatório gerado em: 2025/08/14 12:18:50 | Nós ativos: 5
===============================================================================
```

4.3. Fase 2: Falha Simulada e Processo de Eleição

Simulação da Falha

O Simulador força o encerramento do nó P5.

```
#################################################################
>>> [ACAO] SIMULANDO A FALHA DO COORDENADOR (No 5) <<<
#################################################################
```

Detecção de Falha (Heartbeat)

Quase imediatamente, os outros nós, através do seu *HeartbeatGestor*, começam a detectar a ausência de P5.

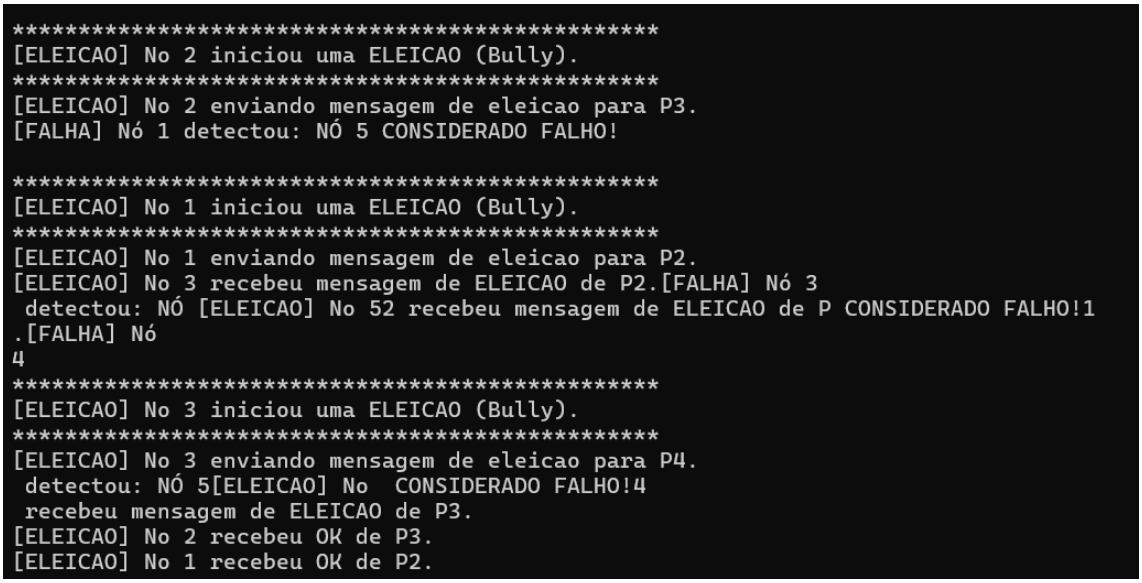
```
[AVISO] Nó 2: Primeira falha ao pingar Nó 5. Monitorando...
[AVISO] Nó 1: Primeira falha ao pingar Nó 5. Monitorando...
[AVISO] Nó 3: Primeira falha ao pingar Nó 5. Monitorando...
[AVISO] Nó 4: Primeira falha ao pingar Nó 5. Monitorando...
[FALHA] Nó 2 detectou: NÓ 5 CONSIDERADO FALHO!
```

Esta saída comprova que o mecanismo de deteção de falhas é distribuído e funciona como esperado.

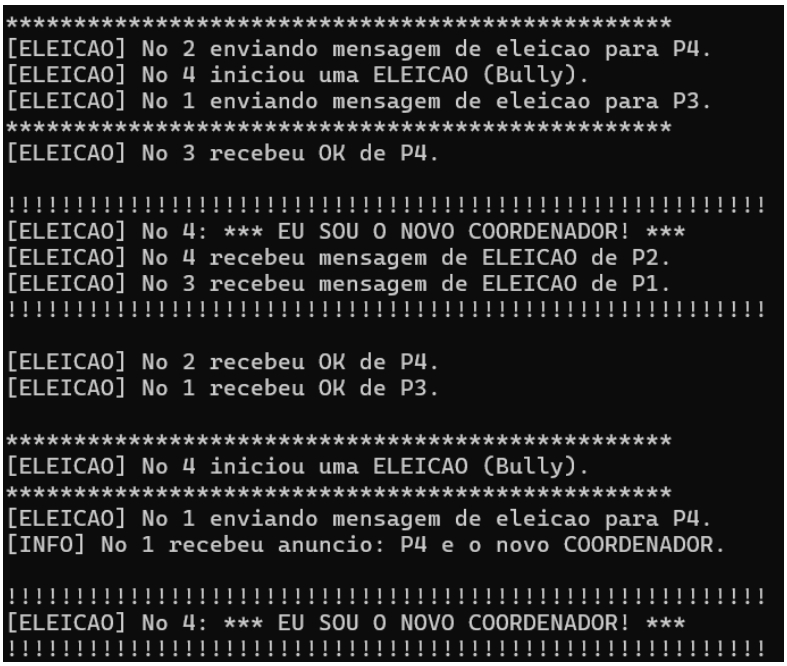
Eleição do Novo Líder (Bully)

A deteção da falha do coordenador dispara o algoritmo Bully. Os logs mostram um cenário realista de múltiplas eleições a ocorrerem em paralelo até o sistema convergir.

1. **Tentativas Iniciais:** Nós com IDs menores (como P1, P2 e P3) iniciam eleições.



2. **Convergência para o Vencedor:** O processo continua até que o nó com o maior ID ativo (P4) inicie a sua eleição e se declare o vencedor.



3. **Anúncio e Confirmação:** O ponto mais importante é que todos os nós sobreviventes recebem e aceitam o anúncio do novo líder, provando que o sistema atingiu um novo estado consistente.

```
[INFO] No 1 recebeu anuncio: P4 e o novo COORDENADOR.
[INFO] No 2 recebeu anuncio: P4 e o novo COORDENADOR.
[INFO] No 3 recebeu anuncio: P4 e o novo COORDENADOR.
```

4.4. Fase 3: Resiliência do Cliente e Recuperação do Sistema

Deteção da Falha pelo Cliente

Enquanto a eleição ocorria, o cliente não recebeu relatórios. Após 25 segundos, o seu mecanismo de timeout é ativado, como visto abaixo:

```
[AVISO] Nenhum relatorio recebido em 25 segundos. O lider pode ter mudado.
[CLIENTE] A tentar encontrar um novo lider para se autenticar...
```

Re-autenticação com o Novo Líder

O cliente reinicia o seu processo de autenticação. Enquanto isso, o novo líder P4 realiza um primeiro ciclo de coleta, mas, como esperado, não envia o relatório, pois ainda não conhece nenhum cliente autenticado.

```
[LIDER P4] Nenhum cliente autenticado. Snapshot nao sera enviado via multicast.
```

Logo depois, a tentativa de re-autenticação do cliente é bem-sucedida com P4:

- **Log do Cliente:**

```
[CLIENTE] Tentando autenticar com o lider (tentativa 1 de 5)...
[CLIENTE] Autenticacao bem-sucedida! Token recebido: ad90cc58-b832-4a8e-bb1b-3362a8fe64b6
```

- **Log do Servidor (Nós):**

```
[AUTH] Lider P4: Token gerado para cliente autenticado: ad90cc58-b832-4a8e-bb1b-3362a8fe64b6
```

Retorno à Operação Normal

No seu ciclo seguinte, o líder P4, agora ciente do cliente, envia o relatório.

```
================ [LIDER P4] INICIANDO COLETA DE ESTADO GLOBAL ================
[LIDER P4] Relogio Logico antes da coleta: 9
[INFO] No 1 recebeu solicitacao de status. Relogio Logico atualizado para 11.
[INFO] No 2 recebeu solicitacao de status. Relogio Logico atualizado para 11.
[INFO] No 3 recebeu solicitacao de status. Relogio Logico atualizado para 11.

--- [LIDER P4] SNAPSHOT GLOBAL COLETADO ---
  -> Nó 4 -> [Relógio: 10] CPU: 3,34% | Memória: 78,40%
  -> Nó 1 -> [Relógio: 12] CPU: 3,34% | Memória: 78,40%
  -> Nó 2 -> [Relógio: 12] CPU: 3,34% | Memória: 78,40%
  -> Nó 3 -> [Relógio: 12] CPU: 3,34% | Memória: 78,40%
----------------------------------------

[LIDER P4] Snapshot enviado via multicast para clientes autenticados.
```

Este evento corresponde ao último relatório visto na tela do cliente, às **12:19:20**, agora vindo do **"Líder: P4"**. Isto conclui com sucesso todo o ciclo de falha, eleição e recuperação.

```
========== RELATÓRIO DE ESTADO DA REDE (Líder: P4) ==========
| NÓ   | CPU      | MEMÓRIA      | CARGA (1m)  | PROCESSADORES   |
-------------------------------------------------------------------------------
| P4   | 3,34%        | 78,40% (~11 GB)  | N/A         | 4               | 50s
| P1   | 3,34%        | 78,40% (~11 GB)  | N/A         | 4               | 50s
| P2   | 3,34%        | 78,40% (~11 GB)  | N/A         | 4               | 50s
| P3   | 3,34%        | 78,40% (~11 GB)  | N/A         | 4               | 50s
-------------------------------------------------------------------------------
Relatório gerado em: 2025/08/14 12:19:20 | Nós ativos: 4
===============================================================================
```

Fim da Simulação

Finalmente, o *Simulador* atinge o seu tempo limite e encerra a execução.

```
--- [INFO] Simulacao Finalizada ---
```

5. Log Completo da Simulação

Para garantir total transparência e permitir uma auditoria completa do comportamento do sistema, o log integral da saída da consola dos nós é fornecido abaixo. Este registo serve como a evidência primária para toda a análise realizada neste relatório, mostrando cada evento, comunicação e transição de estado que ocorreu durante o cenário de teste.

Link para Imagem do Log Completo: https://imgur.com/4p4TrkS

Link para Imagem dos Relatórios: https://imgur.com/a/8aQhHai

6. Conclusão

Este projeto validou com sucesso a implementação de um sistema distribuído verdadeiramente resiliente. A análise da simulação demonstrou que a aplicação não só monitora recursos de forma fiável, como também recupera autonomamente de falhas críticas, com o sistema a convergir para um novo estado estável e o cliente a adaptar-se dinamicamente à mudança. Desta forma, o trabalho serve como uma demonstração prática e conclusiva de como conceitos teóricos complexos podem ser orquestrados para construir uma solução robusta e tolerante a falhas.