Instituto Federal de Educação, Ciência e Tecnologia da Bahia

Campus: Santo Antônio de Jesus **Data**: 07/09/2025

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 8

Sumário

1. Introdução

Este relatório técnico detalha a implementação de um serviço de mensagens distribuídas, concebido para operar numa rede peer-to-peer (P2P) sem uma autoridade central. O projeto, desenvolvido em Java, foca-se na aplicação prática de conceitos fundamentais de sistemas distribuídos, como a replicação de dados para alta disponibilidade, a garantia de consistência eventual, a tolerância a falhas e um mecanismo de autenticação para controle de acesso.

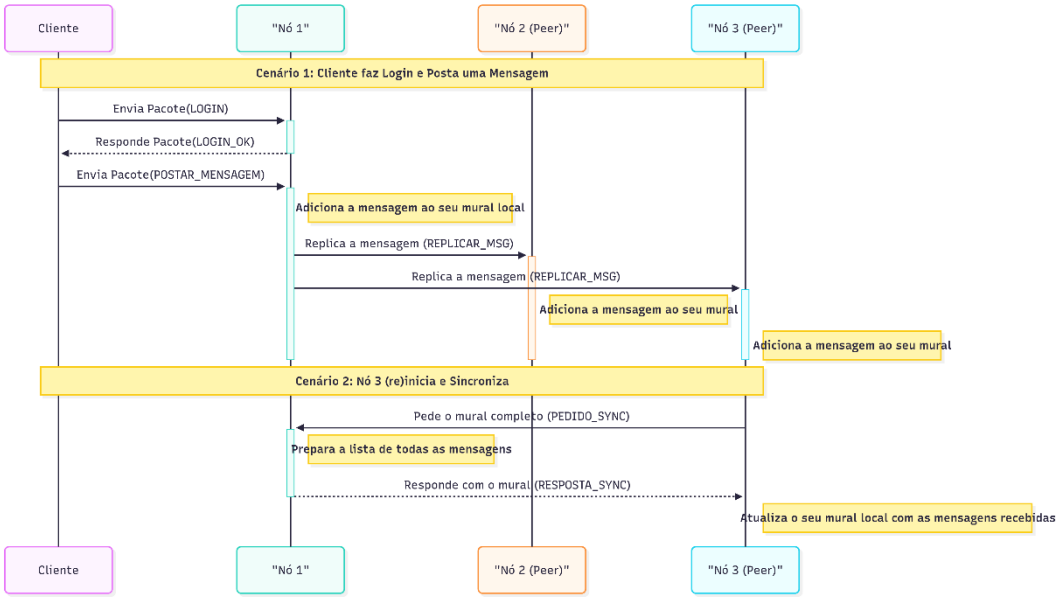
O sistema consiste em múltiplos nós autônomos, onde cada um mantém uma cópia local de um mural de mensagens compartilhado. As interações dos utilizadores, como postar novas mensagens, são replicadas de forma assíncrona pela rede. A robustez do sistema é demonstrada através de uma simulação de falha de um dos nós, seguida pela sua capacidade de se reconciliar com o estado da rede ao reiniciar, garantindo que nenhuma informação seja permanentemente perdida, graças a um mecanismo de persistência de estado em disco que sobrevive a reinicializações completas do sistema.

2. Arquitetura e Diagrama do Sistema

O sistema foi modelado com uma arquitetura P2P, composta por três nós que se comunicam diretamente via sockets TCP. Cada nó possui uma dupla funcionalidade: atua como um servidor, escutando por conexões de clientes e de outros nós, e como um cliente, ao propagar mensagens (replicação) ou ao solicitar o estado da rede (reconciliação).

A comunicação é padronizada através de um protocolo simples baseado na troca de objetos Pacote, que definem a intenção da mensagem (e.g., LOGIN, POSTAR_MENSAGEM, REPLICAR_MSG).

O diagrama de sequência abaixo ilustra como o sistema funciona:



3. Mecanismos Implementados

3.1. Comunicação e Protocolo via Sockets TCP

Toda a comunicação é construída sobre sockets TCP padrão da biblioteca java.net. Para padronizar a troca de informações, foi criada a classe Pacote, que encapsula dois elementos: um Tipo (um enum que define a ação, como LOGIN ou POSTAR_MENSAGEM) e um conteúdo (um Object genérico que carrega os dados, como credenciais ou um objeto Mensagem).

3.2. Replicação Assíncrona e Consistência Eventual

Para garantir que todos os utilizadores vejam o mesmo mural de mensagens, independentemente do nó ao qual se conectam, o sistema emprega um modelo de replicação de dados. A estratégia escolhida foi a **replicação assíncrona** (também conhecida como *lazy replication*), que prioriza a baixa latência para o cliente e a alta disponibilidade do sistema, em detrimento de uma consistência imediata. O resultado deste modelo é a garantia de **consistência eventual**.

Replicação Assíncrona

Quando um nó recebe uma nova mensagem de um cliente, ele não precisa de esperar que todos os outros nós da rede confirmem o recebimento antes de responder ao cliente. Em vez disso, o processo ocorre da seguinte forma:

1. O nó recebe o Pacote(POSTAR_MENSAGEM) de um cliente autenticado.
2. Ele imediatamente adiciona a nova mensagem à sua cópia local do Mural.

3. Imediatamente a seguir, ele inicia o processo de replicação, enviando a nova mensagem para todos os seus pares conhecidos.
4. O cliente não precisa de esperar que a replicação termine; a sua ação é considerada completa assim que o nó local aceita a mensagem.

Esta abordagem torna a operação de postagem muito rápida do ponto de vista do utilizador e garante que o sistema continue a aceitar novas mensagens mesmo que alguns nós da rede estejam temporariamente offline.

3.3. Autenticação e Controle de Acesso

O sistema implementa um mecanismo de autenticação básico, onde cada nó mantém um mapa de utilizadores e senhas. O acesso às funcionalidades do mural é diferenciado:

- **Acesso Público:** A leitura do mural é permitida a qualquer cliente, mesmo que não esteja autenticado.
- **Acesso Privado:** Apenas utilizadores que realizaram o login com sucesso podem postar novas mensagens no mural.

Implementação:

- O gatilho para a replicação está no TratadorDeConexao.java, onde, após uma adição bem-sucedida ao mural local, o método replicarParaPeers() do nó pai é invocado. A lógica de replicação em si, na classe No.java, demonstra a natureza "dispare e esqueça" (fire-and-forget) do mecanismo:

```
110     // Envia uma nova mensagem para todos os outros nós ("peers") na rede.
111     public void replicarParaPeers(Mensagem mensagem) {  1 usage
112         if (!executando) return; // Não tenta replicar se o nó estiver no processo de parada.
113         System.out.printf("[Nó %d] Replicando mensagem para %d peer(s)...%n", id, peers.size());
114         // Cria um pacote específico para replicação.
115         Pacote pacote = new Pacote(Pacote.Tipo.REPLICAR_MSG, mensagem);
116         // Itera sobre a lista de peers conhecidos.
117         for (Integer peerPorta : peers.values()) {
118             // Usa try-with-resources para garantir que o socket e o stream sejam fechados.
119             try (Socket socket = new Socket( host: "localhost", peerPorta);
120                  ObjectOutputStream out = new ObjectOutputStream(socket.getOutputStream())) {
121                 // Envia o pacote para o peer.
122                 out.writeObject(pacote);
123             } catch (IOException e) {
124                 // Se a conexão falhar, apenas informa no console, não para o sistema.
125                 System.err.printf("[Nó %d] Falha ao replicar para a porta %d (nó pode estar offline).%n", id, peerPorta);
126             }
127         }
128     }
```

Consistência Eventual

A consistência eventual é a garantia de que, se nenhuma nova atualização for feita ao sistema, todas as réplicas (os murais em cada nó) irão, **eventualmente**, convergir para

o mesmo estado. No nosso sistema, esta garantia é alcançada através de dois mecanismos combinados:

1. **Propagação Ativa (Replicação):** O método replicarParaPeers() descrito acima, que ativamente "empurra" as novas mensagens para os nós que estão online.
2. **Sincronização Passiva (Reconciliação):** Um mecanismo de "puxar" (pull) que permite que um nó que esteve offline se atualize ao voltar para a rede.

É a combinação destes dois mecanismos que torna o sistema tolerante a falhas. Um nó pode falhar, perder várias mensagens, mas ao reiniciar, ele irá se reconciliar e atingir o mesmo estado dos outros.

Implementação:

A reconciliação é implementada no método sincronizarComPeers(), que é chamado sempre que um nó inicia. Ele atua como um cliente para os seus pares, pedindo o estado completo do mural.

```
130     // Mecanismo de reconciliação: tenta se conectar a outros nós para obter o estado atual do mural.
131     @SuppressWarnings("unchecked")  1 usage
132     private void sincronizarComPeers() {
133         System.out.printf("[Nó %d] Tentando sincronizar com a rede...%n", id);
134         Pacote pacoteDePedido = new Pacote(Pacote.Tipo.PEDIDO_SYNC,  conteudo: null);
135         for (Integer peerPorta : peers.values()) {
136             try (Socket socket = new Socket( host: "localhost", peerPorta);
137                  ObjectOutputStream out = new ObjectOutputStream(socket.getOutputStream());
138                  ObjectInputStream in = new ObjectInputStream(socket.getInputStream())) {
139
140                 // Envia um pacote pedindo a sincronização.
141                 out.writeObject(pacoteDePedido);
142                 // Aguarda a resposta.
143                 Pacote pacoteDeResposta = (Pacote) in.readObject();
144
145                 if (pacoteDeResposta.getTipo() == Pacote.Tipo.RESPOSTA_SYNC) {
146                     // Se a resposta for a correta, atualiza o mural local.
147                     List<Mensagem> mensagensRecebidas = (List<Mensagem>) pacoteDeResposta.getConteudo();
148                     int adicionadas = muralLocal.adicionarTodas(mensagensRecebidas);
149                     System.out.printf("[Nó %d] Sincronização bem-sucedida com a porta %d. %d novas mensagens adicionadas.%n", id, peerPorta, adicionadas);
150                     return; // Para na primeira sincronização bem-sucedida.
151                 }
152             } catch (IOException | ClassNotFoundException e) {
153                 System.err.printf("[Nó %d] Falha ao sincronizar com a porta %d (nó pode estar offline).%n", id, peerPorta);
154             }
155         }
156         // Se não conseguiu se conectar a nenhum peer, inicia com o mural vazio.
157         System.out.printf("[Nó %d] Não foi possível sincronizar com nenhum peer. Iniciando com mural local.%n", id);
158     }
```

3.3. Autenticação e Controle de Acesso

Para garantir que apenas utilizadores autorizados possam modificar o estado do mural, o sistema implementa um mecanismo de autenticação básico, porém eficaz, baseado em utilizador e senha. A autenticação é **baseada na sessão**, o que significa que um utilizador permanece autenticado durante todo o tempo em que a sua conexão TCP com um nó estiver ativa.

A lógica de controle de acesso distingue claramente entre ações públicas e privadas:

- **Acesso Público (Leitura):** Qualquer cliente, mesmo sem se autenticar, pode solicitar e receber a lista completa de mensagens do mural. Esta é considerada uma operação pública e de baixo risco.
- **Acesso Privado (Escrita):** Apenas clientes que tenham concluído com sucesso o processo de login podem postar novas mensagens. Esta é uma ação privada que modifica o estado compartilhado do sistema e, portanto, exige autorização.

Fluxo de Autenticação

1. **Envio de Credenciais:** O cliente solicita ao utilizador o seu nome e senha. Estas duas informações são concatenadas numa única string (ex: "anderson;123") e enviadas para o nó num Pacote do tipo LOGIN.
2. **Validação no Nó:** O servidor (especificamente a thread TratadorDeConexao que gere aquela conexão) recebe o pacote. Ele extrai as credenciais e as valida contra um Map de utilizadores e senhas armazenado no objeto No.
3. **Gestão de Estado da Sessão:** Se as credenciais estiverem corretas, uma flag booleana autenticado dentro do objeto TratadorDeConexao é definida como true. Esta flag persiste enquanto a conexão estiver ativa, servindo como a "sessão" do utilizador. O nó então envia uma resposta de LOGIN_OK ou LOGIN_FALHA para o cliente.
4. **Verificação de Permissão:** Sempre que o nó recebe um pedido para POSTAR_MENSAGEM, ele primeiro verifica o estado da flag autenticado daquela conexão. Se for true, a operação é permitida; caso contrário, é recusada.

Implementação:

A lógica central de validação e controle de acesso reside no método processarPacote da classe TratadorDeConexao.java.

```java
private void processarPacote(Pacote pacote) throws IOException {  1 usage
    System.out.printf("[Nó %d] Pacote recebido: %s%n", noPai.getId(), pacote.getTipo());
    // Obtém uma referência ao mural local do nó pai para poder manipulá-lo.
    Mural mural = noPai.getMuralLocal();

    // Estrutura switch que funciona como o roteador principal da lógica do servidor.
    switch (pacote.getTipo()) {
        case LOGIN:
            // Separa o conteúdo "usuario:senha" em duas partes.
            String[] credenciais = ((String) pacote.getConteudo()).split( regex: ":");
            if (credenciais.length == 2) {
                String usuario = credenciais[0];
                String senha = credenciais[1];
                // Verifica se o usuário existe e se a senha corresponde.
                if (noPai.getUsuarios().getOrDefault(usuario,  defaultValue: "").equals(senha)) {
                    this.autenticado = true; // Marca esta conexão como autenticada.
                    out.writeObject(new Pacote(Pacote.Tipo.LOGIN_OK,  conteudo: "Login bem-sucedido!"));
                } else {
                    out.writeObject(new Pacote(Pacote.Tipo.LOGIN_FALHA,  conteudo: "Usuário ou senha inválidos."));
                }
            }
            break;

        case LER_MURAL:
            // Qualquer cliente, autenticado ou não, pode ler o mural.
            // Envia de volta um pacote contendo a lista completa de mensagens.
            out.writeObject(new Pacote(Pacote.Tipo.MURAL_ATUALIZADO, mural.getTodasAsMensagens()));
            break;

        case POSTAR_MENSAGEM:
            // Apenas usuários autenticados podem postar.
            if (autenticado) {
                Mensagem novaMensagem = (Mensagem) pacote.getConteudo();
                // Adiciona a mensagem ao mural local. O 'if' verifica se a mensagem era nova.
                if (mural.adicionarMensagem(novaMensagem)) {
                    // Se a mensagem foi adicionada com sucesso (não era duplicata),
                    // o nó pai é instruído a replicá-la para os outros nós da rede.
                    noPai.replicarParaPeers(novaMensagem);
                }
            }
```

3.4. Simulação de Falha e Reconciliação

A capacidade de um sistema distribuído de continuar a operar corretamente na presença de falhas é a sua principal medida de robustez. Para validar esta capacidade, o nosso sistema implementa um ciclo completo de tolerância a falhas, que consiste em dois mecanismos distintos, mas complementares: a **simulação de uma falha** controlada e a **recuperação através da reconciliação**.

Simulação de Falha

Para testar a resiliência da rede, é simulada a "queda" abrupta de um nó. Esta simulação não se limita a simplesmente parar de enviar mensagens; ela encerra de forma explícita e garantida o processo servidor do nó alvo.

1. **Orquestração Centralizada:** A classe Simulador atua como o orquestrador do teste. Após iniciar todos os nós da rede, ela agenda uma falha para ocorrer após um período predeterminado (60 segundos).
2. **Paragem Explícita:** Em vez de usar mecanismos menos fiáveis como a interrupção de threads, o simulador mantém uma referência a cada objeto No.

Quando o tempo de espera termina, ele invoca diretamente o método parar() no nó alvo.

3. **Encerramento do Servidor:** O método parar() define uma flag volatile boolean executando para false e, crucialmente, fecha o ServerSocket do nó. Esta ação força o método serverSocket.accept(), que estava bloqueado à espera de conexões, a lançar uma SocketException, garantindo que o loop do servidor termine e a sua porta de escuta seja liberada, tornando o nó verdadeiramente offline.

Implementação:

O controlo da simulação é visível no Simulador.java, que chama o método de paragem explícita.

```
68          No noQueVaiFalhar = nos.get(noParaDerrubar - 1); // (-1 porque a lista começa em 0)
69          System.out.printf("%n>>> SIMULANDO FALHA DO NÓ %d <<<%n", noParaDerrubar);
70          // Chama o metodo 'parar()' do nó, que o encerra de forma controlada e segura.
71          noQueVaiFalhar.parar();
```

O método parar() na classe No.java garante o encerramento seguro do servidor.

```
71      // Metodo para parar o nó de forma explícita e segura, chamado pelo Simulador.
72      public void parar() {  1 usage
73          this.executando = false; // Sinaliza para o loop do servidor parar.
74          try {
75              // Fecha o ServerSocket para forçar o método 'accept()' a sair do estado de bloqueio.
76              if (serverSocket != null && !serverSocket.isClosed()) {
77                  serverSocket.close();
78              }
79          } catch (IOException e) {
80              System.err.printf("[Nó %d] Erro ao fechar o socket do servidor: %s%n", id, e.getMessage());
81          }
82      }
```

Reconciliação

A recuperação de um nó que esteve offline é tratada pelo mecanismo de **reconciliação**. Este processo é ativado automaticamente sempre que um nó inicia, garantindo que ele possa reintegrar-se à rede e recuperar qualquer estado que tenha perdido.

1. **Ativação na Inicialização:** A primeira ação de um nó, após um breve atraso para estabilização da rede, é chamar o método sincronizarComPeers().

2. **Mecanismo de "Pull":** O nó atua como um cliente para os seus pares. Ele envia um Pacote do tipo PEDIDO_SYNC para um dos nós ativos.
3. **Transferência de Estado:** O nó que recebe o pedido responde com um Pacote do tipo RESPOSTA_SYNC, cujo conteúdo é a sua lista completa de mensagens do mural.
4. **Atualização Local:** O nó que iniciou a sincronização recebe esta lista e a utiliza para atualizar o seu próprio mural. O método adicionarTodas garante que apenas as mensagens que ele ainda não possui sejam adicionadas, evitando duplicatas e convergindo o seu estado para o do resto da rede.

Este mecanismo é a espinha dorsal da consistência eventual, pois garante que, mesmo após uma ausência, um nó pode autonomamente restaurar o seu estado para corresponder ao dos seus pares, tornando o sistema resiliente a falhas temporárias.

Persistência de Estado

Para tornar o sistema verdadeiramente tolerante a falhas e permitir uma recuperação completa, foi implementada uma camada de persistência. O estado do mural de cada nó, que antes existia apenas em memória, agora é salvo em disco.

1. **Carregamento na Inicialização:** Ao iniciar, cada nó invoca o método carregarMuralDoDisco(). Este método verifica a existência de um ficheiro de dados local (ex: mural_no_1.dat). Se o ficheiro existir, o objeto Mural é desserializado, restaurando o estado do nó para o ponto em que ele estava antes de ser desligado.
2. **Salvamento Automático:** Sempre que o estado do mural é alterado (seja pela adição de uma nova mensagem de um cliente ou pela recepção de uma mensagem replicada), o método salvarMuralNoDisco() é chamado. Ele serializa o objeto Mural atual e o sobrescreve em disco, garantindo que o estado persistido esteja sempre atualizado.

Implementação: A persistência foi implementada na classe No.java utilizando FileOutputStream e FileInputStream para manipular os ficheiros de dados.

```java
// Metodo para carregar o mural de um ficheiro.
private void carregarMuralDoDisco() {  1 usage
    File arquivoMural = new File(NOME_ARQUIVO_MURAL);
    if (arquivoMural.exists()) {
        // Usa try-with-resources para garantir que os streams sejam fechados.
        try (FileInputStream fis = new FileInputStream(arquivoMural);
             ObjectInputStream ois = new ObjectInputStream(fis)) {
            // Lê o objeto Mural completo do ficheiro.
            this.muralLocal = (Mural) ois.readObject();
            System.out.printf("[Nó %d] Mural carregado do disco com sucesso.%n", id);
        } catch (IOException | ClassNotFoundException e) {
            System.err.printf("[Nó %d] Erro ao carregar mural do disco. Iniciando com um novo. Erro: %s%n", id, e.getMessage());
            this.muralLocal = new Mural();
        }
    } else {
        // Se o ficheiro não existe, simplesmente inicia com um mural novo e vazio.
        System.out.printf("[Nó %d] Nenhum mural salvo encontrado. Iniciando com um novo.%n", id);
        this.muralLocal = new Mural();
    }
}
```

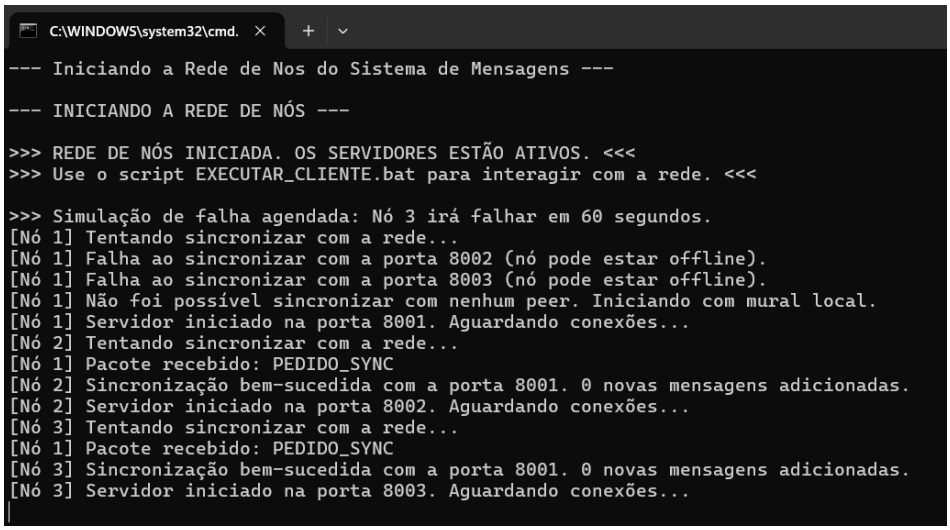
```java
// Metodo para salvar o mural atual em um ficheiro.
public synchronized void salvarMuralNoDisco() {  3 usages
    // Usa try-with-resources.
    try (FileOutputStream fos = new FileOutputStream(NOME_ARQUIVO_MURAL);
         ObjectOutputStream oos = new ObjectOutputStream(fos)) {
        // Escreve o objeto Mural completo no ficheiro.
        oos.writeObject(this.muralLocal);
    } catch (IOException e) {
        System.err.printf("[Nó %d] Erro ao salvar mural no disco: %s%n", id, e.getMessage());
    }
}
```

4. Evidências de Execução e Análise

Esta seção apresenta as evidências de execução do sistema, demonstrando o funcionamento de cada requisito através dos logs gerados.

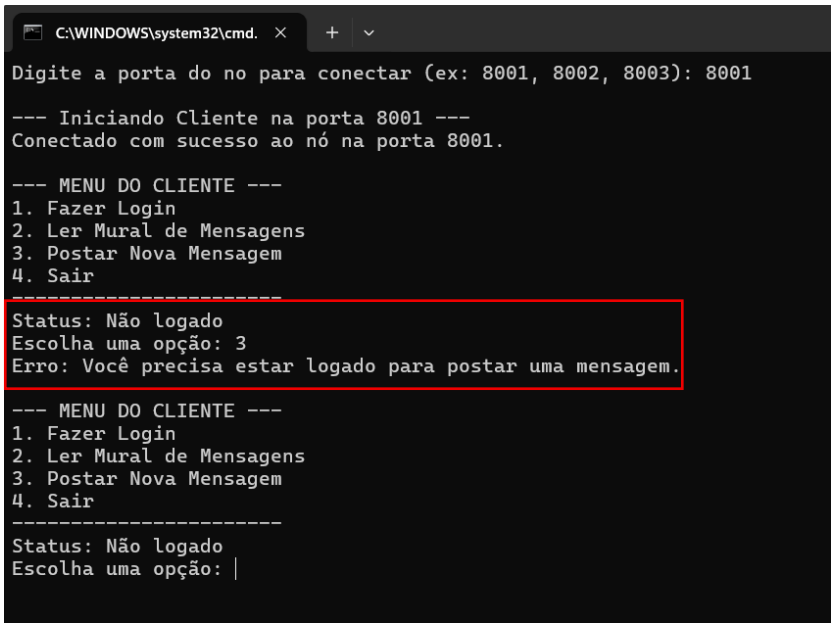
4.1. Inicialização da Rede e Sincronização

Ao iniciar a rede, os nós entram online de forma escalonada para evitar condições de corrida. O log abaixo demonstra que os Nós 2 e 3, ao iniciarem, conseguem contactar o Nó 1 (que já estava online) e sincronizar o seu estado, provando que o mecanismo de reconciliação é ativado com sucesso desde o início.

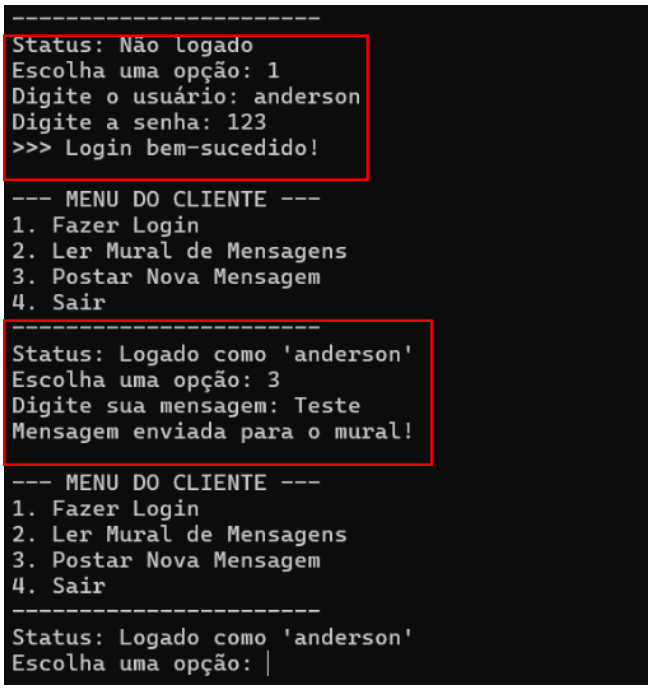


4.2. Demonstração do Controle de Acesso

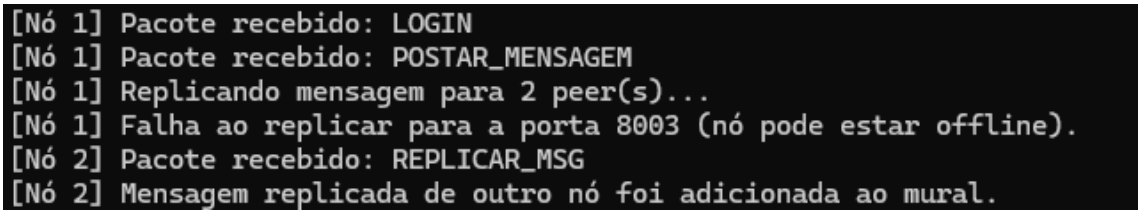
Para validar o sistema de autenticação, foram realizados dois testes. Primeiramente, um cliente não autenticado tentou postar uma mensagem, e a ação foi corretamente bloqueada pelo sistema.



Em seguida, o cliente realizou o login com sucesso e conseguiu postar a mensagem.

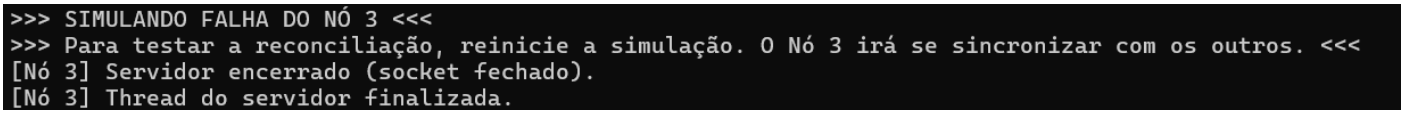


O log do servidor correspondente (abaixo) confirma o recebimento do pacote POSTAR_MENSAGEM e a subsequente replicação da mensagem para os outros 2 peers da rede, como esperado.



4.3. Simulação de Falha e suas Consequências

Conforme programado no Simulador, após 60 segundos de operação, a falha do Nó 3 é acionada. O log do servidor abaixo mostra o momento exato em que a falha é simulada e o servidor do Nó 3 é encerrado.



Para provar que o nó está de facto inativo, uma tentativa de conexão à porta 8003 foi realizada. O resultado, como visto abaixo, é um erro de "conexão recusada", confirmando que o nó está offline.

```
Digite a porta do no para conectar (ex: 8001, 8002, 8003): 8003

--- Iniciando Cliente na porta 8003 ---
Erro: Não foi possível conectar ao nó na porta 8003. O nó está offline?
Pressione qualquer tecla para continuar. . .
```

4.4. Operação Durante a Falha e Mecanismo de Reconciliação

Com o Nó 3 offline, uma nova mensagem foi enviada por um cliente conectado a um nó funcional (Nó 1).

```
--- MENU DO CLIENTE ---
1. Fazer Login
2. Ler Mural de Mensagens
3. Postar Nova Mensagem
4. Sair
-----------------------
Status: Logado como 'anderson'
Escolha uma opção: 3
Digite sua mensagem: Esta mensagem foi enviada durante a falha do Nó 3
Mensagem enviada para o mural!
```

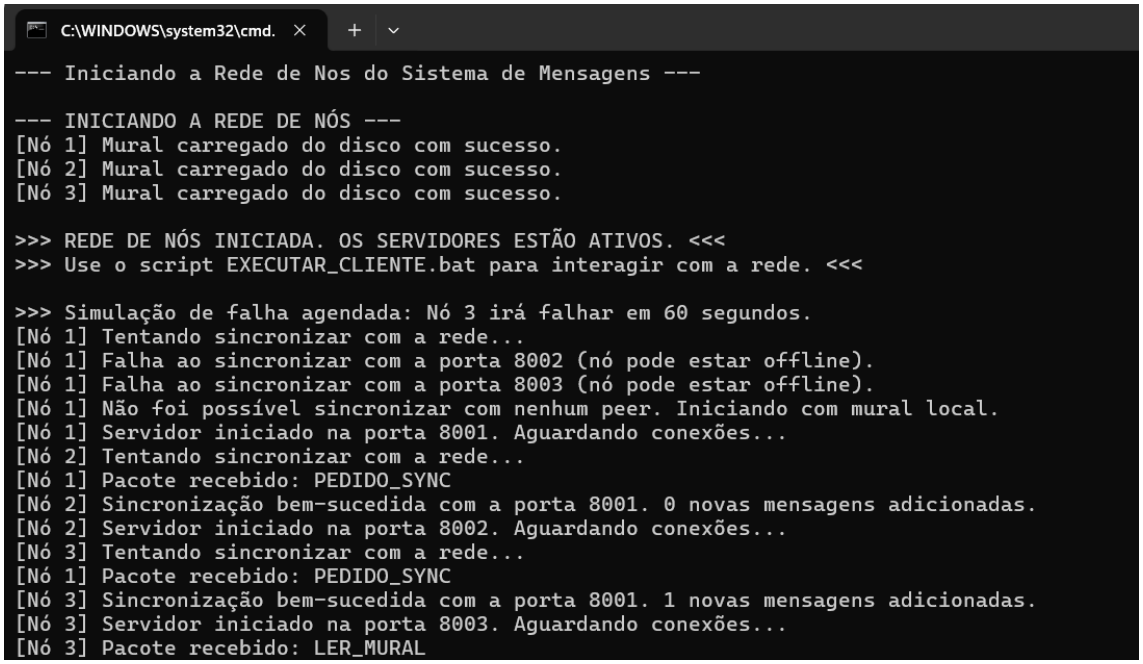
O log do servidor (Nó 1) mostra que ele tentou replicar a mensagem para os seus 2 peers, mas falhou ao tentar contactar a porta 8003, o que é o comportamento esperado.

```
[Nó 1] Pacote recebido: POSTAR_MENSAGEM
[Nó 1] Replicando mensagem para 2 peer(s)...
[Nó 1] Falha ao replicar para a porta 8003 (nó pode estar offline).
```

A última fase do teste demonstra a resiliência do sistema: a sua capacidade de continuar a operar com um nó a menos e, crucialmente, a eficácia do mecanismo de recuperação com persistência.

Com o Nó 3 offline, uma nova mensagem foi enviada por um cliente conectado a um nó funcional (Nó 1). O sistema permaneceu disponível e o log do Nó 1 confirmou que a tentativa de replicação para a porta 8003 falhou, como esperado. Neste momento, o estado dos Nós 1 e 2 foi salvo em disco, contendo a nova mensagem, enquanto o estado do Nó 3 permaneceu desatualizado.

Finalmente, a simulação foi reiniciada para simular o regresso do Nó 3 à rede.



Análise: O log acima demonstra o sucesso inequívoco da nova arquitetura persistente e do ciclo completo de falha e recuperação. A análise da sequência de eventos é a seguinte:

1. **Recuperação de Estado:** As três primeiras linhas [Nó X] Mural carregado do disco com sucesso. provam que os Nós 1, 2 e 3 restauraram o seu estado a partir dos seus respetivos ficheiros .dat. O Nó 1 e o Nó 2 recuperaram o mural que continha a mensagem enviada durante a falha, enquanto o Nó 3 carregou o seu estado desatualizado.
2. **Ativação da Reconciliação:** Em seguida, o mecanismo de reconciliação é ativado. O log [Nó 3] Sincronização bem-sucedida com a porta 8001. 1 novas mensagens adicionadas. é a prova final do processo. Ele mostra que:
    - O **Nó 3**, ao voltar, conectou-se ao **Nó 1** e "puxou" o seu estado completo.
    - Ele comparou o mural recebido com o seu próprio e identificou **1 nova mensagem** que não possuía (exatamente aquela que foi enviada enquanto estava offline).
    - A mensagem foi adicionada ao seu mural local e, em seguida, salva em disco, completando o ciclo de convergência.

5. Conclusão

A implementação resultou num sistema funcional que demonstra na prática conceitos teóricos essenciais de sistemas distribuídos. A utilização de Sockets TCP em Java mostrou-se uma base sólida para a construção de um protocolo de comunicação customizado e para a criação de uma rede peer-to-peer sem um ponto único de falha.

A adição de uma camada de persistência de estado foi fundamental para garantir uma reconciliação completa, permitindo que um nó recuperasse mensagens perdidas mesmo após uma reinicialização total da rede.

O projeto serviu como uma valiosa validação prática dos conceitos de consistência eventual, tolerância a falhas e recuperação de estado, consolidando o conhecimento teórico adquirido na disciplina.