INSTITUTO FEDERAL DE EDUCAÇÃO, CIÊNCIA E TECNOLOGIA DA BAHIA
CAMPUS - SANTO ANTÔNIO DE JESUS - BAHIA
CURSO DE TECNOLOGIA EM ANÁLISE E DESENVOLVIMENTO DE SISITEMAS

MARCELO DE JESUS

RONALDO CORREIA COUTO

Simulação Completa de Ambiente Distribuído com Núcleo Modular, Comunicação Multigrupo e Implementação Híbrida de Middleware

SANTO ANTONIO DE JESUS – BA
2025

MARCELO DE JESUS

RONALDO CORREIA

COUTO

Simulação Completa de Ambiente Distribuído com Núcleo Modular, Comunicação Multigrupo e Implementação Híbrida de Middleware

Relatório técnico da atividade de Sistemas Distribuídos Simulação Completa de Ambiente Distribuído com Núcleo Modular, Comunicação Multigrupo e Implementação Híbrida de Middleware, elaborado como requisito parcial de avaliação para a disciplina de Padrões de Projeto, ministrada pelo Prof. Felipe Silva.

SANTO ANTONIO DE JESUS – BA

2025

1. Introdução

Este relatório apresenta de forma descritiva o desenvolvimento de uma simulação completa de sistema distribuído modular, com foco em comunicação multigrupo, sincronização, eleição de líderes e tolerância a falhas. Utilizando Python, o projeto integra múltiplas formas de comunicação (TCP, UDP Multicast), middlewares híbridos (gRPC e Pyro5), relógios lógicos de Lamport e mecanismos de heartbeat.

Com o objetivo de demonstrar a aplicação prática de conceitos avançados em sistemas distribuídos, a simulação implementa liderança distribuída, captura de estado global, autenticação por token e controle de acesso descentralizado. A arquitetura proposta é escalável, extensível e voltada à interoperabilidade entre diferentes grupos de processos.

2.Justificativa dos modelos arquiteturais adotados

A arquitetura do sistema foi projetada com foco em modularidade, escalabilidade, tolerância a falhas e interoperabilidade entre diferentes modelos de middleware. A separação em dois grupos de processos independentes, com middlewares distintos, permite simular cenários reais de sistemas heterogêneos distribuídos, nos quais diferentes tecnologias coexistem e precisam se comunicar.

A escolha de uma **arquitetura modular** facilita a manutenção e evolução do sistema. Cada componente (TCP, UDP, gRPC, RMI, Autenticação, etc.) foi isolado em pacotes distintos, seguindo o princípio da separação de responsabilidades. Isso torna a aplicação mais extensível e facilita a integração de novos recursos sem impactar outras partes do sistema.

A divisão em **grupos independentes** permite simular ambientes paralelos com políticas internas distintas. A comunicação **intra-grupo via TCP (Sockets)** foi escolhida por sua confiabilidade e controle, o que é essencial para garantir integridade nas trocas de mensagens dentro do mesmo grupo. Já a comunicação **intergrupos via UDP Multicast** foi adotada para permitir uma troca eficiente e assíncrona entre grupos, economizando largura de banda e reduzindo o acoplamento entre as partes do sistema.

O uso de **middlewares distintos por grupo** tem justificativas complementares:

- O **gRPC** (utilizado no Grupo A) oferece alto desempenho, contratos bem definidos por meio de arquivos `.proto`, suporte a múltiplas linguagens e uma comunicação eficiente baseada em HTTP/2. Foi escolhido para representar um middleware moderno e escalável, ideal para microsserviços.
- O **Java RMI** (utilizado no Grupo B) representa uma solução tradicional, com forte integração à linguagem Java e baixo custo de configuração inicial, facilitando a troca de objetos remotos em ambientes homogêneos. Essa escolha permite contrastar as limitações e vantagens entre tecnologias legadas e modernas.

O sistema também adota **relógios lógicos de Lamport** para garantir uma ordenação causal dos eventos distribuídos, o que é fundamental para manter a consistência entre processos que operam de forma assíncrona.

Por fim, a aplicação de algoritmos clássicos como **Bully e Anel** para eleição de líderes, juntamente com o uso de **Chandy-Lamport para captura de estado global** e **heartbeat para detecção de falhas**, justifica a escolha de uma arquitetura robusta e realista, que cobre os principais desafios de um sistema distribuído: coordenação, consistência e disponibilidade.

3. Tecnologias de middleware empregadas

Nesta simulação, cada grupo de processos foi projetado para utilizar um middleware distinto, a fim de representar a interoperabilidade entre sistemas heterogêneos. Na parte desenvolvida pelo Grupo A, a comunicação foi implementada utilizando o **gRPC (Google Remote Procedure Call)**.

O **gRPC** é um framework moderno de comunicação remota baseado em HTTP/2 e serialização com Protocol Buffers. Foi escolhido por oferecer:

- Alta performance e baixo overhead em chamadas remotas;
- Suporte a autenticação e metadados;
- Compatibilidade com diversas linguagens;
- Fortes garantias de contrato via arquivos `.proto`.

No Grupo A, cada nó atua como cliente e servidor gRPC, permitindo chamadas bidirecionais entre os nós do grupo, utilizando TCP para transporte. A configuração e inicialização do servidor gRPC podem ser observadas nos logs de execução (porta 7001, 7002, 7003 para os nós 1, 2 e 3, respectivamente).

Além do gRPC, a infraestrutura também inclui:

- **Sockets TCP:** para comunicação ponto a ponto dentro do grupo;
- **UDP Multicast:** para comunicação entre grupos, incluindo sincronização, eleição de supercoordenador e envio de mensagens globais;
- **RMI:** já inicializado nos nós do Grupo A, mas ainda não utilizado, pois está reservado para interação com o Grupo B na parte a ser implementada pelo outro integrante.

4. Estratégias de sincronização, eleição e detecção de falhas

Sincronização Temporal

Cada nó mantém um **Relógio Lógico de Lamport**, utilizado para marcar eventos internos, envios e recebimentos de mensagens. Isso garante uma ordenação parcial dos eventos e permite detectar causalidade entre ações distribuídas. O relógio é incrementado a cada evento local ou recebido via mensagem, como visto nos logs: `Clock=2`, `Clock=3`, etc.

Eleição de Líder

No Grupo A, a eleição de líder segue o **algoritmo de Bully**, que será disparado quando necessário (por exemplo, falha do líder). Até o momento da execução registrada, a eleição ainda não foi ativada. A lógica foi projetada para identificar o nó com o maior ID ativo como líder, conforme o algoritmo clássico.

Detecção de Falhas

Foi implementado um mecanismo de **heartbeat periódico** entre os nós do grupo. Cada nó envia sinais de vida em intervalos regulares. Se um nó deixar de responder por três ciclos consecutivos, ele é considerado inativo. O sistema então reage com:

- Registro de falha;
- Retirada do nó da comunicação ativa;
- Nova eleição de líder, caso o nó falho fosse o atual líder.

Este mecanismo torna o sistema mais resiliente a falhas de processo e garante disponibilidade contínua dos serviços distribuídos.

5.Detalhamento sobre o funcionamento das políticas de acesso.

Geração e emissão de tokens

- O sistema gera tokens únicos para cada nó com um tempo de vida limitado (TTL de 60 segundos).
- A geração é feita pelo `AuthManager`, que cria um `AuthToken` contendo um identificador único e a data/hora de expiração.
- Esses tokens funcionam como “credenciais digitais” que identificam o nó.

2. Validação dos tokens

- Cada requisição recebida pelos serviços deve incluir um token.

- O `AuthManager` verifica se o token é válido, ou seja:
    - O token existe no armazenamento (`ConcurrentHashMap`).
    - O token não está expirado (baseado na data/hora atual vs. data/hora de expiração do token).
- Se o token for inválido ou expirado, o acesso é negado.

3. Gerenciamento de sessão

- Além do token de autenticação, o sistema mantém sessões gerenciadas pelo `SessionManager`.
- Cada sessão tem um token UUID com tempo de expiração configurável.
- Sessões expiradas são removidas automaticamente para evitar uso indevido.
- As sessões podem ser renovadas para manter o acesso ativo enquanto o usuário estiver usando o sistema.

6. Avaliação crítica da solução: vantagens, limitações e sugestões de melhorias

Vantagens:

- **Robustez no processo de eleição:** Algoritmos como Bully e Anel garantem que um líder será eleito mesmo em caso de falhas de nós.
- **Descentralização:** Especialmente no algoritmo de Anel, não há um ponto único de falha.
- **Escalabilidade:** O algoritmo em anel é simples e funciona bem em grupos de tamanho moderado.
- **Negociação entre líderes:** A definição de um supercoordenador global permite uma coordenação eficiente entre grupos diferentes.

Limitações:

- **Complexidade em grupos grandes:** Algoritmo de anel pode ser lento em grupos muito grandes, pois a mensagem circula por todos os nós.
- **Falha em comunicação:** A comunicação multicast e TCP precisa ser confiável; perdas podem afetar a eleição.
- **Sincronização e consistência:** Garantir que todos os nós concordem sobre o líder pode ser desafiador, especialmente em ambientes distribuídos reais.
- **Ponto único de falha no supercoordenador:** Apesar de haver eleição entre líderes, o supercoordenador pode ser um gargalo.

Sugestões de melhorias:

- **Implementar mecanismos de timeout e retry:** Para lidar melhor com falhas temporárias de comunicação.
- **Adicionar redundância para o supercoordenador:** Usar um grupo de líderes com consenso em vez de um único supercoordenador.
- **Otimizar comunicação:** Usar algoritmos mais eficientes para grupos muito grandes, como eleição baseada em anéis com saltos.
- **Monitoramento dinâmico:** Adaptar o algoritmo para detectar e reagir a mudanças dinâmicas no grupo (entradas e saídas de nós).
- **Simular falhas:** Testar o sistema com falhas reais para garantir a resiliência do protocolo.

7.Descrição de testes realizados com resultados esperados e observados

Teste: Execução simultânea de múltiplos nós com autenticação e comunicação

Objetivo:

Verificar se múltiplos nós do sistema conseguem iniciar corretamente, gerar tokens de autenticação, se comunicar via TCP, Multicast e RMI, e validar seus status de forma sincronizada.

Procedimento:

- Três nós foram iniciados em terminais diferentes com comandos Maven específicos.
- Cada nó recebeu um token de autenticação único gerado pelo sistema (`AuthManager`).
- Os nós enviaram mensagens via Multicast (`mc`) e TCP (`tcp`).
- Comandos de `login` e `status` foram usados para testar a autenticação e consulta ao estado dos nós via RMI.

Resultados esperados:

- Cada nó deve iniciar com sucesso seus servidores gRPC, TCP, Multicast e RMI.
- Cada nó deve gerar um token único válido.
- Mensagens enviadas via multicast devem ser recebidas pelos demais nós.
- O comando `login` deve autenticar os nós, retornando tokens.
- O comando `status` deve indicar o estado atual de cada nó, confirmando que estão ativos e autenticados.

Resultados observados na imagem:

- Os três nós iniciaram sem erros críticos, cada um em portas distintas.
- Tokens únicos foram gerados para cada nó (`Token=node-1-xxxx`, `Token=node-2-xxxx`, etc).
- Mensagens multicast foram enviadas e recebidas corretamente, confirmadas pelas saídas do terminal (`[MC] Enviado: ...`).

- Os comandos `login` mostraram tokens válidos, e os comandos `status` retornaram confirmação de status OK para cada nó (`Status via RMI: OK: node=1`, etc).
- Nenhum erro de comunicação ou autenticação foi detectado durante o teste.

Conclusão:

O sistema demonstrou funcionamento esperado na comunicação entre múltiplos nós, com autenticação e validação de sessões funcionando corretamente. A interação via TCP, multicast e RMI está consistente.

Link do repositório: github