Relatório Técnico – Simulação de Ambiente Distribuído com Núcleo Modular

Hildemar Lemos de Santana Júnior, Kleberson de Jesus Sousa e Thiago Sampaio Santos

Introdução

Este relatório tem como objetivo apresentar, de forma detalhada e crítica, o desenvolvimento de uma simulação de ambiente distribuído realizada no contexto da disciplina de Sistemas Distribuídos. A proposta não se limitou à implementação técnica: buscou-se aplicar, com profundidade, os conceitos teóricos discutidos em sala — como coordenação distribuída, eleição de líderes, sincronização temporal, comunicação entre processos e tolerância a falhas — em um sistema funcional, modular e escalável.

Mais do que cumprir os requisitos da disciplina, o projeto foi concebido como uma oportunidade de explorar tecnologias modernas do ecossistema Java, com foco em boas práticas de engenharia de software, organização arquitetural e viabilidade de manutenção futura.

1 - Justificativa Arquitetural e Descrição do Sistema

A arquitetura adotada partiu da premissa de modularidade e separação de responsabilidades. Cada nó da rede distribuída foi projetado como uma unidade autônoma, capaz de se comunicar, coordenar e tomar decisões de forma independente, mas também colaborativa.

A divisão em dois grupos distintos — Grupo A e Grupo B — permitiu explorar diferentes abordagens de middleware e algoritmos de eleição, enriquecendo a análise comparativa entre soluções. Essa segmentação não foi arbitrária: ela refletiu a intenção de testar tecnologias com características complementares e avaliar seu comportamento em cenários distribuídos.

Os módulos principais implementados foram:

- Núcleo de Comunicação TCP: responsável pela troca direta de mensagens entre nós, com suporte a heartbeat e verificação de falhas. A escolha pelo TCP se deu pela confiabilidade e controle de fluxo que ele oferece, essenciais para garantir consistência entre os nós.
- gRPC (Grupo A): adotado por sua performance superior, tipagem forte e suporte a streaming bidirecional. Ideal para serviços que exigem comunicação eficiente e estruturada.
- RMI (Grupo B): escolhido por sua simplicidade e integração nativa ao Java. Embora menos performático que o gRPC, mostrou-se suficiente para o escopo do grupo B.
- Multicast UDP: utilizado para disseminação de mensagens de controle, especialmente em fases de inicialização e eleição. A leveza e o alcance do multicast justificaram sua adoção para comunicação intergrupo.
- Mecanismos de Eleição: foram implementados dois algoritmos clássicos — Bully (Grupo A) e Ring Election (Grupo B). Essa escolha permitiu observar comportamentos distintos em termos de tempo de resposta, tolerância a falhas e complexidade de implementação.

2 - Tecnologias de Middleware Empregadas

A escolha das tecnologias de middleware foi guiada por critérios rigorosos relacionados a desempenho, compatibilidade com a plataforma Java e facilidade de integração em sistemas distribuídos:

- gRPC destacou-se pela robustez e pela capacidade de lidar com chamadas assíncronas e streaming de dados, o que o torna ideal para aplicações que exigem alta performance e comunicação estruturada entre serviços distribuídos. Seu suporte a múltiplas linguagens também abre possibilidades de integração futura com componentes escritos em outros ambientes.
- RMI, embora seja uma tecnologia de middleware mais antiga, ainda permanece válida para cenários menos exigentes e com requisitos de comunicação mais simples. Sua simplicidade e integração direta com o ecossistema Java tornam-no uma opção prática e eficiente para projetos acadêmicos, protótipos e sistemas que não demandam alto throughput.
- UDP Multicast foi essencial para simular o comportamento broadcast entre os nós do sistema. Essa tecnologia permite o envio de mensagens para múltiplos destinatários simultaneamente, com baixa sobrecarga e boa escalabilidade, característica fundamental para ambientes distribuídos onde a propagação rápida de eventos é necessária.

Além dessas tecnologias, a orquestração com Docker e Docker Compose representou um diferencial importante:

- Ao possibilitar a criação e o gerenciamento de múltiplos contêineres interconectados em uma rede virtual, conseguimos simular um ambiente distribuído realista sem a necessidade de infraestrutura física adicional.
- Essa abordagem facilitou a replicação dos testes em diferentes máquinas e ambientes, garantindo portabilidade e consistência na execução da solução.

3 - Estratégias de Sincronização, Eleição e Detecção de Falhas

Para assegurar uma operação consistente e confiável do sistema distribuído, diversas estratégias foram aplicadas focando na sincronização temporal, eleição de líderes e detecção de falhas.

- A sincronização temporal foi fundamentada nos Relógios Lógicos de Lamport, uma abordagem que garante a ordenação consistente dos eventos mesmo em ambientes assíncronos. Essa técnica se revelou eficaz para manter a coerência temporal entre os nós, fator essencial durante processos críticos, como eleição e autenticação.
- O módulo de heartbeat monitorava continuamente a atividade dos nós, enviando sinais periódicos para detectar disponibilidade. A política adotada consistia em considerar um nó inativo após três tentativas consecutivas sem resposta, equilibrando precisão na detecção de falhas com a minimização de falsos positivos.

- Quando um nó era marcado como inativo, o sistema acionava automaticamente um novo processo de eleição, simulando um comportamento autocorretivo e garantindo o restabelecimento rápido da liderança.
- A eleição de líderes foi conduzida de maneiras distintas para os dois grupos testados:
  - No Grupo A, o algoritmo Bully foi empregado, onde o nó com maior prioridade assumia a liderança. Essa estratégia é simples e eficiente, desde que exista conhecimento prévio das prioridades dos nós.
  - No Grupo B, adotou-se o algoritmo Ring Election, onde o token circula entre os nós até que um líder seja escolhido. Embora mais complexo que o Bully, esse método é mais democrático e apresenta maior tolerância a falhas.

Ambas as estratégias funcionaram conforme o esperado, e a escolha por abordagens diferentes possibilitou uma análise aprofundada das vantagens e limitações de cada método.

4 - Políticas de Acesso e Autenticação

A segurança foi prioridade ao longo de todo o desenvolvimento, garantindo que o sistema tivesse um controle rigoroso sobre quem poderia acessar seus recursos:

- A autenticação utilizou tokens JWT, assinados com o algoritmo HMAC256, o que proporcionou uma camada sólida para validação da identidade dos usuários e serviços.
- Cada requisição enviada ao sistema era cuidadosamente validada antes do processamento, para assegurar que apenas usuários autenticados e autorizados pudessem executar ações, evitando acessos indevidos.
- Os tokens possuíam um tempo de expiração definido, obrigando a renovação periódica e reduzindo o risco de uso indevido em caso de captura ou vazamento.
- O gerenciamento das políticas de acesso ficou sob responsabilidade do líder de cada grupo, que validava e propagava as regras para os demais nós, garantindo consistência na aplicação das políticas no ambiente distribuído.
- Essa centralização trouxe agilidade e uniformidade, contudo elevou a importância do líder para a manutenção da segurança ativa e atualizada, configurando um ponto de atenção relevante para futuras melhorias.

5 - Avaliação Crítica da Solução

A solução desenvolvida apresentou vantagens importantes, mas também algumas limitações a serem consideradas para aprimoramentos futuros:

- Escalabilidade: A modularidade adotada, aliada ao uso de contêineres, permitiu que o sistema fosse expandido com relativa facilidade para atender demandas maiores sem comprometer a performance.
- Organização arquitetural: A aplicação dos padrões de projeto (Observer, Factory Method, Singleton, Strategy) conferiu ao sistema clareza, extensibilidade e facilidade na manutenção e evolução dos componentes.
- Segurança: O uso da autenticação via tokens JWT, junto à validação rigorosa das sessões, se mostrou eficaz no controle de acesso e proteção do ambiente.
- Resiliência: Os mecanismos implementados para eleição de líderes e detecção de falhas proporcionaram uma recuperação automática eficiente diante da inatividade de nós, garantindo continuidade da operação.

Porém, algumas limitações foram identificadas:

- Dependência do líder: A centralização do gerenciamento em um único nó pode representar um ponto de falha e risco em ambientes reais. O uso de algoritmos de consenso distribuído como Raft ou Paxos poderia mitigar essa vulnerabilidade.
- Monitoramento limitado: A ausência de uma interface visual atrativa e interativa dificultou o acompanhamento em tempo real do funcionamento e do estado dos nós, limitando a capacidade de reação rápida a eventos inesperados.
- Persistência de logs: Dados gerados durante a execução não foram armazenados de forma estruturada, restringindo a realização de análises posteriores detalhadas, auditorias e melhorias baseadas em dados históricos.

6 - Testes Realizados

Os testes foram realizados em ambientes controlados, cuidadosamente elaborados para validar os principais aspectos funcionais e não funcionais do sistema. Foram verificados os seguintes pontos:

- Inicialização simultânea: todos os nós foram ativados de forma correta e em tempo adequado, e as conexões TCP entre eles foram estabelecidas sem falhas ou instabilidades, o que confirma a robustez da configuração inicial.
- Eleição de líderes: os algoritmos implementados funcionaram conforme o esperado, apresentando tempos de resposta compatíveis com os requisitos do sistema. A capacidade de reinicialização automática após falhas foi validada, assegurando a continuidade operacional.
- Comunicação intergrupo: as mensagens enviadas via Multicast UDP foram recebidas por todos os nós participantes, demonstrando a eficiência dessa estratégia para propagação rápida e confiável de informações dentro do ambiente distribuído.

- Detecção de falhas: ao simular falhas propositalmente, foram gerados alertas imediatos, e o processo de eleição foi reiniciado automaticamente, evidenciando a resiliência e a capacidade de autocorreção do sistema.
- Autenticação: requisições contendo tokens inválidos foram bloqueadas corretamente, comprovando a efetividade da política de controle de acesso implementada.

Os resultados confirmaram que a solução apresentou estabilidade e confiabilidade satisfatórias, mesmo em condições simuladas que replicam situações adversas.

7 - Extensões Futuras

A partir das lições aprendidas e das observações feitas durante o desenvolvimento e os testes, conseguimos identificar várias melhorias que podem deixar a solução ainda mais completa e alinhada com as necessidades reais que um sistema distribuído moderno exige.

- Uma das principais oportunidades é a implementação de algoritmos de consenso, como Raft ou Paxos. Esses algoritmos ajudariam a tornar o processo de eleição dos líderes muito mais robusto, acabando com a dependência de um único nó para essa função e, assim, tornando o sistema mais tolerante a falhas. Isso significa que, mesmo que um líder falhe ou fique indisponível, o sistema conseguiria se organizar rapidamente, sem perda de controle ou risco de inconsistências.
- Outra melhoria importante é a criação de um dashboard web simples e intuitivo, que permita acompanhar em tempo real o funcionamento do sistema. Hoje, acompanhar o status dos nós e os eventos que acontecem exige um olhar mais técnico e descentralizado, o que pode dificultar decisões rápidas. Um painel visual facilitaria a vida da equipe de operação, permitindo identificar problemas, entender melhor o comportamento do sistema e agir com mais agilidade quando necessário.
- Pensando em ambientes maiores, com necessidade de escalabilidade real, integrar a solução a plataformas de orquestração como Kubernetes é uma evolução natural. Isso possibilitaria que a infraestrutura se adapte automaticamente conforme a demanda cresce ou diminui, otimizando recursos e mantendo o sistema sempre disponível e performático, sem a necessidade de intervenções manuais frequentes.
- Por fim, a persistência estruturada dos logs e dos estados do sistema é um ponto que merece destaque. Guardar essas informações de forma organizada permitiria realizar auditorias completas, entender de forma profunda o comportamento sob diferentes condições e, principalmente, facilitar a identificação de falhas e oportunidades de melhoria com base em dados históricos. Essa análise contínua é essencial para manter a saúde do sistema a longo prazo e garantir uma evolução consistente e segura.

De modo geral, essas extensões futuras representam passos importantes para consolidar a solução, tornando-a não apenas eficiente hoje, mas também preparada para os desafios que podem surgir em ambientes de produção mais críticos e complexos.

Conclusão

O projeto atingiu os objetivos propostos, entregando uma simulação realista e funcional de um ambiente distribuído. A combinação das tecnologias atuais com padrões de projeto e os princípios SOLID resultou em uma solução que é organizada, escalável e segura.

Mais do que apenas uma implementação técnica, esse trabalho foi uma oportunidade prática para entender os desafios e as soluções envolvidas em sistemas distribuídos. As decisões ao longo do desenvolvimento foram feitas com base em critérios técnicos, mas também pensando na clareza para aprendizado e aplicação. Os resultados mostram que a abordagem usada funciona bem e tem potencial para uso em situações reais.

Além disso, o projeto trouxe aprendizados importantes, como a necessidade de ferramentas melhores para monitoramento e de algoritmos mais avançados para eleger líderes, que podem ser foco em futuras melhorias.

Assim, esse projeto não só cumpriu o que foi planejado, mas também serviu como uma boa base para futuros avanços, ajudando a construir sistemas distribuídos mais confiáveis, fáceis de manter e que possam crescer junto com a necessidade.