Relatório Técnico Laboratório III

Disciplina: Programação Distribuída e Paralela
Alunos:
Ana Beatriz Baltazar Boldrini (1-2012338)
Jordian Coutinho Pereira (1-2312738)000000000
Matheus Gothe (1-2121135)
Pedro Miguel Furghieri de Sousa (1-2312739)

1. Introdução

Este relatório descreve a implementação de um sistema distribuído que simula um ambiente de mineração de criptomoedas, utilizando o paradigma de comunicação indireta (Publish/Subscribe). O objetivo principal foi desenvolver um sistema capaz de realizar a descoberta de participantes, eleição de um líder (controlador) e processamento de transações distribuídas (prova de trabalho) sem acoplamento direto entre os processos, utilizando um Broker MQTT.

2. Metodologia de Implementação

A implementação do sistema foi realizada adotando uma arquitetura distribuída e descentralizada, baseada no modelo de troca de mensagens assíncronas (Pub/Sub). O desenvolvimento foi estruturado em três camadas lógicas principais: Gerenciamento de Rede (MQTT), Máquina de Estados (Coordenação) e Processamento (Mineração/Validação).

2.1. Tecnologias e Ferramentas

Linguagem: Python 3, selecionada pela robustez na manipulação de *threads* e facilidade de prototipagem.
Protocolo de Comunicação: MQTT (*Message Queuing Telemetry Transport*), utilizando a biblioteca paho-mqtt. A escolha se deve à sua leveza e eficiência em ambientes de rede instáveis.
Broker: Utilizou-se o *broker* público broker.emqx.io na porta 1883 para intermediar a comunicação sem a necessidade de um servidor central proprietário.
Serialização: Todas as cargas úteis (*payloads*) foram estruturadas em formato JSON, garantindo interoperabilidade e facilidade de *parsing*.

2.2. Estratégia de Desenvolvimento

A lógica do sistema foi implementada seguindo o padrão de Máquina de Estados Finitos. Cada nó do sistema possui um comportamento dinâmico que evolui conforme as mensagens recebidas, garantindo a sincronização entre processos distribuídos sem um relógio global.
Concorrência e Paralelismo: Para evitar o bloqueio do loop de rede durante a mineração (que é intensiva em CPU), utilizou-se a biblioteca threading.
Uma *thread* principal gerencia a lógica de estados.

Uma *thread* de *background* (gerenciada pela biblioteca Paho) escuta o *socket* de rede.
Múltiplas *worker threads* (4 por processo) são disparadas para realizar o cálculo do *hash* (Proof of Work) em paralelo.
Sincronização de Eventos: A comunicação entre a *thread* de rede (Callback on_message) e a lógica principal foi feita através de objetos threading.Event. Isso permitiu, por exemplo, que a rotina de mineração fosse interrompida imediatamente (evt_stop_mining) assim que uma mensagem de "Transação Encerrada" fosse recebida, evitando processamento desperdiçado.
Algoritmo de Eleição: Implementou-se uma variação do algoritmo *Bully*. Para mitigar a duplicidade de código onde todos os nós são idênticos, cada participante gera um VoteID aleatório de 16 bits. O critério de eleição é o maior VoteID, com o ClientID servindo como critério de desempate determinístico.

3. Arquitetura e Fluxo de Execução

O sistema opera em um ciclo contínuo dividido em três fases distintas, onde todos os nós executam o mesmo binário, mas assumem papéis diferentes (Líder ou Minerador) dinamicamente.

3.1. Fase de Inicialização (Discovery)

O objetivo desta fase é a barreira de sincronização.
O nó publica periodicamente seu ClientID no tópico sd/init.
Simultaneamente, escuta o mesmo tópico para construir uma lista de peers_found.
O sistema só transita para a próxima fase quando o tamanho da lista de pares encontrados iguala o parâmetro NUM_PARTICIPANTES configurado, garantindo que o grupo esteja completo antes da eleição.

3.2. Fase de Eleição

Nesta etapa, define-se a hierarquia do *cluster*.
Os nós publicam seus votos aleatórios (VoteID) no tópico sd/election.
Após receber os votos de todos os pares conhecidos, cada nó executa localmente a lógica de decisão: Vencedor = Max(VoteID).
Se Meu_ID == Vencedor, o nó assume a flag am_i_leader = True. Caso contrário, torna-se um *worker*.

3.3. Fase de Operação (Proof of Work)

Esta é a fase de carga de trabalho, onde a comunicação ocorre via tópicos de desafio e solução.O Controlador (Líder): Atua como orquestrador. Gera um TransactionID sequencial e um nível de dificuldade (Challenge). Publica no tópico sd/challenge e aguarda. Ao receber uma solução em sd/solution, recalcula o *hash* SHA-1. Se válido, publica a aprovação em sd/result. O Minerador: Aguarda passivamente. Ao receber um desafio, limpa os eventos de parada e inicia os *workers*. A mineração consiste em gerar *nonces* aleatórios, concatenar com o desafio e calcular o SHA-1 até que o *hash* resultante inicie com o número de zeros exigido (target_prefix).

Tratamento de Concorrência (Race Condition): Um ponto crítico da implementação foi garantir que, se um minerador receber uma mensagem sd/result (indicando que outro nó venceu) enquanto ainda está minerando, ele deve abortar imediatamente sua tentativa para aguardar a próxima rodada.

4. Testes e Resultados

O sistema foi configurado para executar com 3 participantes. Os testes demonstraram o seguinte fluxo:

Sincronização: Observou-se nos logs que os processos aguardaram corretamente até que os 3 nós estivessem conectados e trocassem mensagens de init antes de prosseguir.
Consenso na Eleição: Todos os nós identificaram corretamente o mesmo vencedor com base nos VoteIDs gerados, evitando a existência de múltiplos líderes (split-brain).
Análise da Robustez (Descoberta Implícita): Durante a execução dos testes, foi possível validar na prática o mecanismo de tolerância a falhas descrito na metodologia. Em um dos cenários (conforme observado nos logs), o nó detectou um participante (Peer detectado via Eleição) através de uma mensagem de voto antes mesmo de receber a mensagem de inicialização padrão. Isso comprovou que a arquitetura do sistema foi capaz de lidar com a assincronicidade da rede pública MQTT, mantendo a consistência do grupo e permitindo que a eleição prosseguisse sem travamentos (*deadlocks*), mesmo com a inversão na ordem de chegada das mensagens.

Processamento de Transações: O Controlador gerou desafios (ex: Dificuldade 5) e os publicou. Os Mineradores receberam o desafio simultaneamente. Observou-se a variação no tempo de mineração devido à aleatoriedade do algoritmo de força bruta e ao uso de threads. A validação pelo líder funcionou conforme esperado, rejeitando soluções incorretas (se houvessem) e aceitando a primeira correta, notificando todos os nós via tópico result.

Exemplo de Log de Execução:
Peer encontrado: 12345. Total: 3/3 Eleição concluída. LÍDER: 67890. Sou eu? False Novo desafio recebido! ID: 1, Dificuldade: 5 Mineração concluída em 1.5s. Solução encontrada. Transação encerrada! Vencedor: 12345

5. Conclusão

O laboratório permitiu a aplicação prática de conceitos fundamentais de sistemas distribuídos. A utilização do padrão Pub/Sub desacoplou os componentes, facilitando a escalabilidade (adicionar mais mineradores não exige mudança no código do líder). A implementação da eleição e da coordenação de tarefas demonstrou a importância da sincronização de estados em ambientes onde não há memória compartilhada.