SISTEMAS DISTRIBUÍDOS

RELATÓRIO FINAL

Universidade Estadual Do Sudoeste Da Bahia
Vitória da Conquista - Bahia
Dezembro de 2023

SISTEMA DISTRIBUÍDO COM ALGORITMO DE CONSENSO UTILIZANDO JAVA

Aluno: Vitor Rosenbergre dos Santos Carmo
Matrícula: 201912182
E-mail: vicorintias3@gmail.com
Coordenador da disciplina: Marco Antônio

DESCRIÇÃO DO PROJETO:

O projeto tem como objetivo implementar um sistema de tomada de decisão distribuída em uma rede ponto a ponto (P2P), onde cada nó mantém um conjunto de valores elegíveis chamado "proper" e propostas desses valores.

Conceitos-Chave:

- Proper: Refere-se ao conjunto de valores elegíveis para uma proposta dentro de um nó.
- Proposta: São os valores dentro do conjunto "proper" que cada nó pode possuir e compartilhar com outros nós.
- Decisão: A decisão é tomada com base no valor de proposta de um nó que possui o maior relógio lógico.
- Decisão Final: Representa o resultado final de uma rodada de decisão, baseada nas propostas e nos relógios lógicos dos nós.
- Quorum: Refere-se ao total de propostas recebidas durante o processo de decisão.

Arquitetura do Sistema:

Cada instância representa um nó na rede, onde mantém uma lista de outros nós conhecidos. Utiliza uma fila de mensagens para receber e processar mensagens. Gerencia um mapa de decisões para contabilizar as propostas recebidas.

Comunicação entre Nós:

Os nós se comunicam trocando mensagens, onde as mensagens de tipo "Direction" contêm informações sobre propostas e direções. As propostas são compartilhadas e atualizadas entre os nós durante o processo de tomada de decisão.

Processo de Tomada de Decisão:

Cada nó atualiza seu relógio lógico com base nas propostas recebidas. Após um número predefinido de transmissões, os nós identificam a proposta com o maior relógio lógico. A decisão final é baseada na proposta com o relógio lógico mais alto, refletindo a escolha majoritária.:

O projeto representa uma implementação inicial de um sistema de tomada de decisão distribuída em uma rede P2P, baseado em propostas e relógios lógicos dos nós.
Este trabalho descreve os aspectos fundamentais do projeto, destacando o funcionamento do sistema de tomada de decisão distribuída em uma rede ponto a ponto, baseado em propostas e relógios lógicos dos nós.

IMPORTÂNCIA DIDÁTICA DO TRABALHO:

O trabalho realizado possui grande relevância educacional ao oferecer uma abordagem prática para a compreensão de sistemas distribuídos e algoritmos de comunicação em redes. Ele permite aos leitores explorar conceitos fundamentais, como tomada de decisão distribuída, aplicação de relógios lógicos e programação concorrente.

A implementação prática do mecanismo de tomada de decisão distribuída baseado em votos dos nós possibilita o entendimento dos desafios associados à coordenação entre entidades distribuídas. Além disso, a utilização de relógios lógicos para atribuição e comparação de tempos em mensagens compartilhadas oferece uma aplicação tangível desse conceito teórico. Ao implementar o projeto como uma rede de nós concorrentes em threads separadas, permite o aprendizado sobre sincronização de threads, gerenciamento de concorrência e resolução de problemas típicos desse ambiente, como condições de corrida e deadlocks.

Portanto, é colocado em prática o estímulo ao desenvolvimento das habilidades em resolver desafios complexos inerentes à comunicação em sistemas distribuídos. A experiência obtida nesse projeto contribui significativamente para a formação de competências essenciais para lidar com sistemas distribuídos na prática.

MÉTODOS E TÉCNICAS UTILIZADAS:

O projeto utiliza o método de comunicação conhecido como "Broadcast Roteamento Total Order Broadcast" para facilitar a troca de mensagens entre os nós na rede P2P. Esse método é empregado para garantir a ordem total das mensagens transmitidas para todos os nós da rede.

Para implementar o Broadcast Roteamento Total Order Broadcast, os seguintes métodos e técnicas foram utilizados:

Broadcast de Mensagens:

- A classe “Node” implementa um método de broadcast de mensagens para enviar direções e propostas para todos os nós vizinhos conhecidos na rede. A técnica de broadcasting é utilizada para disseminar as direções escolhidas entre nós, permitindo que cada um atualize suas contagens e propostas conforme necessário.

Controle de Entrega de Mensagens:

- O Broadcast Atômico (*Total Order Broadcast*) garante que todas as mensagens sejam entregues a todos nós na mesma ordem. Isso é crucial para manter a consistência entre nós e garantir que todos tenham uma visão uniforme das propostas e contagens.

Sistema de Votação Distribuída:

- O método baseia-se em um sistema de votação distribuída, onde cada nó mantém sua contagem de votos para determinadas direções. A contagem dos votos é atualizada à medida que as mensagens são transmitidas e recebidas entre nós.

Mecanismo de Reset e Decisão:

- Após um número predefinido de transmissões, o sistema verifica a maioria das direções no mapa de decisões para tomar uma ação baseada na direção majoritária. Após tomar a decisão com base na direção majoritária, o mapa de decisões volta a ter os valores iniciais para começar um novo ciclo de decisões (turno).

Threads e Programação Concorrente:

- A implementação de nós como threads separados permite a execução concorrente de múltiplos nós na rede. Isso facilita a comunicação simultânea entre os nós, refletindo a natureza distribuída do sistema.

Ao aplicar o Broadcast Roteamento *Total Order Broadcast* e as técnicas mencionadas acima, o projeto busca assegurar a transmissão, recebimento e processamento ordenado e consistente das mensagens entre os nós da rede P2P, mantendo a coerência das propostas e contagens em um ambiente distribuído.

DESENVOLVIMENTO:

O desenvolvimento da classe "Node" representa a criação de um nó em uma rede ponto a ponto (P2P) capaz de trocar mensagens com outros nós na rede. A classe é estendida a partir da classe "Thread", permitindo que cada nó seja executado como uma thread separada, facilitando a comunicação concorrente entre os nós.

Os principais métodos e técnicas utilizados na implementação incluem:

Estrutura de Dados:

A classe utiliza estruturas de dados como Listas (List), Filas (Queue) e Mapas (Map) para armazenar informações importantes. O *knownNodes* mantém uma lista dos nós conhecidos na rede, enquanto *messageQueue* é uma fila para gerenciar mensagens recebidas. Mostrado na Imagem 1 abaixo:

```
    // Atributos privados da classe Node
    private final List<Node> knownNodes = new ArrayList<>(); // Lista de nós conhecidos
    private final Queue<Message> messageQueue = new ConcurrentLinkedQueue<>(); // Fila de
mensagens
    private int broadcastsCompleted = 0; // Contagem de transmissões concluídas
    private int totalBroadcasts = 10; // Número total de transmissões desejadas
    private final Map<Direction, Integer> decisionMap = new HashMap<>(); // Mapa de
decisões
```

Imagem 1: Estrutura de dados

Inicialização do Mapa de Decisões:

O método *initializeDecisionMap()* é responsável por inicializar o mapa de decisões (*decisionMap*) com contagens iniciais zeradas para cada direção possível. Mostrado na Imagem 2 abaixo:

```
    // Inicializa o mapa de decisões com contagens iniciais zeradas
    private void initializeDecisionMap() {
        for (Direction direction : Direction.values()) {
            decisionMap.put(direction, 0);
        }
    }
```

Imagem 2: Inicialização do Mapa

Envio de Mensagens de Broadcast:

O método *broadcastMessage()* é fundamental para a propagação de informações na rede de nós. Dentro desse método, a lógica de broadcasting é implementada para enviar mensagens, neste caso, direções, para os nós vizinhos. Ele começa selecionando aleatoriamente uma direção entre as disponíveis. Cada nó escolhe uma direção aleatória (LEFT, RIGHT, UP, DOWN) para tomada de decisão. Isso é feito através da obtenção de um conjunto de direções possíveis e escolhendo uma delas aleatoriamente. Essa direção selecionada será enviada para os outros nós como parte do processo de broadcasting.

Utilizando a lista de nós conhecidos (*knownNodes*), a direção escolhida é enviada para todos os nós vizinhos na rede, exceto para o próprio nó que está enviando a

mensagem. Isso é realizado por meio de um loop que itera sobre a lista de nós conhecidos e envia a direção para cada um desses nós.

Após enviar a direção para os nós vizinhos, o nó atual atualiza seu mapa de decisões. Incrementa a contagem da direção escolhida no decisionMap, registrando quantas vezes essa direção foi enviada nesta iteração de broadcasting. O método que realiza o broadcast é mostrado na Imagem 3 abaixo:

```
public void broadcastMessage(String content, String messageType, String metadata, String format, int size,
                             String qos) {
    if (broadcastsCompleted < totalBroadcasts) {
        Direction[] directions = Direction.values();
        Direction chosenDirection = directions[new Random().nextInt(directions.length)];

        Set<Node> sentNodes = new HashSet<>(); // Conjunto para armazenar os nós que já receberam a direção

        // Enviando a direção escolhida como mensagem para os outros nós conhecidos
        for (Node node : knownNodes) {
            if (node != this && !sentNodes.contains(node)) {
                node.receiveDirectionMessage(this, chosenDirection); // Envia a direção para o nó vizinho
                sentNodes.add(node); // Adiciona o nó à lista de nós para os quais a direção foi enviada
            }
        }

        // Imprimindo a direção para o nó atual
        System.out.println("\nSENDER:");
        System.out.println(getName() + "\n" + chosenDirection);

        // Incrementando a contagem para a posição escolhida apenas para o nó atual
        int currentCount = decisionMap.getOrDefault(chosenDirection, 0);
        decisionMap.put(chosenDirection, currentCount + 1);

        // Restante da lógica de broadcast conforme necessário
        broadcastsCompleted++;
        if (broadcastsCompleted == totalBroadcasts) {
            // Verificar se há maioria de direções antes de resetar o mapa
            Direction majorityDirection = getMajorityDirection();
            if (majorityDirection != null) {
                System.out.println("Maioria de votos para a direção: " + majorityDirection);
                // Realizar a ação com base na maioria das direções antes do reset
                // Faça aqui o que deseja com base na direção majoritária encontrada

                // Aqui, você pode executar a lógica específica com base na direção majoritária
                // encontrada
                // Por exemplo, tomar uma decisão com base nessa maioria antes de resetar o mapa

                // Resetar o mapa de decisões após tomar ação com base na maioria
                resetDecisionMap();
                broadcastsCompleted = 0;
            }
        }
    }
}
```

Imagem 3: Sistema de Broadcast comentado

Tomada de Decisão Baseada em Maioria:

O método *getMajorityDirection()* determina a direção com a contagem mais alta no mapa de decisões, permitindo a identificação da direção majoritária. Mostrado na Imagem 4 abaixo:

```java
// Determina a direção com a contagem mais alta no mapa de decisões
public Direction getMajorityDirection() {
    int maxCount = 0;
    Direction majorityDirection = null;

    for (Map.Entry<Direction, Integer> entry : decisionMap.entrySet()) {
        if (entry.getValue() > maxCount) {
            maxCount = entry.getValue();
            majorityDirection = entry.getKey();
        }
    }

    return majorityDirection;
}
```

Imagem 4: Método para obter o quorum

Processamento de Mensagens:

Esta etapa é vital para a execução contínua do nó. Enquanto a thread não é interrompida, este método mantém o nó em execução. Ele contém um loop que verifica se há mensagens na fila (messageQueue). Se houver mensagens na fila, elas são processadas. Se a mensagem não contiver a rota deste nó (indicando que o nó ainda não a recebeu), ela é processada de acordo com seu tipo. Por exemplo, se a mensagem for do tipo "Direction", a direção é atualizada no mapa de decisões. Caso contrário, a mensagem é encaminhada para nós vizinhos não visitados ainda. Mostrado na Imagem 5 abaixo:

```java
public void run() {
    while (!Thread.currentThread().isInterrupted()) {
        while (!messageQueue.isEmpty()) {
            Message message = messageQueue.poll();
            if (!message.getRoute().contains(this)) {
                if (message.getMessageType().equals("Direction")) {
                    Direction receivedDirection = Direction.valueOf(message.getContent());
                    // Incrementando a contagem para a posição recebida apenas para o nó atual
                    int currentCount = decisionMap.getO int currentCount - Node.run() ;
                    decisionMap.put(receivedDirection, currentCount + 1);
                } else {
                    message.getRoute().add(this);
                    List<Node> nodesForwarded = new ArrayList<>();
                    for (Node node : knownNodes) {
                        if (node != this && !message.getRoute().contains(node) && !nodesForwarded.contains(node)) {
                            node.receiveDirectionMessage(this, message.getDirection());
                            nodesForwarded.add(node);
                        }
                    }
                }
            }
        }
    }
}
```

Imagem 5: Processamento da mensagem

Mapa de Decisões:

Os métodos *resetDecisionMap()* e *printDecisionMap()* são utilizados para reiniciar o mapa de decisões, zerando as contagens, e imprimir o mapa de decisões atual deste nó, respectivamente. Mostrado na Imagem 6 abaixo:

```java
// Reinicia o mapa de decisões, configurando todas as contagens de direções de volta para zero
public void resetDecisionMap() {
    for (Direction direction : decisionMap.keySet()) {
        decisionMap.put(direction, 0);
    }
}

// Imprime o mapa de decisões atual deste nó
public void printDecisionMap() {
    System.out.println("\nDecision Map for Node :");
    for (Map.Entry<Direction, Integer> entry : decisionMap.entrySet()) {
        System.out.println(entry.getKey() + ": " + entry.getValue());
    }
    System.out.println();
}
```

Imagem 6: Mapa de decisão.

CONCLUSÕES, CONSIDERAÇÕES FINAIS:

Ao explorar a dinâmica de decisões e propostas em um sistema distribuído, este artigo apresentou a estrutura e funcionamento de nós interconectados para alcançar consenso em um ambiente descentralizado. Através dos conceitos de "proper", "proposta", "decisão" e "decisão final", foi possível compreender a interação entre os nós e como eles processam informações para chegar a uma conclusão.

O processo de decisão foi delineado com base na avaliação das propostas de nós, onde a seleção é determinada pelo valor de proposta associado a um nó com o relógio lógico mais alto. Essa abordagem dinâmica demonstrou como um nó pode influenciar as decisões finais por meio de suas propostas e como a contagem de quorum, representada pelo total de propostas recebidas, desempenha um papel crítico nesse processo.

A implementação prática do código em Java para ilustrar a troca de mensagens entre os nós, o armazenamento das decisões e a busca por uma maioria de direções proporcionou uma visão tangível de como esses conceitos são aplicados no contexto de sistemas distribuídos.

A análise detalhada dos resultados dos dois turnos de decisão utilizados como amostras revela a evolução do processo de consenso e a dinâmica de escolha das direções predominantes entre 10 nós criados para execução. No primeiro turno (Imagem 7), observamos a troca de mensagens e o aumento progressivo das contagens de direções nos mapas de decisões de cada nó. Isso demonstra o fluxo de propostas e a busca pela direção majoritária.

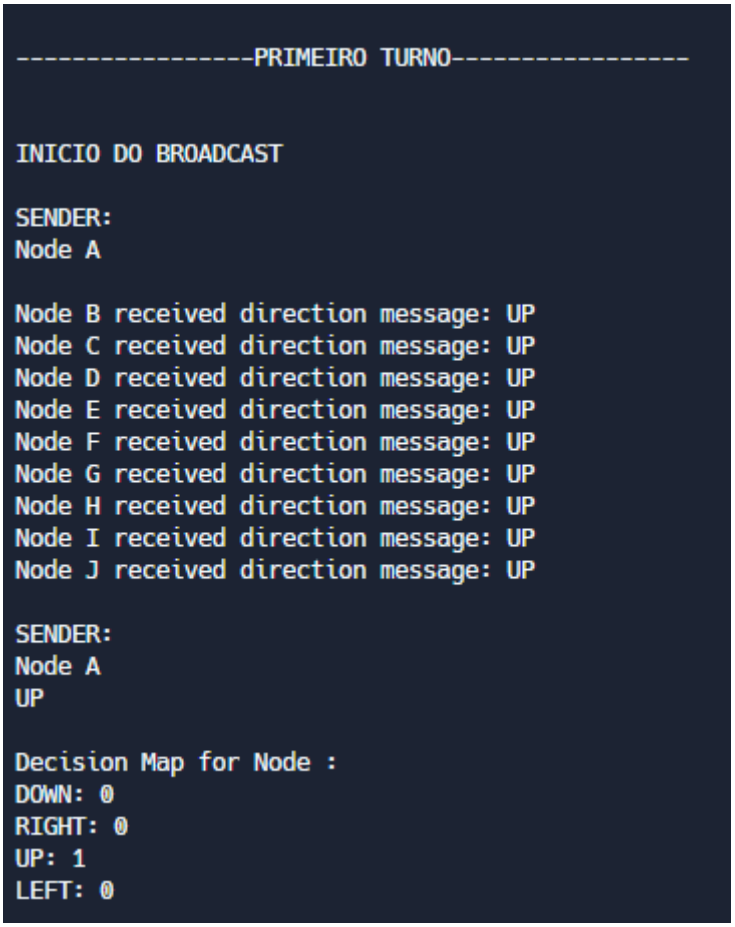

Imagem 7: Broadcast realizado no primeiro turno

Inicialmente, no primeiro turno, as mensagens de "UP" foram predominantes (Imagem 8), chegando a um consenso com quatro dos nós optando por essa direção.

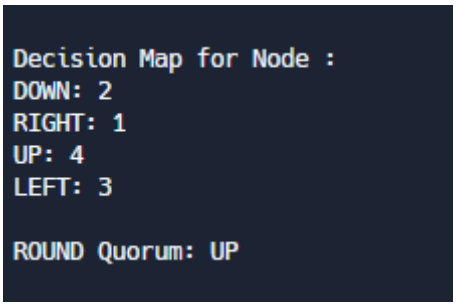

Imagem 8: Quorum do primeiro Turno

No entanto, no segundo turno (Imagem 9), houve uma mudança significativa, com as mensagens de "RIGHT" ganhando predominância, alcançando o consenso com cinco nós adotando essa direção.

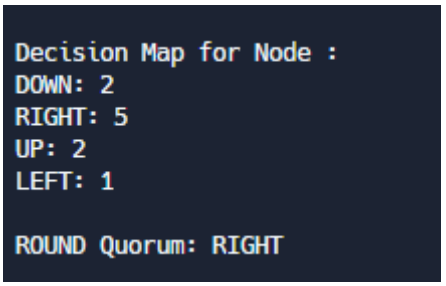

Imagem 9: Quorum do segundo Turno

Essas mudanças refletem a natureza dinâmica do processo de decisão distribuída, onde as propostas são avaliadas e os nós adaptam suas escolhas com base nas mensagens recebidas dos vizinhos. Isso ilustra a importância do quorum e da influência dos relógios lógicos mais altos na determinação das direções predominantes.

O estudo desses resultados reforça a complexidade e a sensibilidade do processo de consenso em sistemas distribuídos, ressaltando a necessidade de estratégias robustas de comunicação e coordenação entre os nós para alcançar uma decisão final com base na maioria. Essa análise oferece insights valiosos para a compreensão dos desafios e das dinâmicas envolvidas na tomada de decisões em ambientes distribuídos.

É crucial ressaltar a importância do controle de fluxo e da sincronização entre nós para garantir a consistência das decisões finais e a integridade do sistema como um todo. O código apresentado oferece uma base sólida para compreensão e experimentação adicional nesse domínio.

Portanto, este projeto destaca a complexidade e a relevância dos processos de decisão em sistemas distribuídos, enfatizando a necessidade de estratégias eficazes de comunicação e coordenação entre os nós para alcançar um consenso confiável em ambientes descentralizados.

REFERÊNCIAS BIBLIOGRÁFICAS:

DENNING, P.J. **Parallel computation**. Am. Sci. Julho-Agosto. 1985.

DOLEV, D., DWORK, C., AND STOCKMEYER, L. 1987. **On the minimal synchrony needed for distributed consensus**. J. ACM 34, Janeiro, 77–97.

FRITZKE, U., INGELS, P., MOSTEFAOUI, A., AND RAYNAL, M. 2001. **Consensus-based fault-tolerant total order multicast**. IEEE Trans. Parall. Distrib. Syst. Fevereiro,147–156´

GUERRAOUI, R. 1995. **Revisiting the relationship between non-blocking atomic commitment and consensus**. In Proc. 9th Intl. Workshop on Distributed Algorithms (WDAG-9). LNCS 972. Springer-Verlag, Le Mont-St-Michel, França, 87–100.

KLEINROCK, L. **On the theory of distributed processing**. In Proceedings of the 22nd Annual Aflerton Conference on Communication, Control and Computing, Univ. of Illinois, Monticello, Outubro. 1964, pp. 60-70.

KIM, J. AND KIM, C. 1997. **A total ordering protocol using a dynamic token-passing scheme**. Distrib. Syst. Eng. 4, 2. Junho, 87–95.

SHLOMI, D., PETIG, T., SCHILLER, E.. **Self-stabilizing and private distributed shared atomic memory in seldomly fair message passing networks**. Algorithmica. Aparece também em CoRR, abs/1806.03498,2022.