Implementação do Algoritmo Paxos

1st Linsmar da Silva Vital
Departamento de Ciência da Computação
UFBA - Universidade Federal da Bahia
Salvador, Brasil
linsmar.vital@ufba.br

2rd Pedro Hugo Passos da Silva Carlos
Departamento de Engenharia Elétrica e de Computação
UFBA - Universidade Federal da Bahia
Salvador, Brasil
pedro.hugo@ufba.br

***Resumo*—Este relatório apresenta a implementação e análise do algoritmo Paxos, um protocolo fundamental para alcançar consenso em sistemas distribuídos. Aborda o problema de consenso, descreve a implementação do Paxos e analisa seu comportamento em cenários com e sem falhas, demonstrando sua robustez e tolerância a falhas. O trabalho oferece insights sobre os desafios e soluções na implementação do Paxos, contribuindo para a compreensão de sua importância na construção de sistemas distribuídos confiáveis.**

***Index Terms*—Algoritmo Paxos, Consenso em sistemas distribuídos, Tolerância a falhas, Implementação, Análise de desempenho**

I. Introdução

Em sistemas distribuídos, alcançar consenso - ou seja, garantir que todos os processos concordem com um único valor ou decisão - é um desafio crucial, especialmente em face de falhas e da natureza assíncrona da comunicação. O algoritmo Paxos, proposto por Leslie Lamport, é um protocolo seminal que aborda esse problema de forma elegante e robusta.

Este relatório documenta a implementação do algoritmo Paxos e explora sua capacidade de alcançar consenso em diferentes cenários. Inicialmente, apresentamos uma breve introdução ao problema de consenso e ao funcionamento do Paxos. Em seguida, detalhamos os aspectos mais relevantes da implementação realizada, destacando os desafios enfrentados e as soluções adotadas.

Para demonstrar a eficácia do algoritmo, analisamos dois exemplos de execução: um cenário sem falhas, onde o Paxos opera em condições ideais, e um cenário com falhas, onde a tolerância a falhas do protocolo é posta à prova. Por fim, discutimos as lições aprendidas ao longo do processo, as dificuldades encontradas e as soluções implementadas para superá-las.

Este trabalho visa não apenas apresentar uma implementação funcional do Paxos, mas também aprofundar a compreensão sobre seus mecanismos e sua importância para a construção de sistemas distribuídos confiáveis.

A. Objetivos

Os principais objetivos deste projeto são:

- **Entender o funcionamento do algortimo de Paxos:** Adquirir um conhecimento profundo sobre o funcionamento do algoritmo de Paxos, incluindo seus papéis, fases, mecanismos de tolerância a falhas e aplicações em sistemas distribuídos;
- **Implementar o algortimo de Paxos:** Desenvolver uma implementação funcional do algoritmo Paxos, capaz de alcançar consenso em um sistema distribuído, lidando com falhas e garantindo a consistência dos dados.

II. Revisão Bibliográfica

A. Algortimo Paxos

O algoritmo Paxos, desenvolvido por Leslie Lamport, é um dos métodos mais influentes e amplamente estudados para alcançar consenso em sistemas distribuídos, especialmente em ambientes sujeitos a falhas e atrasos de comunicação. O Paxos é projetado para garantir que um grupo de processos distribuídos consiga concordar em um único valor, mesmo que alguns dos processos falhem ou mensagens sejam perdidas.

De acordo com Lamport, o problema fundamental do consenso envolve uma coleção de processos que podem propor valores, sendo que o objetivo do algoritmo é garantir que apenas um dos valores propostos seja escolhido, e que os processos possam aprender qual valor foi escolhido, se algum. Os requisitos de segurança do algoritmo incluem que apenas valores propostos possam ser escolhidos, que somente um valor seja escolhido, e que um processo só possa aprender que um valor foi escolhido se ele de fato foi escolhido.

Para alcançar o consenso, o Paxos utiliza três papéis principais: **proposers** (proponentes), **acceptors** (aceitadores) e **learners** (aprendizes). Os proposers são responsáveis por sugerir valores, os acceptors por aceitar propostas, e os learners por aprender o valor que foi finalmente escolhido. Uma característica chave do Paxos é que ele funciona em um modelo assíncrono, onde os agentes operam em velocidades arbitrárias, podem falhar e reiniciar, e as mensagens podem ser entregues com atraso, duplicadas ou perdidas, mas nunca corrompidas.

O Paxos se inicia com um proposer selecionando um número único para sua proposta e enviando uma solicitação de ”prepare”para um subconjunto de acceptors. Se a maioria dos acceptors responde com uma promessa de não aceitar nenhuma proposta numerada inferior a essa, o proposer então envia uma proposta formal contendo o valor que deseja que seja aceito. Os acceptors, por sua vez, podem aceitar a proposta se não tiverem prometido anteriormente rejeitar propostas com números menores. Esse mecanismo de numeração de propostas é fundamental para manter a segurança do sistema,

garantindo que, mesmo que várias propostas sejam feitas simultaneamente, apenas uma possa ser escolhida.

Uma das principais garantias de segurança do Paxos é a propriedade P2, que estabelece que se uma proposta com um determinado valor é escolhida, então todas as propostas subsequentes escolhidas devem ter o mesmo valor. Para manter essa garantia, o algoritmo assegura que se um valor for escolhido, qualquer proposta futura de número maior deve também conter o mesmo valor escolhido, assegurando assim que apenas um valor pode ser escolhido ao longo do tempo

O processo de aprendizado do valor escolhido envolve a comunicação dos acceptors com os learners. Uma abordagem comum é que, ao aceitar uma proposta, os acceptors informam imediatamente os learners sobre o valor aceito, o que permite que os learners saibam o valor escolhido assim que possível. No entanto, para minimizar a complexidade da comunicação, é possível utilizar um learner distinto que coleta as respostas dos acceptors e então distribui o valor escolhido aos demais learners. Esse método diminui a sobrecarga de comunicação, mas torna o sistema menos resiliente a falhas, uma vez que a falha do learner distinto pode atrasar a disseminação do valor escolhido.

Para garantir o progresso do algoritmo, é necessário selecionar um proposer distinto que atue como líder e coordene o processo de proposição de valores. Esse líder deve usar números de propostas que superem quaisquer números anteriormente utilizados, garantindo que suas propostas sejam aceitas. Caso múltiplos proposers tentem liderar simultaneamente, o algoritmo pode entrar em um ciclo de propostas conflitantes, onde nenhum valor é escolhido. Assim, a eleição de um líder é crucial para assegurar o progresso do consenso, embora a segurança do algoritmo seja mantida independentemente do sucesso ou falha do processo de eleição.

A implementação do Paxos descrita por Lamport enfatiza a necessidade de armazenar persistentemente os números das propostas e as promessas feitas pelos acceptors, especialmente em cenários onde falhas e reinicializações são comuns. Esse armazenamento persistente é crucial para manter a integridade do processo de consenso, mesmo após falhas. O algoritmo também apresenta uma série de otimizações, como permitir que proposers abandonem propostas em andamento se perceberem que outra proposta com um número maior está sendo processada, o que ajuda a evitar desperdício de recursos e reduz a latência do sistema.

Além de alcançar consenso em um único valor, o Paxos pode ser estendido para implementar uma máquina de estados distribuída, onde uma sequência de comandos é ordenada e executada de forma consistente em múltiplos servidores. Cada comando é tratado como uma instância separada do algoritmo de consenso, e o líder atribui números de sequência aos comandos, coordenando sua execução ordenada. Essa abordagem permite que sistemas distribuídos, como bancos de dados replicados e sistemas de controle de configuração, mantenham consistência entre os servidores, mesmo na presença de falhas.

Em resumo, o Paxos é um algoritmo altamente robusto para alcançar consenso em sistemas distribuídos, projetado para operar corretamente mesmo em ambientes adversos. A estrutura do algoritmo, com sua separação de papéis e o uso de números de proposta, proporciona uma base sólida para a construção de sistemas tolerantes a falhas, garantindo que a integridade dos dados seja mantida mesmo quando confrontada com falhas de comunicação ou de componentes individuais.

III. Metodologia

Para entender o funcionamento do algoritmo de Paxos, o primeiro passo é adquirir um conhecimento profundo sobre seus princípios e operações. Isso envolve uma pesquisa detalhada, consultando livros, artigos acadêmicos e recursos online para entender os conceitos fundamentais. É essencial analisar os papéis dos participantes no algoritmo, como Propositor, Aceitador e Aprendizado, além de estudar as fases do algoritmo, que incluem Proposição, Aceitação e Aprendizado. Também é crucial compreender como o algoritmo lida com falhas e mantém a consistência dos dados, bem como suas aplicações em sistemas distribuídos.

Após o entendimento teórico, o próximo passo é a implementação prática do algoritmo de Paxos. Primeiro, deve-se elaborar um plano detalhado para a implementação, definindo a arquitetura do sistema e as tecnologias a serem utilizadas. Em seguida, desenvolver a implementação do algoritmo, configurando os papéis de Propositor, Aceitador e Aprendizado e incorporando mecanismos para lidar com falhas e garantir a consistência dos dados. É importante testar a implementação em diversos cenários para validar sua funcionalidade e robustez, realizando testes de consenso para verificar o correto alcance de consenso e testes de tolerância a falhas para assegurar a integridade dos dados durante falhas.

A. Ferramentas Utilizadas

- *Visual Studio Code*;
- Repositório GitHub

IV. Implementação do código

A. Arquitetura Geral

O código é organizado em diferentes módulos, cada um responsável por uma parte específica da simulação. O arquivo `main.py` contém a lógica principal para a criação e gerenciamento dos processos que simulam os nós do Paxos. Esses processos executam a função `PaxosNos`, definida em `paxos_node.py`, que implementa o comportamento de um nó dentro do algoritmo Paxos. A comunicação entre os nós é gerenciada utilizando a biblioteca `zmq` (ZeroMQ), que facilita a criação de sockets para o envio e recebimento de mensagens. Além disso, o código inclui funções para simular falhas de comunicação, essenciais para testar a robustez do algoritmo.

B. Implementação Detalhada dos Módulos

1) `main.py`: Este módulo é responsável por configurar e iniciar a simulação. Ele define o número de nós, a probabilidade de falha nas comunicações e o número de rodadas do algoritmo, a partir dos argumentos de linha de comando. A

função `main` cria uma lista de processos que representam cada nó da simulação, usando a função `criar_processos`, que instancia processos do módulo `multiprocessing`. Esses processos executam o comportamento do nó Paxos de forma paralela e independente.

O `Barrier` da biblioteca `multiprocessing` é utilizado para sincronizar os nós em momentos críticos da execução. Ele força todos os nós a esperar uns pelos outros antes de prosseguir para a próxima fase, garantindo que as operações ocorram de forma coordenada e consistente.

2) `paxos_node.py`: Este módulo contém a função `PaxosNos`, que implementa o comportamento de um nó Paxos. A função gerencia todo o fluxo do protocolo, incluindo a alternância entre as fases de proposição, aceitação e decisão, conforme definido pelo algoritmo Paxos clássico.

Cada nó no Paxos pode atuar como Proposer (Propositor), Acceptor (Aceitador) ou Learner (Aprendiz), dependendo do seu papel na rodada atual. A função utiliza sockets `PULL` para receber mensagens e `PUSH` para enviar, com cada nó se conectando a portas específicas baseadas no seu ID e na constante `BASE_PORT`, definida no módulo `config.py`. Esta organização de portas garante que cada nó saiba exatamente como se comunicar com os outros nós.

A execução do Paxos ocorre em múltiplas rodadas, com cada nó podendo se tornar o Proposer dependendo da rodada e do seu ID. Na primeira fase, o nó Proposer envia uma mensagem de início (`"INICIAR"`) para todos os outros nós. Os Acceptors recebem a mensagem e respondem com suas propostas (`"JOIN"`), indicando se aceitam o round proposto com base nas propostas recebidas anteriormente. Se o Proposer receber respostas suficientes, ele avança para a segunda fase, onde propõe um valor para consenso (`"PROPOSE"`). Os Acceptors então avaliam e votam na proposta, respondendo com `"VOTO"`. Se o Proposer receber votos suficientes, o consenso é alcançado, e o valor é decidido.

3) Comunicação e Falhas: O módulo `communication.py` é essencial para simular a comunicação entre os nós, especialmente em cenários de falhas. A função `enviarComFalha` simula o envio de uma mensagem com uma probabilidade de falha definida. Esta probabilidade é configurada através de um parâmetro e utiliza a biblioteca `numpy` para decidir aleatoriamente se a mensagem será entregue ou substituída por uma mensagem de falha (`"FALHA"`). Esta simulação é crítica para testar como o algoritmo Paxos lida com falhas de comunicação, uma situação comum em sistemas distribuídos reais.

As funções `broadcastComFalha` e `broadcastSemFalha` são utilizadas para enviar mensagens para todos os nós da rede, com ou sem considerar a probabilidade de falha, respectivamente. Este mecanismo de broadcast é usado para garantir que mensagens importantes, como propostas e resultados de votação, sejam distribuídas entre todos os nós participantes.

4) Sincronização e Controle: O uso da `Barrier` é um aspecto importante da sincronização na simulação. Ela é usada para garantir que todos os nós concluam suas operações antes de avançar para a próxima fase ou rodada. Isso é crucial para o funcionamento correto do Paxos, pois todos os nós precisam estar sincronizados para que o algoritmo funcione de maneira consistente.

A introdução de pausas com `time.sleep` ao longo do código serve para simular latências de rede e evitar corridas de condição onde mensagens possam ser recebidas fora da ordem esperada. Isso também ajuda a criar um ambiente de teste mais realista, onde o tempo de comunicação entre nós pode variar.

5) Execução do Código: Para executar o código, o usuário deve fornecer três argumentos na linha de comando: o número de processos (nós), a probabilidade de falha das mensagens e o número de rodadas a serem executadas. A função `main` então gerencia a execução geral do algoritmo, garantindo que todos os processos sejam iniciados corretamente e que o Paxos seja simulado conforme os parâmetros fornecidos.

Se os argumentos de entrada não forem fornecidos corretamente, o script exibe uma mensagem de erro e não prossegue com a execução, assegurando que os parâmetros essenciais para a simulação estejam sempre corretos.

V. Resultados e Discussões

Durante a execução do algoritmo, foram conduzidos múltiplos experimentos variando o número de nós (processos), a probabilidade de falha e o número de rodadas de votação. Os principais resultados incluem:

Atingimento de Consenso: Na maioria das execuções, o algoritmo conseguiu alcançar um consenso entre os nós, especialmente em cenários com baixa probabilidade de falha. Quando todos os nós participaram ativamente das rodadas, o consenso foi alcançado com sucesso, e o valor proposto pelo líder foi aceito pela maioria.

Impacto da Probabilidade de Falha: À medida que a probabilidade de falha aumentou, a frequência de rodadas onde o consenso não foi alcançado também aumentou. Nos cenários de falha elevada (acima de 50%), em muitos casos, nenhum valor foi decidido, pois o líder não conseguiu obter respostas suficientes dos nós para prosseguir com a proposta de valor. Isso confirma a sensibilidade do algoritmo Paxos a falhas de comunicação, ressaltando a necessidade de sistemas robustos de retransmissão ou espera (timeouts).

Mudança de Rodada: Em casos onde o consenso não foi possível em uma determinada rodada (devido a falhas ou discordâncias), o algoritmo executou uma mudança de rodada, onde um novo líder era escolhido para iniciar o processo de consenso. Observou-se que, em rodadas subsequentes, o consenso foi eventualmente alcançado, demonstrando a resiliência do Paxos em ambientes adversos.

Desempenho em Diferentes Configurações de Nós: Com um número maior de nós, o tempo necessário para alcançar o consenso aumentou ligeiramente, devido ao maior número de mensagens trocadas e à maior complexidade em garantir a maioria. No entanto, o Paxos mostrou-se eficiente mesmo em configurações com mais de 10 nós, desde que a probabilidade de falha não fosse excessivamente alta.

Os resultados obtidos indicam que o algoritmo Paxos é efetivo para alcançar consenso em sistemas distribuídos, mesmo na presença de falhas. No entanto, a eficiência do processo depende significativamente da taxa de falhas e da configuração dos nós.

Robustez em Ambientes Reais: Em sistemas reais, onde a falha de comunicação pode ser comum, é essencial implementar mecanismos complementares ao Paxos, como detecção de falhas e retransmissão de mensagens, para garantir que o consenso seja alcançado de forma confiável. Além disso, o Paxos pode ser otimizado para lidar com falhas parciais, onde apenas um subconjunto de nós experimenta problemas de comunicação.

Limitações do Modelo Simulado: Embora o modelo implementado simule com sucesso o processo de consenso, algumas limitações podem ser apontadas, como a ausência de mensagens assincronizadas e a consideração de apenas um tipo de falha (crash). Em sistemas distribuídos reais, outros tipos de falhas, como corrupção de dados ou atrasos significativos nas mensagens, podem ocorrer e precisam ser tratados.

Impacto da Probabilidade de Falha: A análise dos resultados mostrou que, em altas taxas de falha, o algoritmo enfrenta dificuldades em alcançar o consenso, o que levanta questões sobre a viabilidade do Paxos em cenários altamente instáveis. Alternativas como o algoritmo Raft podem ser consideradas em tais situações.

VI. Dificuldades encontradas

Uma das principais dificuldades encontradas foi garantir a sincronização adequada entre os processos simulados que representam os nós. Como cada nó atua de forma independente em processos separados, foi dificil coordenar as rodadas de consenso, especialmente quando há falhas simuladas que atrasam ou interrompem a comunicação. A utilização de barreiras (`Barrier`) ajudou a mitigar parte desse problema, mas encontrar o tempo ideal para sincronização sem comprometer a performance ou a precisão do algoritmo foi um desafio.

Embora o foco principal não tenha sido a otimização de performance, foi percebido que o tempo de execução aumentava significativamente com o número de nós e rodadas. Isso gerou a necessidade de encontrar um equilíbrio entre a precisão do algoritmo e a eficiência do código, especialmente em simulações maiores.

Outra dificuldade foi garantir que o sistema lidasse corretamente com exceções e falhas que poderiam ocorrer durante a execução do código, como problemas de rede ou interrupções imprevistas no processo.

VII. Conclusão

A implementação do algoritmo Paxos neste projeto proporcionou uma compreensão aprofundada dos desafios e complexidades inerentes aos sistemas distribuídos. O Paxos, sendo um dos algoritmos de consenso mais estudados e utilizados, demonstrou sua robustez e eficácia na resolução do problema de consenso, mesmo em ambientes sujeitos a falhas. Em termos de resultados, o algoritmo se comportou conforme o esperado, alcançando consenso na maioria das rodadas e lidando adequadamente com as falhas simuladas. Isso confirmou a teoria de que o Paxos é capaz de funcionar corretamente em um ambiente distribuído, garantindo que uma decisão seja tomada e propagada de forma consistente entre os nós.

Referências

[1] Lamport, Leslie. **Paxos Made Simple**. Novembro de 2001.