Universidade Federal da Bahia - UFBA

Lismar Vital
Pedro Hugo Passos da Silva Carlos

DESCRIÇÃO TÉCNICA DA IMPLEMENTAÇÃO DO ALGORITMO PAXOS

Salvador - Bahia
2024

Contents

1 Visão Geral

O algoritmo Paxos é um protocolo fundamental para alcançar consenso em sistemas distribuídos, especialmente em cenários onde os nós podem falhar ou as mensagens podem ser perdidas. A implementação apresentada simula o Paxos em um ambiente local, usando processos paralelos e comunicação por sockets para ilustrar como os Proposers e Acceptors interagem para alcançar o consenso, mesmo com a presença de falhas simuladas. Abaixo está uma descrição detalhada da implementação, cobrindo a arquitetura do sistema, o funcionamento do algoritmo, as justificativas de projeto, e as instruções para compilar e executar o código.

2 Arquitetura do Sistema

O sistema é implementado usando o algoritmo Paxos distribuído para atingir consenso entre vários nós, mesmo na presença de falhas. A arquitetura do sistema inclui as seguintes partes:

- **Processos Paxos (Nós)**: Cada processo representa um nó Paxos que pode atuar como propositor ou aceitador durante a execução do algoritmo. Cada nó é iniciado como um processo separado utilizando a biblioteca `multiprocessing` do Python.
- **Comunicação entre Nós**: A comunicação entre os nós é gerenciada através de sockets ZeroMQ (`zmq`), onde cada nó cria um socket PULL para receber mensagens e vários sockets PUSH para enviar mensagens a outros nós.
- **Módulos do Sistema**:
  - `main.py`: Controla a inicialização dos processos (nós) e gerencia o fluxo principal do sistema.
  - `paxos_node.py`: Define o comportamento de cada nó Paxos, incluindo as fases do algoritmo Paxos.
  - `communication.py`: Gerencia o envio e broadcast de mensagens entre os nós, com a opção de simular falhas de comunicação.
  - `config.py`: Contém a configuração básica do sistema, como a porta base para comunicação entre os nós.
- **Sincronização**: Utiliza a barreira (`Barrier`) para sincronizar as fases entre os processos/nós.

3 Explicação Passo a Passo do Algoritmo

3.1 Inicialização

- O script `main.py` recebe três argumentos: número de nós (`num_proc`), probabilidade de falha (`prob_falha`) e número de rodadas (`num_rodadas`).
- Uma barreira é criada para sincronizar os nós em cada fase.
- Processos são criados e iniciados para cada nó Paxos com os parâmetros fornecidos.

3.2 Fases do Algoritmo Paxos (Executadas em `paxos_node.py`)

- **Fase de Proposição**: Cada nó pode ser um propositor dependendo da rodada (`r % numProc == id_no`). O propositor envia uma mensagem de "INICIAR" para todos os acceptors via broadcast. Acceptors respondem com "JOIN", indicando sua disponibilidade para participar do consenso, ou "FALHA" se não puderem participar.
- **Fase de Preparação (Join)**: O propositor coleta respostas dos acceptors e decide se deve propor um novo valor baseado nas respostas recebidas. Se a maioria dos acceptors responder com "JOIN", o propositor decide propor um novo valor.
- **Fase de Proposta**: O propositor envia uma mensagem "PROPOSE" para todos os acceptors. Se a proposta for aceita pela maioria dos acceptors, o valor é considerado decidido. Se o valor não for aceito, a rodada é reiniciada com a mensagem "MUDARODADA".
- **Decisão**: O propositor verifica se a maioria dos votos foi recebida e decide o valor proposto. Acceptors respondem com "VOTO" se aceitarem a proposta.
- **Sincronização entre Fases**: As barreiras sincronizam a execução entre todos os processos para garantir que todos estejam na mesma fase do algoritmo antes de continuar.

4 Justificativa das Escolhas de Projeto

- **Uso do Multiprocessing**: Cada nó é representado como um processo separado para simular um ambiente distribuído, garantindo a independência e a paralelização de cada nó.
- **ZeroMQ para Comunicação**: `zmq` é utilizado para comunicação eficiente entre processos com suporte para padrões de comunicação PULL/PUSH, ideal para o modelo de mensagens usado pelo algoritmo Paxos.
- **Simulação de Falhas**: As funções `enviarComFalha` e `broadcastComFalha` permitem simular falhas de comunicação de acordo com a probabilidade de falha definida, replicando cenários reais em sistemas distribuídos.
- **Barreiras de Sincronização**: Uso de barreiras (`Barrier`) para garantir que todos os processos/nós estejam sincronizados na mesma fase antes de continuar, essencial para a execução correta do algoritmo Paxos.

5 Instruções de Compilação e Execução do Código

5.1 Pré-requisitos

- Python 3.x instalado.
- Biblioteca ZeroMQ (`pyzmq`) instalada (`pip install pyzmq`).
- Biblioteca NumPy instalada (`pip install numpy`).

5.2 Execução

1. Salve todos os arquivos do código em um diretório.
2. Abra o terminal na pasta onde os arquivos estão salvos.
3. Execute o comando para rodar o script principal:

```
python main.py <num_proc> <prob_falha> <num_rodadas>
```

6 Conclusão

Esta implementação do Paxos oferece uma simulação realista e detalhada de como o consenso é alcançado em sistemas distribuídos com falhas. Utilizando processos paralelos, comunicação por sockets e simulação de falhas, o código demonstra as complexidades e a resiliência do algoritmo Paxos, sendo uma excelente ferramenta de aprendizado e teste para entender os desafios de alcançar consenso em sistemas distribuídos.