Single-Decree Paxos

Miguel Alves, Nº 49828

1 Descrição de Alto Nível

O protocolo **Single-Decree Paxos** é utilizado para atingir o consenso sobre um único valor, não existindo aqui a noção de replicação de um log entre diversos participantes.

A cada participante no protocolo dá-se o nome de **agente**, e cada agente pode assumir o papel de **Proposer**, **Acceptor** e **Learner** (podendo assumir todos os papéis em simultâneo).

1.1 Proposer

Quando um agente recebe um valor por parte de um cliente, ele começa a agir como **Proposer**.

Começa por enviar uma mensagem de `PREPARE` para todos os **Acceptors** contendo o número de proposta que pretende utilizar, e aguarda pelas mensagens de `PROMISE` dos **Acceptors**.

Ao receber uma maioria de `PROMISES` (podendo estas conter um valor já aceite) procede ao envio de uma mensagem de `PROPOSE` também para todos os **Acceptors**, contendo o número de sequência e o valor associado à proposta (note-se que se alguma das `PROMISE` continha um valor já aceite, este **Proposer** terá que propôr esse mesmo valor).

Se receber uma maioria de respostas de mensagem `ACCEPT` então a sua proposta obteve consenso e pode informar os **Learners** (dependendo da implementação) e responder ao cliente.

1.2 Acceptor

Ao receber uma mensagem de `PREPARE` vinda de um **Proposer**, um agente começa a agir como **Acceptor**. Aqui, o **Acceptor** pode agir de uma de três formas:

- Caso o número de proposta contido na mensagem de `PREPARE` seja o maior que o agente já viu, e caso ainda não tenha aceite nenhuma proposta, o agente responde ao **Proposer** com uma mensagem de `PROMISE`, simbolizando que não irá aceitar nenhuma proposta com menor número.
- Caso o número de proposta contido na mensagem de `PREPARE` não seja o maior que o agente já viu, e caso ainda não tenha aceite nenhuma proposta, ignora a mensagem.
- Caso o agente já tenha aceite uma proposta, responde com uma mensagem de `PROMISE` contendo o número de proposta mais alto que já viu, e o valor que aceitou anteriormente.

Ao receber uma mensagem de `PROPOSE` vinda de um **Proposer**, se o **Acceptor** ainda não aceitou nenhuma outra proposta e o número de proposta incluído no `PROPOSE` é maior ou igual ao maior número de sequência já visto, então ele aceita a proposta respondendo com uma mensagem de `ACCEPT`. Caso contrário, responde com uma mensagem de `REJECT` ou simplesmente ignora a mensagem (depende da implementação).

1.3 Learner

Um **Learner** representa a entidade que fará o ato de "aprender" o valor para o qual foi atingido o consenso. Em sistemas reais os **Learners** são, por exemplo, bases de dados.

Assim, nos artigos sobre o protocolo pouco ou nada é mencionado sobre o comportamento dos mesmos.

2 Estrutura das Mensagens Trocadas entre Agentes

Segue-se a estrutura base das mensagens enviadas. De notar que consoante as implementações e/ou linguagens, as mensagens podem conter mais informação.

- `PREPARE` → número de proposta
- `PROMISE` → número de proposta; valor já aceite (opcional); número de proposta do valor aceite (opcional)
- `PROPOSE` → número de proposta; valor a proposto
- `ACCEPT` → número de proposta; valor aceite
- `REJECT` → número de proposta

3 Etapas do Protocolo

Podemos assim definir os passos/etapas de um fluxo de execução deste protocolo da seguinte forma:

1. Um agente recebe um valor por parte de um cliente, e começa a agir como **Proposer**
2. Esse **Proposer** incrementa o seu número de proposta e envia uma mensagem de `PREPARE` para todos os **Acceptors**
3. Cada **Acceptor**, ao receber a mensagem de `PREPARE`:
   - Caso não tenha ainda aceite uma proposta, e o número de proposta seja o maior que já viu até agora, responde com uma mensagem de `PROMISE` contendo o número de proposta recebido no `PREPARE`
   - Caso o número de proposta não seja o maior que já viu até agora, ignora a mensagem recebida
   - Caso já tenha aceite uma outra proposta, responde com uma mensagem de `PROMISE` contendo o número de proposta e o valor aceite
4. Ao receber uma mensage de `PROMISE` vinda de um dos **Acceptors**, o **Proposer**:
   - Começa por verificar se a `PROMISE` inclui um valor já aceite. Se sim, então o **Proposer** altera o valor que pretende propôr para ser igual ao valor já aceite
   - Verifica se já recebeu uma `PROMISE` de uma maioria dos **Acceptors**, se sim passamos para o passo seguinte do protocolo, caso contrário mantemo-nos neste passo
5. Ao receber uma `PROMISE` de uma maioria dos **Acceptors**, o **Proposer** envia uma mensagem de `PROPOSE` para todos os **Acceptors**
6. Ao receber uma mensagem de `PROPOSE` vinda de um **Proposer**, um **Acceptor**:
   - Se o **Acceptor** já aceitou alguma proposta, ou o número de sequência da proposta é menor do que o maior número visto até ao momento, então responde com uma mensage de `REJECT`
   - Se as condições anteriores não se verificarem, então o **Acceptor** aceita a proposta, respondendo com uma mensagem de `ACCEPT`
7. Finalmente, ao receber uma mensagem de `ACCEPT` de uma maioria dos **Acceptors**, o **Proposer** informa o cliente e os **Learners** de que foi atingido o consenso.

4 Implementação em Rust

Para a implementação em **Rust**, foram feitas as seguintes escolhas de implementação:

- Cada agente é simulado por uma thread
- A troca de mensagens é feita com recurso ao módulo std::sync::mpsc
- Adicionou-se uma probabilidade (ajustável) de uma mensagem não ser enviada
- Como os números de proposta de diferentes agentes devem ser diferentes, cada agente tem um id associado e o seu número de proposta é inicializado a id/1000 (sendo que este valor deve ser ajustado de acordo com o número de agentes utilizados)

O mapeamento entre as etapas do protocolo e as funções em Rust é o seguinte (usando a mesma numeração que em ***3***):

1. `agent.rs` → na função `run` recebe-se uma mensagem com tipo `BEGIN`
2. `proposer.rs` → após a receção do `BEGIN` é executada a função `snd_prepare`
3. `acceptor.rs` → após a receção do `PREPARE` é executada a função `rcv_prepare` a qual processa o `PREPARE` e executa a função `snd_promise`
4. `proposer.rs` → após a receção de uma `PROMISE` é executada a função `rcv_promise`
5. `proposer.rs` → ao atingir uma maioria de `PROMISE` é executada a função `snd_propose`
6. `acceptor.rs` → ao receber uma mensagem de `PROPOSE` é executada a função `rcv_propose` e consoante as condições é executada a função `snd_reject` ou `snd_accept`
7. `proposer.rs` → ao receber uma mensagem de `ACCEPT` é executada a função `rcv_accept` no final da qual é verificado se foi recebida uma maioria de `ACCEPT` e em caso afirmativo é executada a função `inform_learners`

5 Testes e Resultados

// **TODO** (quais os testes realizados; comparação de tempos de execução com o aumento do número de agentes e aumento do número de propostas concorrentes, etc.)

6 Fontes

Paxos Made Simple, Leslie Lamport, https://lamport.azurewebsites.net/pubs/paxos-simple.pdf