CSE535 Asynchronous Systems
Paxos

YoungMin Kwon

Paxos

- Paxos: a crash-fault tolerant distributed consensus algorithm in asynchronous communication channels

The Consensus Problem

- What about Fisher’s impossibility result of a crash-fault tolerant consensus in asynchronous communication channels?

- Paxos satisfies the safety requirements, but not the liveness requirements.

The Problem

- The consensus problem
  - A collection of processes can propose values
  - A consensus algorithm ensures that a single one among the proposed values is chosen
  - If no value is proposed no value is chosen
  - If a value is chosen, processes should be able to learn the chosen value

The Problem

- Safety Requirements
  - Only a value that has been proposed may be chosen
  - Only a single is chosen
  - A process never learns that a value has been chosen unless it actually has been
- Liveliness
  - The algorithm may not terminate

Three Classes of Agents

- Proposer
  - Proposes a value
- Acceptor
  - Accepts a proposed value
  - A value is chosen when a majority of acceptors accept the value
- Learner
  - Learns the chosen value

- A single process may act as more than one agent

Choosing a Value

- A Naïve solution
  - Have a single acceptor and let it choose a value
  - Failure of the agent will stop the protocol

- Have multiple acceptors
  - An agent may accept a proposed value
  - A value is chosen when a majority of acceptors accept the value

Choosing a Value

- Extended proposal
  - To keep track of different proposals, a proposal is extended with a proposal number and a value
  - Different proposals have different numbers
- 3 Message types
  - Prepare(n): request acceptors not to accept proposals whose number is less than n
  - Promise(n, m, v): response to proposers that the acceptor won't accept any proposals less than n; m, v are from the accepted proposal if it already did
  - Accept(n, v): request accepts to accept the proposal with the number n and the value v

Choosing a Value

- Phase1 (Proposer)
  - Selects a proposal number n and sends Prepare(n) to a majority of acceptors

- Phase1 (Acceptor: on receiving Prepare(n))
  - If n > k then send Promise(n, m, v), where
    k: the highest proposal number it has promised
    m, v: the number and value of the accepted proposal if it already accepted one

Choosing a Value

- Phase 2 (Proposer)
  - If Promise(n,m_i,v_i) is received from the majority of acceptors, send Accept(n,v) where
    v: v_i of the highest m_i or any value if all m_i, v_i are invalid

- Phase 2 (Acceptor: on receiving Accept(n,v))
  - Accepts the proposal unless it already sent Promise(m,k,u) for $m > n$

Choosing a Value

- A proposer can make multiple proposals
- A proposer can abandon a proposal at any time

- An acceptor can ignore prepare or accept requests because it already promised for a higher number
  - However, sending reject messages to the proposers will speed up the protocol

Learning a Chosen Value

- To learn that a value has been chosen, a learner must find out that a proposal has been accepted by a majority of acceptors

- An algorithm
  - Make acceptors send messages to all learners every time they accept a proposal
  - # of messages: # of acceptors times # of learners

Learning a Chosen Value

- Improved Algorithm
  - Make acceptors send messages to a set of designated learners
  - The designated learners send message to other learners only when a value is chosen
- Message loss
  - A value could be chosen with no learner finding out
  - A learner can make a proposer propose so that the chosen value can be announced again

Progress

- A scenario where the protocol does not end
  - Two proposers keep issuing proposals and none of which are ever chosen
  - p_1 sends Propose(n_1) to acceptors
  - In between acceptors send Promise(n_1,m_i,v_i) and receive Accept(n_1,v_1) from p_1,
    p_2 sends Promise(n_2) to acceptors with $n_2>n_1$
  - In between acceptors send Promise(n_2,n_1,v_1) and receive Accept(n_2,v_2) from p_2,
    p_1 sends Promise(n_3) to acceptors with $n_3>n_2$
  - And so on

Progress

- To guarantee progress,
  - A distinguished proposer must be selected as the only one to try issuing proposals
  - However, the impossibility result by Fisher et al attests its unfeasibility
  - Randomized or real time (using timeouts) algorithms can ensure the progress