Distributed Systems

Assignment 3 Review

David Domingo
Paul Krzyzanowski

Rutgers University

Fall 2018

Paxos summary

- Paxos is a fault-tolerant distributed consensus algorithm
  - A collection of systems have to agree on **exactly one** proposed value
- **Proposers**
  - Receive requests from clients
  - Send them to acceptors to choose requested values
  - We typically choose one proposer to handle all requests, called a ***Leader***
- **Acceptors**
  - Respond to requests from proposers
  - Store state about the highest accepted proposal
- **Learners**
  - Propagate info about the value chosen by acceptors
  - (We often ignore this and let the *Leader* do the work)

Paxos algorithm summary

- **Prepare Phase**
  - **Proposer**
    - Contact all acceptors with a *PREPARE(proposal #, value)* message
  - **Acceptor**
    - If this is the highest or only proposal # the acceptor has seen:
      - Acceptor promises it will not accept proposals with smaller numbers
      - Save the proposal # and value (in case another *prepare* message comes later).
    - If the acceptor already **accepted** a proposal (see phase 2)
      - Acceptor returns the highest proposal # and value it has accepted
- **Accept Phase**
  - **Proposer**
    - Waits for responses from a majority of acceptors – checks if any acceptors returned an *accepted* proposal
      - If yes - picks value associated with the highest proposal # returned from any acceptor
      - If no - use the original value that was proposed
      - Send an *ACCEPT(proposal #, value')* message to all acceptors
  - **Acceptor**
    - Compares received proposal # with the highest # it has seen
      - **Reject** proposal if the received proposal is # not as high
      - Otherwise **accept** proposal – remember proposal # and value in case we need to return it if we receive a *PREPARE* message from someone else
    - Return the **current value** of the highest proposal that was accepted
  - **Proposer**
    - Receive majority of responses from a majority of acceptors.
    - See if any have been rejected. If rejected, the proposer would have to start again with a higher proposal #
    - If all accepted the request, then we are done.

Paxos summary

Two-phase protocol: started by a *proposer*

- Phase 1: prepare
  Send ***prepare*** request to all acceptors
  - Acceptor will return the information about the highest proposal it has accepted (if any)
    - Allows proposer to find out if any other values have been chosen so we use that value instead
  - Acceptor promises it will never accept a proposal number with a lower request (blocks older proposals)
- Phase 2: accept
  - A proposer collects responses from all live acceptors
  - If a majority of acceptors respond that they agree on this value, then it is chosen by the proposer
  - Proposer sends an ***ACCEPT*** message to all acceptors

Paxos summary

Why use proposal numbers?

- If all requests come from one proposer (leader) then Paxos is trivial
  - We would simply send a message to all the acceptors we can reach
  - Get a response from the majority
- A leader can fail – Paxos handles the case where multiple proposers might think they are the leader
  - Multiple proposers will not lead to inconsistencies
  - Each proposer uses a unique proposal #
  - Proposals are ordered: newer (higher #) proposals take precedence over older ones
  - Acceptor tells it whether it has already accepted a higher numbered proposal
- Why do we need a majority of acceptors?
  - Once a value has been accepted by a majority of acceptors, if any acceptor crashes, at least one acceptor still has the latest (highest) state.

Question 1 (Paxos)

Why can an acceptor not necessarily accept the first value it receives but must sometimes accept different values?

[answer from the John Ousterhout video]

There might not be a majority of proposed values to determine a winner.

For example:

- 2 acceptors might have value *A*
- *2* acceptors might have value *B*
- *1 acceptor* might have value C

no majority!

Therefore, there won’t be one value that all servers can agree on as the majority value.

⇒ An acceptor has the right to change its mind.

A value that has been *accepted* does not mean it is ultimately *chosen*. It just means that it’s the highest numbered proposer that one acceptor has seen so far. It is only chosen once we have a **majority** of acceptors.

First check for existing proposed values. Reject older proposals (each proposal has a proposal number) received after newer ones.

Question 1 – Discussion

Why can an acceptor not necessarily accept the first value it receives but must sometimes accept different values?

If each acceptor just accepts a proposed value, it is possible that no acceptors get a majority of any proposed value

- Acceptors therefore have to be able to accept different values – they may have to change their mind
- They cannot accept *every* proposed value because then multiple values could be chosen
- Once a value has been chosen, a new proposer has to abandon its value and use a previously chosen value
  - We need a 2 phase protocol: phase 1 asks the acceptor for chosen values before proposing a value
  - Any competing proposals have to be aborted
  - This is done by forcing an order: higher numbered (newer) proposals will take precedence over lower-numbered (older) proposals

Question 2 (Paxos)

When does a proposer have to change the value that it is proposing during the Paxos consensus protocol?

- A proposer sends a value to an acceptor (with a *prepare* message)
  - Multiple proposers may do this concurrently and send different values
- Acceptors respond to a *prepare* request from a proposer with the highest numbered proposal that they accepted if another proposal has already been accepted
  - If multiple requests came in concurrently, an acceptor may have seen a higher number. It responds to each proposer with that higher number
- A proposer **must** ask for that value to be accepted even if it initially proposed a different value.
  - **The proposer is the one who figures out the highest accepted proposal from all acceptors and propagates that information to all acceptors**.
  - This does not violate the requirement of consensus since the algorithm selects one of the proposed values.

Question 3 (Raft)

Raft uses a single leader (one server is elected as a leader).
Explain how Raft performs leader election.

Short answer:

Each candidate starts a random timer before proposing itself as a leader & sending election messages to the group.

If you receive a *leader proposal* message and you have not yet proposed yourself, you will acknowledge that candidate and not vote for yourself.

If a candidate gets majority votes, it becomes the leader.

Question 3 – Longer Answer

Raft uses a single leader (one server is elected as a leader). Explain how Raft performs leader election.

To start an election, a candidate votes for itself and sends a *request vote* message to all other servers. Other servers that have not yet voted and receive the request acknowledge the candidate to be the leader. Each server that receives a request will vote for at most one candidate.

If the candidate receives a majority of acknowledgements, it becomes the leader.

If the candidate does not win or lose an election, it times out and starts a new election. Randomized timeouts are used to ensure that split votes happen rarely.

To support recovery and avoid stale state, a "term number" is incremented after each election

> If the candidate receives a heartbeat from another server and that leader's term # is at least as large as the candidate's current term, then the candidate recognizes the leader as legitimate and becomes a follower.

Question 4 (Raft)

An elected leader takes client requests. Each request is essentially a log entry that will be replicated among the servers. When is a log entry committed in Raft?

A log entry is committed once the leader that created the entry has replicated it on a majority of the servers.

Committed means that the log entry is applied to the state machine.

The End