Optimizing and Implementing Paxos

Yang Wang
Review: Basic Paxos

• **Termination**: Every correct process decides some value when there are no asynchronous events.

• **Validity**: If all processes propose the same value $v$, then all correct processes decide $v$.

• **Integrity**: Every correct process decides at most one value, and if it decides some value $v$, then $v$ must have been proposed by some process.

• **Agreement**: Every correct process must agree on the same value.

• Validity and integrity are trivial in fail-stop model. Let’s focus on termination and agreement.
Review: Basic Paxos

Client

Server

Prepare
Accept
Learn
Review: Basic Paxos

This is equivalent to the previous one with less latency but more messages
Acceptor and learner

• Revise state machine replication
  – Model application as a deterministic state machine
    --- This part is called a learner in Paxos
  – Run a consensus protocol to decide the next request --- This part is called an acceptor in Paxos

• A replica is logically separated into an acceptor and a learner
  – Many implementations collocate them in a single process
Optimization 1: Elect a leader

• Problem: if the system has multiple clients, conflicts (two clients propose at the same time) may be frequent

• Optimization: elect one server as the leader and ask all clients to send requests to the leader. Only the leader makes proposals.
Optimization 1: Elect a leader

- Problem: if the system has multiple clients, conflicts (two clients propose at the same time) may be frequent.
- Optimization: elect one server as the leader and ask all clients to send requests to the leader. Only the leader makes proposals.
- Wait. How is it different from Primary Backup?
Optimization 1: Elect a leader

• Primary backup: if there are more than one leaders, agreement may be violated
• Paxos: if there are more than one leaders, agreement will not be violated. Termination may be violated when there are asynchronous events.
Optimization 1: Elect a leader

• Primary backup: if there are more than one leaders, agreement may be violated
• Paxos: if there are more than one leaders, agreement will not be violated. Termination may be violated when there are asynchronous events.
• Accurate failure detection = ensure there is at most one leader = solve consensus
Optimization 1: Elect a leader

• How to elect the leader?
  – There are multiple solutions.

• Simplest solution: round robin
  – Server 1 is the first leader. If it fails, server 2 becomes the leader, and then server 3, ...
Optimization 1: Elect a leader

• When to elect a leader?
  – When the current leader fails, but we don’t know.

• Simplest solution: timeout
  – Timeout may be inaccurate, so multiple leaders may be elected, but that is fine.
  – For termination, use a exponentially growing timeout
Optimization 2: Multi Paxos

• So far, we have talked about how to decide the next request
• A real application needs to execute a sequence of requests, instead of just one
• Naive solution: run basic Paxos multiple times
  – Divide execution into multiple slots
  – Use Paxos to decide a request for each slot
  – (Prepare, Accept, Learn) for slot 1, and then for slot 2, ...
Optimization 2: Multi Paxos

• Optimization: only need to run “Prepare” once
  – Prepare, Accept for slot 1, Learn for slot 1, Accept for slot 2, Learn for slot 2, ...

• Revise what Prepare does:
  – Proposer needs to know what has been agreed.
  – Proposer needs an acceptor to promise not to agree on earlier proposals.
Optimization 2: Multi Paxos

• Optimization: only need to run “Prepare” once
  – Prepare, Accept for slot 1, Learn for slot 1, Accept for slot 2, Learn for slot 2, ...

• Revise what Prepare does:
  – Proposer needs to know what has been agreed.
  – Proposer needs an acceptor to promise not to agree on earlier proposals.

• Instead of “prepare” for one request, we can prepare for 100 requests, or more
Multi Paxos

• Each leader is assigned a unique number (epoch number)
  – Simplest approach: round robin

• After a leader is elected, it sends the prepare messages to all acceptors
  – Ask for all requests that have been agreed.
  – Ask acceptors to promise that they will not accept proposals from earlier leaders.
Multi Paxos

Client

Prepare  Accept  Learn  Accept  Learn

Leader

Server

Server
Pipeline execution

• To reduce latency, a leader can propose the next request before the previous one is agreed
• Problem 1: learner must execute requests in order
  – Solution: learner maintains the last slot it has executed.
• Problem 2: after a leader election, during the prepare phase, the new leader may find slot 100 is already agreed while slot 99 is not.
  – Solution: the new leader can propose a special “noop” operation for slot 99
Failure recovery

- If an acceptor or a learner is destroyed, we will need to replace it with a new server
- The server needs to know what requests have been agreed (and record or execute them)
- It can use a protocol similar to Prepare
Garbage collection

• An acceptor needs to remember requests that it has agreed.
  – Otherwise, failures may cause requests to be lost.
  – The log may grow arbitrarily.

• Periodically, the system asks a learner to take a snapshot of its state machine
  – By doing so, the learner promises that it will never need earlier requests.
  – Then the acceptor can delete those requests.
Implementation details

• A client may have multiple outstanding requests
  – How can it match replies with requests? Is TCP sufficient?
Implementation details

• A client may have multiple outstanding requests
  – How can it match replies with requests? Is TCP sufficient?
  – Solution: generate a unique ID for each request
  – A classic approach: requestID=(clientID,noRequests)
Implementation details

• A client may have multiple outstanding requests
  – How can it match replies with requests? Is TCP sufficient?
  – Solution: generate a unique ID for each request
    – A classic approach: requestId=(clientId,noRequests)

• Leader should not propose a request more than once
  – How could this happen?
Implementation details

• A client may have multiple outstanding requests
  – How can it match replies with requests? Is TCP sufficient?
  – Solution: generate a unique ID for each request
    – A classic approach: requestID=(clientID,noRequests)

• Leader should not propose a request more than once
  – How could this happen?
  – Solution: a leader should remember the last requestID it has proposed for each client
Implementation details

• What to do if a client does not get the reply in time?
Implementation details

• What to do if a client does not get the reply in time?

• Possible reasons:
  – Request to the leader is lost
  – Leader receives the request but it fails before it proposes the request
  – Leader proposes the request but it fails before the proposal is agreed by f+1 replicas
  – The proposal is passed but the message to the learner is lost
  – Learner executes the request but the reply to the client is lost
  – ...

Implementation details

• What to do if a client does not get the reply in time?

• Solution: the client resends the request
  – A client should remember all outstanding requests
Implementation details

• What to do if a client does not get the reply in time?

• Solution: the client resends the request
  – A client should remember all outstanding requests

• Leader:
  – If never proposed the request, propose it
  – If already proposed, but not stable, just wait
  – If already stable, send it to the learner
Implementation details

• What to do if a client does not get the reply in time?

• Solution: the client resends the request
  – A client should remember all outstanding requests

• Learner:
  – If never executed it, execute it (but need to follow order)
  – If has already executed, how? Can it execute the request again?
Implementation details

• What to do if a client does not get the reply in time?

• Solution: the client resends the request
  – A client should remember all outstanding requests

• Learner:
  – If never executed it, execute it (but need to follow order)
  – If has already executed, send the previous reply to clients
    • To doing so, a learner needs to remember replies sent to clients
Implementation details

• Reply cache: Learner should remember replies to clients

• Limit the size of reply cache
  – Solution 1: limit the number of outstanding requests per client (e.g. 100): if learner receives request 101, it knows the client must have received the reply for request 1
  – Solution 2: client piggybacks the received reply ID in its requests, so that learner can know the information

• Reply cache is critical for correctness
  – Snapshot of a learner should include the reply cache