EECS 491
Introduction to Distributed Systems

Fall 2019

Harsha V. Madhyastha
RSM via Consensus

- **Idea:** Apply update if majority of replicas commit
  - If $2f+1$ replicas, need $f+1$ to commit
  - Leverage overlap between any pair of majorities

- **First:** Among several concurrent new updates, how to pick next update to apply?

- **Later:** How to apply all updates in a consistent order at all replicas?
Paxos: High-level Intuition

- Protocol runs over multiple rounds
- In each round:
  - Elect a leader
  - Proposal by current leader accepted by majority
- Once a value is accepted by a majority, a different value won’t be proposed subsequently
Let’s plan a camping trip!

- Before going away on internships, three friends plan on going camping.

- Can only coordinate via unreliable text messages.

- How to decide on a camp to meet at?
Example execution of Paxos

- Alice and Bob both propose to be leader; both fail
- Alice re-proposes to be leader and succeeds
- Bob re-proposes to be leader and succeeds
- Alice proposes Yosemite; rejected (not leader)
- Alice re-proposes to be leader and succeeds
- Alice proposes Yosemite; accepted by majority
- Bob re-proposes to be leader and succeeds
- Bob proposes Yosemite; accepted by majority
Paxos Overview

- Three conceptual roles for each process
  - Proposers propose values
  - Acceptors accept values; chosen if majority accept
  - Learners learn the outcome (chosen value)

- Three phases within each round:
  - Prepare phase: elect a leader
  - Accept phase: leader gets majority to accept
  - Learn phase: disseminate chosen value
Paxos State

- Every acceptor maintains three values:
  - \( n_p \rightarrow \) highest proposal number promised to accept
  - \( n_a \rightarrow \) highest proposal number accepted
  - \( v_a \rightarrow \) value accepted

- For now: This state must persist across restarts
Paxos Phase 1

- **Proposer:**
  - Choose unique proposal number n
  - Send <prepare, n> to all acceptors

- **Acceptors:**
  - If $n > n_p$
    - $n_p = n$ ← promise not to accept any new proposals $n' < n$
    - If no prior proposal accepted
      - Reply < promise, n, Ø >
    - Else
      - Reply < promise, n, $(n_a, v_a)$ >
  - Else
    - Reply < prepare-failed >
Prepare Phase

\[ n_p = 2 \]

\[ n_p = 2 \]

\[ n_p = 3 \]

\[ n_p = 3 \]
Paxos Phase 1

- **Proposer:**
  - Choose unique proposal number $n$
  - Send $\langle$prepare, $n$$\rangle$ to all acceptors

- **Acceptors:**
  - If $n > n_p$ Update $n_p$ even if previously accepted proposal
    - $n_p = n$ ← promise not to accept any new proposals $n' < n$
    - If no prior proposal accepted
      - Reply $\langle$ promise, $n$, $\emptyset$$\rangle$
    - Else
      - Reply $\langle$ promise, $n$, $(n_a, v_a)$$\rangle$
  - Else
    - Reply $\langle$ prepare-failed $\rangle$ ← What else is worth including?
Paxos Phase 2

- **Proposer:** When would majority not promise?
  - Once received promises from majority of acceptors,
    - \( v' = v_a \) returned with highest \( n_a \), if exists, else own \( v \)
    - Send \(<\text{accept}, (n, v')>\) to acceptors
  - Why not stop if \( v_a \neq v \)?

- **Acceptors:**
  - Upon receiving \((n, v)\), if \( n \geq n_p \),
    - Accept proposal
      - \( n_a = n_p = n \)
      - \( v_a = v \)
  - Accept even if previously accepted a proposal

September 26, 2019
Accept Phase

$P_1(v_1)$

$n=1$

$v=v_1$

$A_1$

$A_2$

$A_3$

$P_2(v_2)$

$n=2$

$v=v_1$

September 26, 2019

EECS 491 – Lecture 8
Accept Phase

\[ P_1 (v_1) \quad A_1 \quad A_2 \quad A_3 \quad P_2 (v_2) \]

- \( n=1 \)
- \( v=v_1 \)
- \( n=2 \)
- \( v=v_2 \)

September 26, 2019
Accept Phase

P₁ (v₁)

n=1

v=v₁

A₁

A₂

A₃

P₂ (v₂)

n=2

v=v₁
Paxos

- **Prepare Phase**
  - Get promises from a majority of acceptors
  - Discover accepted values, if any

- **Accept Phase**
  - Propose *either own value or accepted value with highest proposal number*

- Retry Prepare phase if either phase fails
Paxos Phase 3

- **Goal:** For all learners to discover if any value was accepted by majority

- **Potential approaches:**
  - Successful proposer informs all learners
  - Acceptors broadcast state updates to all learners
  - Acceptors notify distinguished learner, which informs others

- What is a fault-tolerant and scalable solution?
Paxos Phase 3

- Learners mimic proposers
- Discover value accepted by each acceptor in response to prepare messages
Paxos: Well-behaved Run

<prepare, 1>  

<promise, 1>  

<accept, (1, v₁)>  

<accepted, (1, v₁)>  

decide v₁
Desired Properties of Solution

- **Safety:**
  - Choose a proposal only if accepted by a majority
  - Choose from proposals made

- **Liveness:**
  - If proposals exist, one will eventually be chosen
  - If a proposal is chosen, all replicas will eventually discover that it was chosen
Paxos is safe

- Intuition: if proposal with value \( v \) accepted, then every higher-numbered proposal issued by any proposer has value \( v \).

Majority of acceptors accept \((n, v)\): \( v \) is decided

Next prepare request with proposal \( n+1 \)
Desired Properties of Solution

- **Safety:**
  - Choose a proposal only if accepted by a majority
  - Choose from proposals made

- **Liveness:**
  - If proposals exist, one will eventually be chosen
  - If a proposal is chosen, all replicas will eventually discover that it was chosen
Race condition leads to liveness problem

Process 0
- Completes phase 1 with proposal n0
- Performs phase 2, acceptors reject
- Retries and completes phase 1 with proposal n2 > n1

Process 1
- Starts and completes phase 1 with proposal n1 > n0
- Performs phase 2, acceptors reject

... can go on indefinitely ...

How to fix this?
Fixes to liveness problem

- When proposal fails, **back off for a random period of time** before retrying

- Pre-determined ordering of proposers
  - Negative response from acceptor includes ID of proposer to whom the acceptor has committed
  - Back off period chosen based on ordering

- Note co-operative nature of protocol
Announcements

- Try questions for tomorrow’s discussion section
  - Review this week’s lecture slides and recordings

- Acknowledgment rule in Project 2
  - View service maintains a single view at all times
  - Whether current primary has ACKed current view only dictates whether view service can change view
  - Cannot change view until primary syncs with backup