Availability of P/B-based RSM

- When is RSM unavailable to serve requests?

- Replica is down but viewservice yet to detect

- How to …
  - … make RSM tolerant to network partitions?
  - … ensure that operations don’t block even if some machines are unavailable?
Analogy

- US Senate needs to pass laws
- Senators are often on travel
  - Common case: Not all senators present
- How to pass laws successfully?
**RSM via Consensus**

- **Key idea:** Apply an update if *majority of replicas* commit to it
- If 2f+1 replicas, need f+1 to commit

- Why majority? Why not fewer or more?
- Remaining replicas cannot accept some other update
Context for Today’s Lecture

- Say all replicas are in sync with each other

- **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?
Strawman Approaches

- Every client proposes its value to all replicas
- Every replica accepts first proposal received
- Value accepted by majority is applied
- Why might this not work?

- Every client tags its proposal with seq number
- Every replica collects proposals and accepts lowest seq number proposal
- Why might this not work?
Paxos

The Part-Time Parliament
Leslie Lamport

This article appeared in ACM Transactions on Computer Systems 16, 2 (May 1998), 133-169. Minor corrections were made on 29 August 2000.

- Original paper submitted in 1990
  - Tells mythical story of Greek island of Paxos with “legislators” and “current law” passed through parliamentary voting protocol

- Widely used in industry today
Desirable Properties

- **Safety**
  - “No bad things happen”
  - System *never* reaches an undesirable state

- **Liveness**
  - “Good things eventually happen”
  - System makes progress *eventually*

- **Tradeoff between consistency and latency**
Desired Properties of Solution

- **Safety:**
  - Accept a value only if accepted by a majority
  - Accept a value only if proposed by some client

- **Liveness:**
  - If any values are proposed, one of them will eventually be accepted
  - If a value is accepted, all replicas will eventually discover that it was chosen
Roles of a Process

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

- In reality, a process can play any/all roles
- Roles in bank account example?
- Roles in US Senate example?
Paxos Overview

- Three phases within each round
  - **Prepare Phase:**
    - Proposer sends a unique proposal number to all acceptors
    - Waits to get commitment from majority of acceptors
  - **Accept Phase:**
    - Proposer sends proposed value to all acceptors
    - Waits to get proposal accepted by majority
  - **Learn Phase:**
    - Learners discover value accepted by majority
Paxos State

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

- This state must persist across restarts
- Learners can re-discover accepted value (if any) from acceptors
Paxos Phase 1

- **Proposer:**
  - Choose unique proposal number $n$
  - Send $<\text{prepare, } n>$ to all acceptors

- **Acceptors:**
  - If $n \geq n_p$
    - $n_p = n$  \(\xrightarrow{\text{promise}}\) promise not to accept any new proposals $n' < n$
    - If no prior proposal accepted
      - Reply $<\text{promise, } n, \emptyset>$
    - Else
      - Reply $<\text{promise, } n, (n_a, v_a)>$
  - Else
    - Reply $<\text{prepare-failed}>$
Prepare Phase

\[ n_p = 2 \]

October 2, 2017
Paxos Phase 1

Proposer:
- How to pick unique proposal number?
- 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 >
**Paxos Phase 2**

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

  - *Why not stop if $v_\text{a} \neq v$?*

- **Acceptors:**
  - Upon receiving $(n, v)$, if $n \geq n_\text{p}$,
    - Accept proposal and notify learner(s)
      - $n_\text{a} = n_\text{p} = n$
      - $v_\text{a} = v$
Accept Phase

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

- \( n=1 \)
- \( v=v_1 \)
- \( n=2 \)
- \( v=v_1 \)
Accept Phase

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

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

\[ P_1 (v_1) \]

\[ n=1 \]

\[ v=v_1 \]

\[ n=1 \]

\[ P_2 (v_2) \]

\[ n=2 \]

\[ v=v_1 \]
Paxos: Sample Execution

- Acceptor 1: P1 A1-X P2
- Acceptor 2: P1 A1-X P2 P3
- Acceptor 3: P1 A1-X P3
Paxos: Sample Execution

- Acceptor 1: P1, A1-X, P3, A3-X
- Acceptor 3: P2, A2-Y
Paxos: Sample Execution

- **Acceptor 1:** P1, A1-X
- **Acceptor 2:** P1, A1-X, P2
- **Acceptor 3:** P2
Paxos: Sample Execution

- **Acceptor1:** P1 A1-X
- **Acceptor2:** P1 A1-X P2 A2-X
- **Acceptor3:** P2 A2-X
Paxos: Sample Execution

- **Acceptor 1:** P1

- **Acceptor 2:** P1 A1-X P2

- **Acceptor 3:** P2
Paxos: Sample Execution

- **Acceptor 1:** P1
- **Acceptor 2:** P1 A1-X P2 A2-X
- **Acceptor 3:** P2 A2-X
Paxos: Sample Execution

- Acceptor 1: P1  A1-X
- Acceptor 2: P1  P2
- Acceptor 3:  P2
Paxos: Sample Execution

- Acceptor 1: P1 A1-X
- Acceptor 2: P1 P2 A2-Y
- Acceptor 3: P2 A2-Y
**Paxos is safe**

- Intuition: Once 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:**
  - Accept a value only if accepted by a majority
  - Accept a value only if proposed by some client

- **Liveness:**
  - If any values are proposed, one of them will eventually be accepted
  - If a value is accepted, 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

... can go on indefinitely ...

Process 1

Starts and completes phase 1 with proposal n1 > n0

Performs phase 2, acceptors reject
Paxos: Race condition

- Acceptor1: P1 A1-X P3
- Acceptor2: P1 P2 A1-X P3 A2-Y P4
- Acceptor3: P2 A2-Y P4

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**
Why two phases?

- **Liveness problem is partly due to two phases**
  - Between one proposer’s Prepare and Accept phases, \( n_p \) updated by another proposer

- **Alternate design:**
  - Proposer sends propose messages to all acceptors
  - Retry with higher proposal no. if majority don’t accept

- **Problem?**
  - Once a value is accepted by majority, we don’t want another value accepted by a majority
Paxos: Three Phases

- **Prepare Phase:**
  - Proposer gets commitment from majority of acceptors

- **Accept Phase:**
  - Proposer sends proposed value to all acceptors
  - Waits to get proposal accepted by majority

- **Learn Phase:**
  - Learners discover value accepted by majority
Paxos in Action
Paxos Phase 3

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

- **Potential approaches:**
  - Proposer who has proposal accepted by majority of acceptors informs all learners
  - Acceptor broadcasts to all learners whenever it accepts any value
  - Acceptors notify distinguished learner, which informs others
Paxos Phase 3

- Learners mimic proposers
- Discover value accepted by each acceptor in response to prepare messages
Paxos: Sample Execution

- **Acceptor1:** P1 A1-X P2
- **Acceptor2:** P1 A1-X P2 P3
- **Acceptor3:** P1 A1-X P3
RSM with Paxos

- Log of updates at every replica
  - Replicas execute updates in order in log

- Use Paxos to come to consensus about each slot of the log
RSM with Paxos

Proposals / ballots

Slots in log

40
RSM with Paxos

- **Example:** updates from MapReduce workers submitted to replicated Master

- Whenever an update is submitted:
  - Attempt to get update accepted to a particular slot in replicated log
  - If unsuccessful, retry proposing to higher slot

- **Challenge:** *Must guess slot at end of log*
RSM with Paxos

- $e_1$: an operation is accepted to $i^{th}$ slot in log
- $e_2$: $i^{th}$ operation is executed at all replicas

- Arbitrarily large delay between events $e_1$ and $e_2$

- Consequence: Local state at any replica differs from state of replicated log
Comparing with P/B Replication
Comparing with P/B Replication

- **Benefit of Paxos:**
  - Need only majority of replicas to be up

- **Downside of Paxos?**
  - Need two rounds of inter-replica communication
Leader-based Paxos
Leader-based Paxos

- Pick one of the acceptors as the leader
- All clients submit proposals to leader
- Leader can directly skip to Accept phase because no contention
- Learn phase executed asynchronously

- How to pick a leader?
  - Paxos!

- Drawbacks compared to leaderless Paxos?
  - Leader may be far from client