Today

1. From primary-backup to viewstamped replication

2. Consensus
Review: Primary-Backup Replication

• Nominate one replica primary
  – Clients send all requests to primary
  – Primary orders clients’ requests
From Two to Many Replicas

- Primary-backup with many replicas
  - Primary waits for acknowledgement from all backups
  - All updates to set of replicas needs to update shared disk
What else can we do with more replicas?

• Viewstamped Replication:
  – State Machine Replication for any number of replicas
  – **Replica group**: Group of $2f + 1$ replicas
    • Protocol can tolerate $f$ replica crashes

• Differences with primary-backup
  – No shared disk (no reliable failure detection)
  – Don’t need to wait for all replicas to reply
  – Need more replicas to handle $f$ failures ($2f+1$ vs $f+1$)
Replica State

1. **configuration**: identities of all $2f + 1$ replicas

2. In-memory **log** with clients’ requests in assigned order

\[
\langle \text{op1, args1} \rangle \langle \text{op2, args2} \rangle \langle \text{op3, args3} \rangle \langle \text{op4, args4} \rangle
\]
Normal Operation

1. Primary adds request to end of its log
2. Replicas add requests to their logs in primary’s log order
3. Primary \textit{waits for} \(f\) PrepareOKs \(\rightarrow\) request is \textit{committed}
Normal Operation: Key Points

- Protocol provides state machine replication
- On execute, primary knows request in $f + 1 = 2$ nodes’ logs
  - Even if $f = 1$ then crash, $\geq 1$ retains request in log
Piggybacked Commits  

- Previous Request’s commit piggybacked on current Prepare
- No client Request after a timeout period?
  - Primary sends Commit message to all backups
The Need For a View Change

- So far: **Works** for $f$ failed backup replicas
- But what if the $f$ failures include a **failed primary**?
  - All clients’ requests go to the failed primary
  - System **halts** despite merely $f$ failures
Views

• Let **different replicas** assume role of primary over time
• System moves through a sequence of views
  – **View** = (view number, primary id, backup id, ...)

![Diagram showing sequence of views](image-url)
Correctly Changing Views

• View changes happen locally at each replica
• Old primary executes requests in the old view, new primary executes requests in the new view
• Want to ensure state machine replication
• So correctness condition: Executed requests
  1. Survive in the new view
  2. Retain the same order in the new view
How do replicas agree to move to a new view?

How do replicas agree on what was executed (and in what order) in the old view?
Consensus

- Definition:
  1. A general agreement about something
  2. An idea or opinion that is shared by all the people in a group
Consensus Used in Systems

Group of servers want to:

• Make sure all servers in group receive the same updates in the same order as each other
• Maintain own lists (views) on who is a current member of the group, and update lists when somebody leaves/fails
• Elect a leader in group, and inform everybody
• Ensure mutually exclusive (one process at a time only) access to a critical resource like a file
Consensus

Given a set of processors, each with an initial value:

• **Termination:** All non-faulty processes eventually decide on a value

• **Agreement:** All processes that decide do so on the same value

• **Validity:** Value decided must have proposed by some process
Safety vs. Liveness Properties

- Safety (bad things never happen)

- Liveness (good things eventually happen)
Paxos

- Safety (bad things never happen)
  - Agreement: All processes that decide do so on the same value
  - Validity: Value decided must have proposed by some process

- Liveness (good things eventually happen)
  - Termination: All non-faulty processes eventually decide on a value
Paxos’s Safety and Liveness

- Paxos is always safe

- Paxos is very often live (but not always, more later)

- Also true for Viewstamped Replication, RAFT, and other similar protocols
Roles of a Process in Paxos

• Three conceptual roles
  – Proposers propose values
  – Acceptors accept values, where value is chosen if majority accept
  – Learners learn the outcome (chosen value)

• In reality, a process can play any/all roles
Strawmen

- 3 proposers, 1 acceptor
  - Acceptor accepts first value received
  - No liveness with single failure

- 3 proposers, 3 acceptors
  - Accept first value received, learners choose common value known by majority
  - But no such majority is guaranteed
Paxos

• Each acceptor accepts **multiple proposals**
  – Hopefully one of multiple accepted proposals will have a majority vote (and we determine that)
  – If not, rinse and repeat (more on this)

• How do we select among multiple proposals?
  – Ordering: proposal is tuple \((\text{proposal } #, \text{ value}) = (n, v)\)
  – Proposal # strictly increasing, globally unique
  – Globally unique?
    • Trick: set low-order bits to proposer’s ID
Paxos Protocol Overview

- **Proposers:**
  1. Choose a proposal number $n$
  2. Ask acceptors if any accepted proposals with $n_a < n$
  3. If existing proposal $v_a$ returned, propose same value $(n, v_a)$
  4. Otherwise, propose own value $(n, v)$

Note *altruism*: goal is to reach consensus, not “win”

- **Accepters** try to accept value with highest proposal $n$
- **Learners** are passive and wait for the outcome
Paxos Phase 1

- **Proposer:**
  - Choose proposal n, send <prepare, n> to acceptors

- **Acceptors:**
  - If n > n_h
    - n_h = 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:**
  – If receive promise from *majority* of acceptors,
    • Determine $v_a$ returned with highest $n_a$, if exists
    • Send $<\text{accept, } (n, v_a \| v)>$ to acceptors

• **Acceptors:**
  – Upon receiving $(n, v)$, if $n \geq n_h$,
    • Accept proposal and notify learner(s)
      
      $n_a = n_h = n$
      
      $v_a = v$
Paxos Phase 3

• **Learners** need to know which value chosen

• **Approach #1**
  – Each acceptor notifies all learners
  – More expensive

• **Approach #2**
  – Elect a “distinguished learner”
  – Acceptors notify elected learner, which informs others
  – Failure-prone
Paxos: Well-behaved Run

- <prepare, 1>
- <promise, 1>
- <accept, (1, v1)>
- decide v1

1 1 1 1
2 2 2 2
n n n n

<accepted, (1 ,v1)>
Paxos is Safe

• Intuition: if proposal with value v chosen, then every higher-numbered proposal issued by any proposer has value v.

Majority of acceptors accept (n, v):

v is chosen

Next prepare request with proposal n+1
Often, but not always, live

**Process 0**
- Completes phase 1 with proposal n0
- Performs phase 2, acceptors reject
- Restarts 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 …
Paxos Summary

• Described for a single round of consensus
• Proposer, Acceptors, Learners
  – Often implemented with nodes playing all roles

• Always safe: Quorum intersection
• Very often live

• Acceptors accept multiple values
  – But only one value is ultimately chosen
• Once a value is accepted by a majority it is chosen
Flavors of Paxos

• Terminology is a mess
• Paxos loosely and confusingly defined...

• We’ll stick with
  – Basic Paxos
  – Multi-Paxos
Flavors of Paxos: Basic Paxos

- Run the full protocol each time
  - e.g., for each slot in the command log

- Takes 2 rounds until a value is chosen
Flavors of Paxos: Multi-Paxos

- Elect a leader and have them run 2\textsuperscript{nd} phase directly
  - e.g., for each slot in the command log
  - Leader election uses Basic Paxos

- Takes 1 round until a value is chosen
  - Faster than Basic Paxos

- Used extensively in practice!
  - RAFT is similar to Multi Paxos