Quorum Replication
(Paxos)

Feb 12, 2016
CPSC 416
Goal

• Provide a service
• Survive the failure of up to $f$ replicas
• Provide identical service as a non-replicated version (except more reliable, and perhaps different performance)

(A lot like your assignment 4 (where $f = r-1$) except without durable storage)
We’ll cover

• Primary-backup
  • Operations handled by primary, it streams copies to backup(s)
  • Replicas are “passive”
  • Good: Simple protocol. Bad: Clients must participate in recovery.

• quorum consensus
  • Designed to have fast response time even under failures
  • Replicas are “active” - participate in protocol; there is no master, per se.
  • Good: Clients don’t even see the failures. Bad: More complex.
Problems with p-b

- Not a great solution if you want very tight response time even when something has failed: Must wait for failure detector
- For that, quorum based schemes are used
- As name implies, different result:
  - To handle $f$ failures, must have $2f + 1$ replicas. Why? so that a majority is still alive
Paxos [Lamport]

- quorum consensus usually boils down to the Paxos algorithm.

- Very useful functionality in big systems/clusters.

- Some notes in advance:
  - Paxos is painful to get right, particularly the corner cases. Steal an implementation if you can. See Yahoo’s “Zookeeper” as a starting point.
  - There are lots of optimizations to make the common / no or few failures cases go faster; if you find yourself implementing, research these.
  - Paxos is expensive, as we’ll see. Usually, used for critical, smaller bits of data and to coordinate cheaper replication techniques such as primary-backup for big bulk data.
Paxos requirement

- **Correctness (safety):**
  - All nodes agree on the same value
  - The agreed value $X$ has been proposed by some node

- **Fault-tolerance:**
  - If less than $N/2$ nodes fail, the rest should reach agreement eventually \textit{w.h.p}
  - Liveness is not \textit{guaranteed}
Paxos: general approach

- Elect a replica to be the Leader
- Leader proposes a value and solicits acceptance from others
- If a majority ACK, the leader then broadcasts a commit message.

- This process may be repeated many times, as we’ll see.

Paxos slides adapted from Jinyang Li, NYU; some terminology from “Paxos Made Live” (Google)
Why is agreement hard?

• What if >1 nodes think they’re leaders simultaneously?
• What if there is a network partition?
• What if a leader crashes in the middle of solicitation?
• What if a leader crashes after deciding but before broadcasting commit?
• What if the new leader proposes different values than already committed value?
Basic two-phase commit

• Coordinator tells replicas: “Value V”
• Replicas ACK
• Coordinator broadcasts “Commit!”

• This isn’t enough
  – What if some of the nodes or the coordinator fails during the communication?
  – What if there’s more than 1 coordinator at the same time? (let’s solve this first)
Combined leader election and
two-phase

Propose\((N)\) -- dude, I'm the master

if \(N \geq Nh\), Promise\((N)\) -- ok, you're the boss. (I haven't seen anyone with a higher \(N\), the highest \(N\) that I observed was \(Nh\))

if majority promised: Accept\((V, N)\) -- please agree on the value \(V\)

if \(N \geq Nh\), ACK\((V, N)\) -- Ok!

if majority ACK: Commit\((V)\)
Multiple coordinators

• The value N is basically a lamport clock.
• Nodes that want to be the leader generate an N higher than any they’ve seen before
• If you get NACK’d on the propose, back off for a while - someone else is trying to be leader
• Have to check N at later steps, too, e.g.:
  • Leader1:  N = 5 --> propose --> promise
  • Leader2:  N = 6 --> propose --> promise
  • Leader1:  N = 5 --> accept(V1, ...)
  • Replicas:  NACK! Someone beat you to it.
  • Leader2:  N = 6 --> accept(V2, ...)
  • Replicas:  Ok!
But...

- What happens if there’s a failure? Let’s say the coordinator crashes before sending the commit message.
- Or if only one or two of the replicas received the commit message.
Paxos solution

• Proposals are ordered by proposal #
• Each acceptor may accept multiple proposals
  – If a proposal with value v is chosen, all higher proposals must have value v
• 3-round protocol (complex!)
Paxos operation: node state

• Each node maintains:
  – $n_a$, $v_a$: highest proposal # and its corresponding accepted value for round $a$
  – $n_h$: highest proposal # seen (for round $a$)
  – $m_y$: my proposal # in Paxos round $a$ (leader’s state when proposing in this round)
Paxos operation: 3-phase protocol

• Phase 1 (Prepare)
  – A node decides to be leader (and proposes)
  – Leader choose $m_n > n_h$
  – Leader sends $<\text{prepare, my}_n>$ to all nodes
  – Upon receiving $<\text{prepare, n}>$
    
    If $n < n_h$
    reply $<\text{prepare-reject}>
    
    Else
    $n_h = n$
    reply $<\text{prepare-ok, n}_a,v_a>$

See the relation to lamport clocks?
Paxos operation

• Phase 2 (Accept):
  – If leader gets prepare-ok from a majority
    V = non-empty value corresponding to the highest \( n_a \) received
    If \( V = \) null, then leader can pick any \( V \)
    Send \(<\text{accept}, \text{my}_n, V>\) to all nodes
  – If leader fails to get majority prepare-ok
    • Delay and restart Paxos
  – Upon receiving \(<\text{accept}, n, V>\>
    If \( n < n_h \)
      reply with \(<\text{accept-reject}>\>
    else
      \( n_a = n; v_a = V; n_h = n \)
      reply with \(<\text{accept-ok}>\>
Paxos operation

• Phase 3 (Commit)
  – If leader gets accept-ok from a majority
    • Send <commit, va> to all nodes
  – If leader fails to get accept-ok from a majority
    • Delay and restart Paxos
Paxos Examples

• Failure after getting 1 node to accept the value
  – One example where the master hears the value from one of the nodes
  – One example where a new value wins
• Failure after getting > 1/2 nodes to accept the value
• Simultaneous failure of master and the 1 node that accepted in a 5 node system
## Paxos operation: an example

<table>
<thead>
<tr>
<th>nh</th>
<th>na</th>
<th>va</th>
</tr>
</thead>
<tbody>
<tr>
<td>N0:0</td>
<td></td>
<td>null</td>
</tr>
<tr>
<td>N1:1</td>
<td>null</td>
<td>null</td>
</tr>
<tr>
<td>N2:0</td>
<td></td>
<td>null</td>
</tr>
</tbody>
</table>

**N0**

1. Prepare, N1:1
2. ok, na = va = null
3. Accept, N1:1, val1
4. ok
5. Commit, val1

**N1**

1. Prepare, N1:1
2. ok, na = va = null
3. Accept, N1:1, val1
4. ok
5. Commit, val1

**N2**

1. Prepare, N1:1
2. ok, na = va = null
3. Accept, N1:1, val1
4. ok
5. Commit, val1
Replication Wrap-Up

- Primary/Backup quite common, works well, introduces some time lag to recovery when you switch over to a backup. Doesn’t handle as large a set of failures. \( f+1 \) nodes can handle \( f \) failures.

- Paxos is a general, quorum-based mechanism that can handle lots of failures, and quick response time. \( 2f+1 \) nodes to handle \( f \) failures.