Implementing Replicated Logs with Paxos

John Ousterhout and Diego Ongaro
Stanford University

Note: this material borrows heavily from slides by Lorenzo Alvisi, Ali Ghodsi, and David Mazières
Goal: Replicated Log

- **Replicated log** => **replicated state machine**
  - All servers execute same commands in same order

- Consensus module ensures proper log replication

- System makes progress as long as any majority of servers are up

- Failure model: fail-stop (not Byzantine), delayed/lost messages
Decompose the problem:

- **Basic Paxos** ("single decree"):  
  - One or more servers propose values  
  - System must agree on a single value as chosen  
  - Only one value is ever chosen

- **Multi-Paxos**:  
  - Combine several instances of Basic Paxos to agree on a series of values forming the log
Safety:
- Only a single value may be chosen
- A server never learns that a value has been chosen unless it really has been

Liveness (as long as majority of servers up and communicating with reasonable timeliness):
- Some proposed value is eventually chosen
- If a value is chosen, servers eventually learn about it

The term “consensus problem” typically refers to this single-value formulation
Paxos Components

- **Proposers:**
  - Active: put forth particular values to be chosen
  - Handle client requests

- **Acceptors:**
  - Passive: respond to messages from proposers
  - Responses represent votes that form consensus
  - Store chosen value, state of the decision process
  - Want to know which value was chosen

**For this presentation:**
- Each Paxos server contains both components
Strawman: Single Acceptor

- **Simple (incorrect) approach:** a single acceptor chooses value
- **What if acceptor crashes after choosing?**
- **Solution: quorum**
  - Multiple acceptors (3, 5, ...)
  - Value $v$ is *chosen* if accepted by majority of acceptors
  - If one acceptor crashes, chosen value still available
Problem: Split Votes

- Acceptor accepts only first value it receives?
- If simultaneous proposals, no value might be chosen

Acceptors must sometimes accept multiple (different) values
Problem: Conflicting Choices

- Acceptor accepts every value it receives?
- Could choose multiple values

Once a value has been chosen, future proposals must propose/choose that same value (2-phase protocol)
- $s_5$ needn’t propose red (it hasn’t been chosen yet)
- $s_1$’s proposal must be aborted ($s_3$ must reject it)

Must order proposals, reject old ones
Proposal Numbers

- Each proposal has a unique number
  - Higher numbers take priority over lower numbers
  - It must be possible for a proposer to choose a new proposal number higher than anything it has seen/used before

- One simple approach:
  - Each server stores maxRound: the largest Round Number it has seen so far
  - To generate a new proposal number:
    - Increment maxRound
    - Concatenate with Server Id
  - Proposers must persist maxRound on disk: must not reuse proposal numbers after crash/restart
Basic Paxos

Two-phase approach:

- **Phase 1: broadcast Prepare RPCs**
  - Find out about any chosen values
  - Block older proposals that have not yet completed

- **Phase 2: broadcast Accept RPCs**
  - Ask acceptors to accept a specific value
Basic Paxos

### Proposers

1) Choose new proposal number n

2) Broadcast $\text{Prepare}(n)$ to all servers

4) When responses received from majority:
   - If any acceptedValues returned, replace value with acceptedValue for highest acceptedProposal

5) Broadcast $\text{Accept}(n, \text{value})$ to all servers

6) When responses received from majority:
   - Any rejections (result $> n$)? goto (1)
   - Otherwise, **value is chosen**

### Acceptors

3) Respond to $\text{Prepare}(n)$:
   - If $n > \text{minProposal}$ then $\text{minProposal} = n$
   - Return($\text{acceptedProposal}$, $\text{acceptedValue}$)

6) Respond to $\text{Accept}(n, \text{value})$:
   - If $n \geq \text{minProposal}$ then $\text{acceptedProposal} = \text{minProposal} = n$
   - $\text{acceptedValue} = \text{value}$
   - Return($\text{minProposal}$)

Acceptors must record minProposal, acceptedProposal, and acceptedValue on stable storage (disk)
Basic Paxos Examples

Three possibilities when later proposal prepares:

1. **Previous value already chosen:**
   - New proposer will find it and use it

---

```
“Prepare proposal 3.1 (from s₁)”
```

```
“Accept proposal 4.5 with value X (from s₅)”
```

---

```
P 3.1  | A 3.1 X
P 3.1  | A 3.1 X
P 3.1  | A 3.1 X
P 4.5  | A 4.5 X
P 4.5  | A 4.5 X
```

---

March 1, 2013

Implementing Replicated Logs with Paxos

Slide 13
Three possibilities when later proposal prepares:

2. Previous value not chosen, but new proposer sees it:
   - New proposer will use existing value
   - Both proposers can succeed

![Diagram showing the three possibilities with time values X and Y, and proposers S1 to S5]
Three possibilities when later proposal prepares:

3. Previous value not chosen, new proposer doesn’t see it:
   - New proposer chooses its own value
   - Older proposal blocked
Competing proposers can livelock:

One solution: randomized delay before restarting
- Give other proposers a chance to finish choosing

Multi-Paxos will use leader election instead
Only proposer knows which value has been chosen

If other servers want to know, must execute Paxos with their own proposal
Multi-Paxos

- Separate instance of Basic Paxos for each entry in the log:
  - Add index argument to Prepare and Accept (selects entry in log)

1. Client sends command to server
2. Server uses Paxos to choose command as value for a log entry
3. Server waits for previous log entries to be applied, then applies new command to state machine
4. Server returns result from state machine to client
Multi-Paxos Issues

- Which log entry to use for a given client request?
- Performance optimizations:
  - Use leader to reduce proposer conflicts
  - Eliminate most Prepare requests
- Ensuring full replication
- Client protocol
- Configuration changes

Note: Multi-Paxos not specified precisely in literature
When request arrives from client:

- Find first log entry not known to be chosen
- Run Basic Paxos to propose client’s command for this index
- Prepare returns acceptedValue?
  - Yes: finish choosing acceptedValue, start again
  - No: choose client’s command
Selecting Log Entries, cont’d

- Servers can handle multiple client requests concurrently:
  - Select different log entries for each

- Must apply commands to state machine in log order
Improving Efficiency

- **Using Basic Paxos is inefficient:**
  - With multiple concurrent proposers, *conflicts* and restarts are likely (higher load $\rightarrow$ more conflicts)
  - 2 rounds of RPCs for each value chosen (Prepare, Accept)

**Solution:**

1. **Pick a leader**
   - At any given time, only one server acts as Proposer

2. **Eliminate most Prepare RPCs**
   - Prepare once for the entire log (not once per entry)
   - Most log entries can be chosen in a single round of RPCs
Leader Election

One simple approach from Lamport:

- Let the server with highest ID act as leader
- Each server sends a heartbeat message to every other server every $T$ ms
- If a server hasn’t received heartbeat from server with higher ID in last $2T$ ms, it acts as leader:
  - Accepts requests from clients
  - Acts as proposer and acceptor
- If server not leader:
  - Rejects client requests (redirect to leader)
  - Acts only as acceptor
Eliminating Prepares

Why is Prepare needed?

- Block old proposals
  - Make proposal numbers refer to the entire log, not just one entry
- Find out about (possibly) chosen values
  - Return highest proposal accepted for current entry
  - Also return noMoreAccepted: no proposals accepted for any log entry beyond current one

If acceptor responds to Prepare with noMoreAccepted, skip future Prepares with that acceptor (until Accept rejected)

Once leader receives noMoreAccepted from majority of acceptors, no need for Prepare RPCs
  - Only 1 round of RPCs needed per log entry (Accepts)
So far, information flow is incomplete:
- Log entries not fully replicated (majority only)
  Goal: full replication
- Only proposer knows when entry is chosen
  Goal: all servers know about chosen entries

Solution part 1/4: keep retrying Accept RPCs until all acceptors respond (in background)
- Fully replicates most entries

Solution part 2/4: track chosen entries
- Mark entries that are known to be chosen:
  acceptedProposal[i] = ∞
- Each server maintains firstUnchosenIndex: index of earliest log entry not marked as chosen
Solution part 3/4: proposer tells acceptors about chosen entries

- Proposer includes its firstUnchosenIndex in Accept RPCs.
- Acceptor marks all entries i chosen if:
  - $i < request.firstUnchosenIndex$
  - $\text{acceptedProposal}[i] == request.proposal$

Result: acceptors know about most chosen entries

<table>
<thead>
<tr>
<th>log index</th>
<th>1</th>
<th>2</th>
<th>3</th>
<th>4</th>
<th>5</th>
<th>6</th>
<th>7</th>
<th>8</th>
<th>9</th>
</tr>
</thead>
<tbody>
<tr>
<td>acceptedProposal</td>
<td>$\infty$</td>
<td>$\infty$</td>
<td>$\infty$</td>
<td>2.5</td>
<td>$\infty$</td>
<td>3.4</td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

... Accept(proposal = 3.4, index=8, value = v, firstUnchosenIndex = 7) ...

<table>
<thead>
<tr>
<th>log index</th>
<th>1</th>
<th>2</th>
<th>3</th>
<th>4</th>
<th>5</th>
<th>6</th>
<th>7</th>
<th>8</th>
<th>9</th>
</tr>
</thead>
<tbody>
<tr>
<td>acceptedProposal</td>
<td>$\infty$</td>
<td>$\infty$</td>
<td>$\infty$</td>
<td>2.5</td>
<td>$\infty$</td>
<td>$\infty$</td>
<td></td>
<td>3.4</td>
<td></td>
</tr>
</tbody>
</table>

Still don’t have complete information
Solution part 4/4: entries from old leaders

- Acceptor returns its firstUnchosenIndex in Accept replies
- If proposer’s firstUnchosenIndex > firstUnchosenIndex from response, then proposer sends Success RPC (in background)

Success(index, v): notifies acceptor of chosen entry:

- acceptedValue[index] = v
- acceptedProposal[index] = ∞
- return firstUnchosenIndex
- Proposer sends additional Success RPCs, if needed
送命令至领导者
- 如果领导者未知，联系任何服务器
- 如果联系的服务器不是领导者，它将重定向到领导者

领导者在命令被日志条目选择并由领导者的状态机执行之前不会响应

如果请求超时（例如，领导者崩溃）：
- 客户端重新向其他服务器发送命令
- 最终重定向到新领导者
- 用新领导者重试请求
What if leader crashes after executing command but before responding?
- Must not execute command twice

Solution: client embeds a unique id in each command
- Server includes id in log entry
- State machine records most recent command executed for each client
- Before executing command, state machine checks to see if command already executed, if so:
  - Ignore new command
  - Return response from old command

Result: exactly-once semantics as long as client doesn’t crash
Configuration Changes

- **System configuration:**
  - ID, address for each server
  - Determines what constitutes a majority

- **Consensus mechanism must support changes in the configuration:**
  - Replace failed machine
  - Change degree of replication
Safety requirement:
- During configuration changes, it must not be possible for different majorities to choose different values for the same log entry:
Paxos solution: use the log to manage configuration changes:

- Configuration is stored as a log entry
- Replicated just like any other log entry
- Configuration for choosing entry i determined by entry i-\(\alpha\).

Suppose \(\alpha = 3\):

<table>
<thead>
<tr>
<th></th>
<th>1</th>
<th>2</th>
<th>3</th>
<th>4</th>
<th>5</th>
<th>6</th>
<th>7</th>
<th>8</th>
<th>9</th>
<th>10</th>
</tr>
</thead>
<tbody>
<tr>
<td>C1</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td>C2</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

- Notes:
  - \(\alpha\) limits concurrency: can’t choose entry i+\(\alpha\) until entry i chosen
  - Issue no-op commands if needed to complete change quickly
Paxos Summary

- **Basic Paxos:**
  - Prepare phase
  - Accept phase

- **Multi-Paxos:**
  - Choosing log entries
  - Leader election
  - Eliminating most Prepare requests
  - Full information propagation

- **Client protocol**

- **Configuration changes**