DISTRIBUTED CONSENSUS-
PART 5: PAXOS AND PREPARE PHASE

Instructor: Prasun Dewan (FB 150, dewan@unc.edu)
Auto Select Server (Last Slide of Part 4)

Non centralized algorithm with replica proposing itself necessary if sorted list not sufficient

Different consensus states possible even if ids used for switchover

We could use the voice of more than one member
MONARCHY VS. PARLIAMENT SYSTEM

Issue: how to detect and sequence concurrent requests?
In fact, it is among the simplest and most obvious of distributed algorithms.

**PAXOS AND ASSOCIATED ALGORITHM**

**Paxos Made Simple**

Leslie Lamport

01 Nov 2001

Fictional consensus system

**LESLIE LAMPORT**

United States – 2013

CITATION

For fundamental contributions to the theory and practice of distributed and concurrent systems, notably the invention of concepts such as causality and logical clocks, safety and liveness, replicated state machines, and sequential consistency.
Feedback Symmetry

Issue: how to detect and sequence concurrent requests?

A single replica decides

Multiple processes (acceptors) decide

They already decide site-specific aspects of feedback in synchronous case

Asynchronous version does not make sense as multiple rounds required
Synchronous

Acceptors

Proposers

Learners
(Almost) Basic Paxos
Assumptions and Guarantees

- Messages can be delayed and out of order and get lost and be retransmitted
  - At application level
- The replicated value is written once
  - A single command to a replicated state machine
- All learner commits have the same proposed value.
- A learner site commit to this value
  - When and if it receives its acceptance from a majority of known acceptors
- Nodes can come back up and messages can be retransmitted
  - Eventually all listeners will commit to this value
MAJORITY RULES

Accepter (accept)

Each acceptor accepts only the first proposal it gets

Learner (accepted)

A learner picks the value that has majority acceptances

Different acceptors make get proposals in different order

Acceptances may arrive in learners in different order so different learners may use different majorities

But each pair of majorities will have at least one member in common so will select same proposal
Two proposal may get the same number of votes

Choose odd number of acceptors?

Given N acceptors, it is possible to have N concurrent proposals, each of which wins in one acceptor
Some proposals are more equal than others

Accepter (accept)

Each acceptor accepts the proposal with the highest proposal number it will see

Learner (accepted)

A learner picks the value that has majority acceptances

Cannot predict future
A learner may learn multiple values associated with higher proposal numbers.

Each acceptor accepts a proposal if its proposal number is higher than what it has seen so far.

A learner picks the value that has majority acceptances.

An acceptor can change its mind.
INDIVIDUAL VOTE CAN CHANGE BUT NOT THE AGGREGATE VOTE

Learner (accepted)

A learner picks the first value that has majority acceptances

Different learners can get different majorities

Common members may have changed their mind

So different learners can pick different values
CONTROL SOURCE RATHER THAN SINK

Proposers

Acceptors

Learners
Propose Does not Play Spoiler

Proposer (propose)

A proposer does not override consensus (to be) achieved from earlier proposal

Propose needs to know distributed state before proposing

(Last known proposal, last accepted value)
ONE-PROPOSE PHASE, TWO OVERALL

Proposers

Acceptors

Learners
TWO-PROPOSE PHASES THREE-OVERALL

Acceptors

Proposers

Learners
Two Phase Propose

Proposer
(pre-propose (p, v^p))

A propose invocation sends a prepare message to query the current state of each acceptor

Proposer
(prepared(p, v^p))

Proposer does not propose new value if it could override a learnt value from previous proposal

Proposer (prepare)

Acceptor (prepare)

Each acceptor sends back last accepted value and its proposal number (or optionally last seen proposal number if no acceptance so far) and updates last seen proposal number
**Prepared Cases**

Proposer

\[(\text{prepared}(p, v^p))\]

Proposer waits for responses from at least majority of acceptors

\[(a^1, \ldots a^m)\]

Some acceptor has seen a higher proposal

No responding acceptor has accepted a previous proposal

Majority responding acceptors have accepted some previous proposal

At least one acceptor has accepted some proposal but no majority value in the acceptances majority
Proposed: Higher Proposal Exists

Proposer (prepared(p, v_p))

Proposer abandons proposal if it learns about a higher number proposal

Progress rather than safety condition
Proposed Cases

Proposer
(prepared(p, v^p))

Proposer waits for responses from at least majority of acceptors (a^1, ... a^m):

- Some acceptor has seen a higher proposal
- No responding acceptor has accepted a previous proposal ✔
- Majority responding acceptors have accepted some previous proposal
- At least one acceptor has accepted some proposal but no majority value
Prepared: No Known Previous Proposal

Proposer
\((\text{prepared}(p, v^p))\)

Proposer sends its proposal and value if it does not know about a previous accepted value.
**Prepared Cases**

Proposer
\[(\text{prepared}(p, v^p))\]

**Proposer waits for responses from at least majority of acceptors**
(a¹, ... aᵐ):

- Some acceptor has seen a higher proposal
- No responding acceptor has accepted a previous proposal
- Majority responding acceptors have accepted some previous proposal
- At least one acceptor has accepted some proposal but no majority value
**Prepared: No Known Majority Acceptance**

Proposer sends its proposal and value if majority acceptors have not yet accepted any value.

1 and 2 can both find this condition and propose their values $v^{p1}$ and $v^{p2}$, $p2 > p1$.

A majority of acceptors could get and accept $v^{p1}$ first then $v^{p2}$.
**Non Atomic Get and Set Prepare**

A propose message sends a prepare message to query the current state

Proposer sends its proposal and value if majority acceptors have not yet accepted any value

Problem arise because get of acceptor state and set of new proposal number and propose value in acceptor not done atomically

Cannot do all three atomically as result of get affects proposed action (need two proposer phases)

Can do get of acceptor state and set new proposal number atomically
ATOMIC GET AND SET PREPARE

Proposer
(pre-propose (p, v^p))

A propose message sends a prepare message with proposal number to query the current state and set its proposal number

Accepter (prepare)

Each acceptor sends back last accepted value and its proposal number (or optionally last seen proposal number if no acceptance so far) and updates last seen proposal number
**Propose Fixed**

Proposer sends its proposal and value if majority acceptors have not yet accepted any value.

1 and 2 can both find this condition and propose their values $v_{p1}$ and $v_{p2}$, $p_2 > p_1$.

A majority of acceptors could get and accept $v_{p1}$ first then $v_{p2}$.
**Prepare Cases**

- Proposer
  
  \[(\text{prepared}(p, v^p))\]

- A propose message sends a prepare message with proposal number \(p\) to query the current state and set its proposal number.

- Some acceptor has seen a higher proposal.

- No responding acceptor has accepted a previous proposal.

- Majority responding acceptors have accepted some previous proposal.

- At least one acceptor has accepted some proposal but no majority value in the acceptances majority.
**Prepared: Known Majority Value**

**Proposer**

\(\text{prepared}(p, v^p)\)

- Proposer proposes no value if it knows majority has already accepted a previous value.

A prepare phase can prevent some nodes from accepting the previous majority value.

If some of the nodes in current majority die, some learner nodes may not get consensus value even though majority of nodes are alive and can converge to a value.

Want consensus value to propagate to all acceptors for fault tolerance.
**Prepared: Known Majority Value Fixed**

Proposer (prepared(p, v_p))

Proposer (re) proposes majority value as its own value

1’s prepare phase can prevent some nodes from accepting the previous majority value

If some of the nodes in current majority die, some learner nodes may not get consensus value even though majority of nodes are alive

Want consensus value to propagate to all acceptors for fault tolerance
**Prepared Cases**

A propose message sends a prepare message with proposal number to query the current state and set its proposal number.

- Some acceptor has seen a higher proposal
- No responding acceptor has accepted a previous proposal
- Majority responding acceptors have accepted some previous proposal
- At least one acceptor has accepted some proposal but no majority value in the acceptances majority
Proposer (prepared(v^p))

Proposer (re) proposes value of highest accept proposal number as its own value

The value with highest proposal number may or may not be or become majority value (depending on whether there is some other unknown proposal in the system)

If it does become majority value then prepare phase may have locked some nodes from accepting it
**PREPARED CASES**

- **Proposer** \((\text{prepared}(v^p)))\)

**Proposer abandons proposal if it learns about a higher number proposal**

**Proposer sends its proposal and value if majority acceptors have not yet accepted any value**

**Proposer (re) proposes with highest accept proposal number as its own value (which may also be a majority value)**
THREE-PHASE SUMMARY

Acceptors

Proposers

Learners
**Phase 1 Summary**

**Proposer**
(pre-propose \((p, v^p)\))

A propose message sends a prepare message with proposal number to query the current state and set its proposal number.

**Accepter (prepare)**

Each acceptor sends back last accepted value and its proposal number (or optionally last seen proposal number if no acceptance so far) and updates last seen proposal number.
Phase 2 Summary

Proposer:
(prepared(v^p))

Proposer abandons proposal if it learns about a higher number proposal.

Proposer sends its proposal and value if majority acceptors have not yet accepted any value.

Proposer (re) proposes with highest accept proposal number as its own value (which may also be a majority value in majority acceptors).

Accepter (accept):

Each acceptor accepts a proposal if its proposal number is higher than what it has seen so far.
Phase 3

Learner (accepted)

A learner picks the first proposal that has majority acceptances
**Prepare-Prepare--Accept-Accept-Override**

![Diagram of the Prepare-Prepare--Accept-Accept-Override protocol with nodes labeled 1, 2, and 3. The diagram shows the sequence of propose, prepare, accept, and learn actions with values 42, 29, and 1 associated.]


**Prepared-Accept-Prepared-Accept- Re-propose**

Value Proposal

| 42 | 1 |
| 2 | ? |
| ? | 2 |

 propose(2, 29)  
 prepare(2)  
 propose(1, 42)  
 prepare(1)  
 accept(1, 42)  
 accept(2, 42)  
 accept(2, 42)  

Changed slide after lecture to delay 2’s prepared call on 3 so it calls accept after majority such calls rather than after all three prepared calls.
**Prepare-Accept-Prepare-Accept-Fault**

Value Proposal

<p>| | |</p>
<table>
<thead>
<tr>
<th></th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td>42</td>
<td>1</td>
</tr>
<tr>
<td>42</td>
<td>2</td>
</tr>
<tr>
<td>?</td>
<td>2</td>
</tr>
</tbody>
</table>

```plaintext
propose(1, 42)
prepare(1)
(1, ?, ?)
accept(1, 42)
1, 42

propose(2, 29)
prepare(2)
(2, ?, ?)

accept(2, 42)
(1, 42)
(2, 1, 42)
```
**Prepare-No-Accept**

Value Proposal

- ? 1
- ? 2
- ? 2

1. propose(1, 42)
2. prepare(1)
3. (1, ?, ?)

2. propose(2, 29)
3. prepare(2)
2. (2, 29)
2. (2, ?, ?)

3. accept(2, 29)
Proposal numbers are increasing

Value Proposal

1
- propose(1,42)
- prepare(1)
- accept(1,42)
- prepare(3)

2
- propose(2,29)
- prepare(2)
- accept(2,29)
- prepare(4)

3
- propose(2,29)
- prepare(2)
- accept(2,29)
- prepare(4)

LIVELOCK
## DISJOINT PROPOSAL IDS

<table>
<thead>
<tr>
<th>Each replica could be initialized with different sets of ids at initialization time</th>
</tr>
</thead>
<tbody>
<tr>
<td>Id of proposal made by a replica might be smaller than the ids of proposals it has already seen</td>
</tr>
<tr>
<td>Re-proposals may not be accepted</td>
</tr>
</tbody>
</table>
Increment #Proposals

Each replica keeps track of # known proposals.

Id of new proposal is one more than # known proposals.

Two proposals made by different replicas can have same id.
Each replica keeps track of # known proposals.

Id of new proposal is \( \text{siteId}.\#\text{Proposals} + 1 \)

All proposals made by a site will be smaller or larger than all proposals of another site.
Each replica keeps track of # known proposals.

Id of new proposal is  Proposals + 1.siteid

Id of proposal made by a replica will be unique and larger than the ids of proposals it has already seen

protected float getAndSetNextProposalNumber(StateType aProposal) {
    lastProposalNumber = wholePart(lastProposalNumber) + 1 + myPrefix;
}
Proposal numbers are increasing

How to avoid livelock?

Value Proposal

1

propose(1, 42)
preserve(1)
accept(1, 42)
preserve(3)

2

propose(2, 29)
preserve(2)
accept(2, 29)
preserve(4)

3

48
Central Proposer to Avoid Livelock

Inconsistent state?

Acceptors talk directly to learners

Majority value will be same

Can use another Paxos mechanism to achieve consensus about new central proposer

Or use ids to select proposer
CENTRAL LISTENER

Acceptors

Proposers

Learners
LAMPORT’S REQUIREMENTS

“P2. If a proposal with value v is chosen, then every higher-numbered proposal that is chosen has value v. “

“P2a. If a proposal with value v is chosen, then every higher-numbered proposal accepted by any acceptor has value v. “

“P2c. For any v and n, if a proposal with value v and number n is issued, then there is a set S consisting of a majority of acceptors such that either (a) no acceptor in S has accepted any proposal numbered less than n, or (b) v is the value of the highest-numbered proposal among all proposals numbered less than n accepted by the acceptors in S. (“Any set S consisting of a majority of acceptors contains at least one member of C , we can conclude that a proposal numbered n has value v by ensuring that the following invariant is maintained.”)
LAMPORT’S REQUIREMENTS

“P1a. An acceptor can accept a proposal numbered $n$ iff it has not responded to a prepare request having a number greater than $n$”
Lamport’s Algorithm (Verbatim)

“Phase 1.

(a) A proposer selects a proposal number n and sends a prepare request with number n to a majority of acceptors.

(b) If an acceptor receives a prepare request with number n greater than that of any prepare request to which it has already responded, then it responds to the request with a promise not to accept any more proposals numbered less than n and with the highest-numbered proposal (if any) that it has accepted.

Phase 2.

(a) If the proposer receives a response to its prepare requests (numbered n) from a majority of acceptors, then it sends an accept request to each of those acceptors for a proposal numbered n with a value v, where v is the value of the highest-numbered proposal among the responses, or is any value if the responses reported no proposals.

(b) If an acceptor receives an accept request for a proposal numbered n, it accepts the proposal unless it has already responded to a prepare request having a number greater than n”