Distributed Systems Consensus

Amir H. Payberah
amir@sics.se

Amirkabir University of Technology
(Tehran Polytechnic)

What is the Problem?

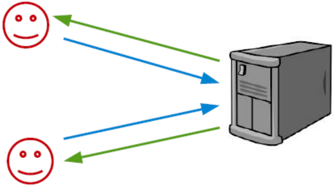
Two Generals' Problem

- Two generals need to be agree on time to attack to win.
- They communicate through messengers, who may be killed on their way.

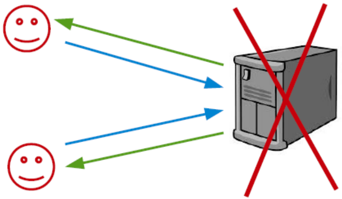
Two Generals' Problem

- Two generals need to be agree on time to attack to win.
- They communicate through messengers, who may be killed on their way.
- Agreement is the problem.

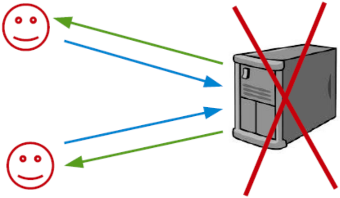
Replicated State Machine Problem (1/2)



Replicated State Machine Problem (1/2)

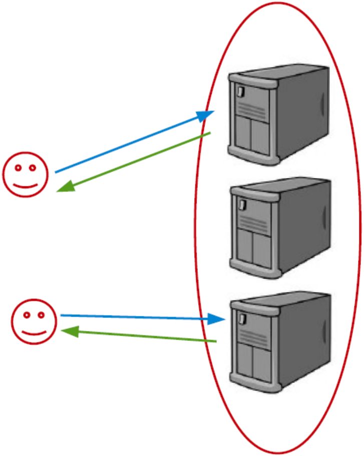


Replicated State Machine Problem (1/2)



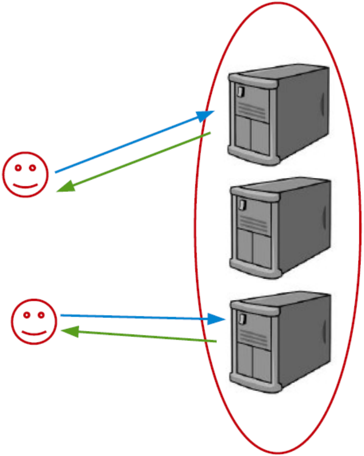
- The solution: replicate the server.

Replicated State Machine Problem (2/2)



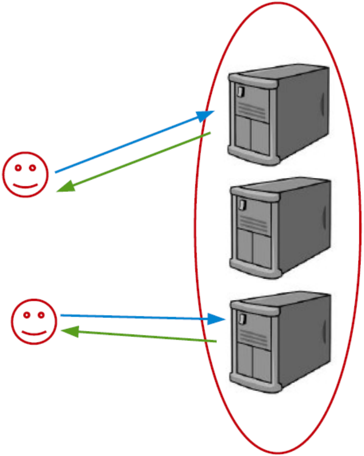
Replicated State Machine Problem (2/2)

- Make the server deterministic (state machine).



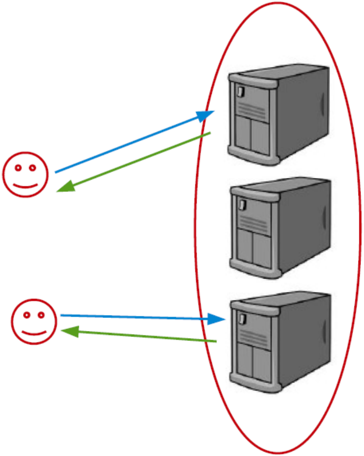
Replicated State Machine Problem (2/2)

- Make the server deterministic (state machine).
- Replicate the server.



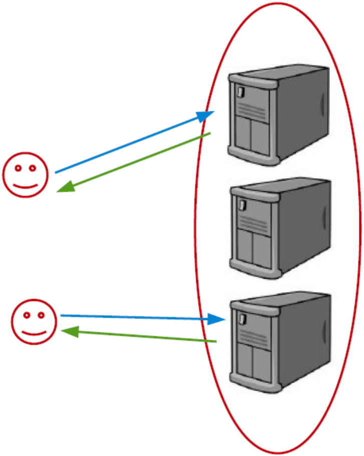
Replicated State Machine Problem (2/2)

- Make the server deterministic (state machine).
- Replicate the server.
- Ensure correct replicas step through the same sequence of state transitions (How?)



Replicated State Machine Problem (2/2)

- Make the server deterministic (state machine).
- Replicate the server.
- Ensure correct replicas step through the same sequence of state transitions (How?)
- Agreement is the problem.



The Agreement Problem

The Agreement Problem

- Some nodes propose values (or actions) by sending them to the others.
- All nodes must decide whether to accept or reject those values.

The Agreement Problem

- Some nodes propose values (or actions) by sending them to the others.
- All nodes must decide whether to accept or reject those values.

- But, ...

The Agreement Problem

- Some nodes propose values (or actions) by sending them to the others.
- All nodes must decide whether to accept or reject those values.

- But, ...

- Concurrent processes and uncertainty of timing, order of events and inputs.
- Failure and recovery of machines/processors, of communication channels.

Agreement Requirements

- Safety

- Liveness

Agreement Requirements

- Safety
  - Validity: only a value that has been proposed may be chosen.

- Liveness

Agreement Requirements

- Safety
  - Validity: only a value that has been proposed may be chosen.
  - Agreement: no two correct nodes choose different values.

- Liveness

Agreement Requirements

- Safety
  - Validity: only a value that has been proposed may be chosen.
  - Agreement: no two correct nodes choose different values.
  - Integrity: a node chooses at most once.
- Liveness

Agreement Requirements

- Safety
  - Validity: only a value that has been proposed may be chosen.
  - Agreement: no two correct nodes choose different values.
  - Integrity: a node chooses at most once.

- Liveness
  - Termination: every correct node eventually choose a value.

Agreement in Distributed Systems: Possible Solutions

- Two-Phase Commit (2PC)
- Paxos

Two-Phase Commit (2PC)

The Two-Phase Commit (2PC) Problem

- The problem first was encountered in database systems.

- Suppose a database system is updating some complicated data structures that include parts residing on more than one machine.

The Two-Phase Commit (2PC) Problem

- The problem first was encountered in database systems.
- Suppose a database system is updating some complicated data structures that include parts residing on more than one machine.
- System model:
  - Concurrent processes and uncertainty of timing, order of events and inputs (asynchronous systems).
  - Failure and recovery of machines/processors, of communication channels.

Intuitive Example (1/3)

- You want to organize outing with 3 friends at 6pm Tuesday.
  - Go out only if all friends can make it.

Intuitive Example (2/3)

- What do you do?

Intuitive Example (2/3)

- What do you do?
  - Call each of them and ask if can do 6pm on Tuesday (voting phase)

Intuitive Example (2/3)

- What do you do?
  - Call each of them and ask if can do 6pm on Tuesday (voting phase)
  - If all can do Tuesday, call each friend back to ACK (commit)

Intuitive Example (2/3)

- What do you do?
  - Call each of them and ask if can do 6pm on Tuesday (voting phase)
  - If all can do Tuesday, call each friend back to ACK (commit)
  - If one cannot do Tuesday, call other three to cancel (abort)

Intuitive Example (3/3)

- Critical details

Intuitive Example (3/3)

- Critical details
  - While you were calling everyone to ask, people who have promised they can do 6pm Tuesday must reserve that slot.

Intuitive Example (3/3)

- Critical details
  - While you were calling everyone to ask, people who have promised they can do 6pm Tuesday must reserve that slot.
  - You need to remember the decision and tell anyone whom you have not been able to reach during commit/abort phase.

Intuitive Example (3/3)

- Critical details
  - While you were calling everyone to ask, people who have promised they can do 6pm Tuesday must reserve that slot.
  - You need to remember the decision and tell anyone whom you have not been able to reach during commit/abort phase.

- That is exactly how 2PC works.

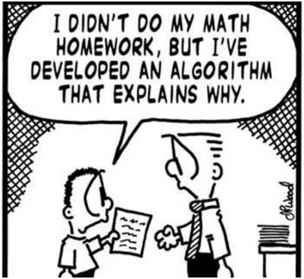
The 2PC Players

- Coordinator (Transaction Manager)
  - Begins transaction.
  - Responsible for commit/abort.

- Participants (Resource Managers)
  - The servers with the data used in the distributed transaction.

The 2PC Algorithm

- Phase 1 - prepare phase
- Phase 2 - commit phase



The 2PC Algorithm - Prepare Phase

- Coordinator asks each participant `canCommit`.

The 2PC Algorithm - Prepare Phase

- Coordinator asks each participant `canCommit`.
- Participants must prepare to commit using permanent storage before answering `yes`.

The 2PC Algorithm - Prepare Phase

- Coordinator asks each participant `canCommit`.

- Participants must prepare to commit using permanent storage before answering `yes`.
    - Lock the objects.
    - Participants are not allowed to cause an abort after it replies `yes` to `canCommit`.

The 2PC Algorithm - Prepare Phase

- Coordinator asks each participant `canCommit`.
- Participants must prepare to commit using permanent storage before answering `yes`.
  - Lock the objects.
  - Participants are not allowed to cause an abort after it replies `yes` to `canCommit`.
- Outcome of the transaction is uncertain until `doCommit` or `doAbort`.
  - Other participants might still cause an abort.

The 2PC Algorithm - Commit Phase

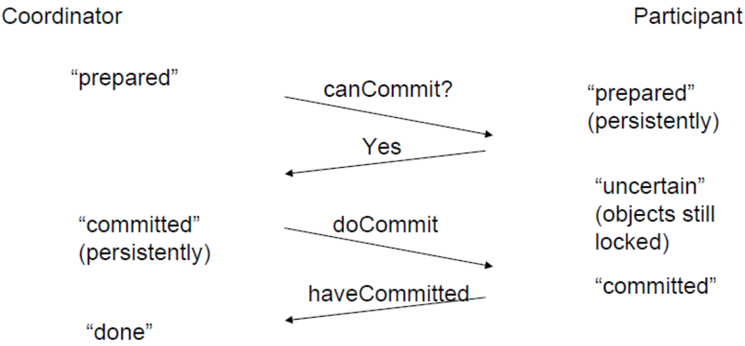
- The coordinator collects all votes.
  - If unanimous yes, causes commit.
  - If any participant voted no, causes abort.

The 2PC Algorithm - Commit Phase

- The coordinator collects all votes.
  - If unanimous yes, causes commit.
  - If any participant voted no, causes abort.

- The fate of the transaction is decided atomically at the coordinator, once all participants vote.
  - Coordinator records fate using permanent storage.
  - Then broadcasts `doCommit` or `doAbort` to participants.

2PC Sequence of Events



Recovery in 2PC

- Recovery after timeouts.
- Recovery after crashes and reboot.

Recovery in 2PC

- Recovery after timeouts.
- Recovery after crashes and reboot.
- Note: you cannot differentiate between the above in a realistic asynchronous network.

Handling Timeout

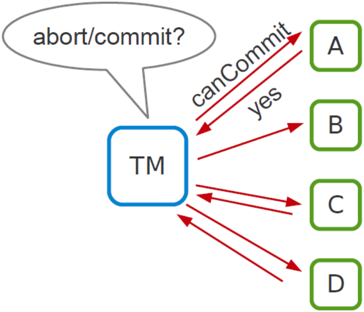
- To avoid processes blocking for ever.

Handling Timeout

- To avoid processes blocking for ever.
- Two scenarios:
  - Coordinator waits for votes from participants.
  - Participant is waiting for the final decision.

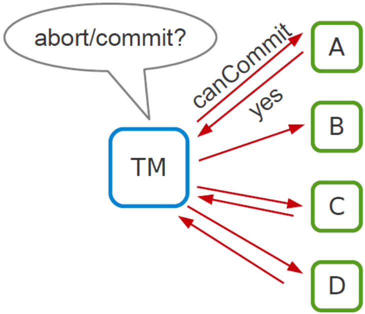
Handling Timeout at Coordinator

- If B voted no, can coordinator unilaterally abort?
- If B voted yes, can coordinator unilaterally abort/commit?



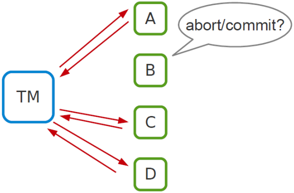
Handling Timeout at Coordinator

- If B voted no, can coordinator unilaterally abort?
- If B voted yes, can coordinator unilaterally abort/commit?
  - Coordinator waits for votes from participants.
  - Participant is waiting for the final decision.
  - Coordinator timeout abort and send `doAbort` to participants.



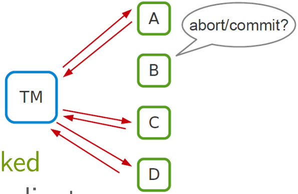
Handling Timeout at Participants

- If B times out on TM and has voted yes, then execute termination protocol.



Handling Timeout at Participants

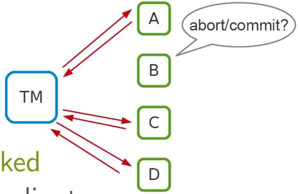
- If B times out on TM and has voted yes, then execute termination protocol.



- Simple protocol: participant is remained blocked until it can establish communication with coordinator.

Handling Timeout at Participants

- If B times out on TM and has voted yes, then execute termination protocol.

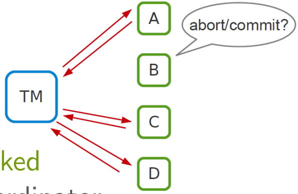


- Simple protocol: participant is remained blocked until it can establish communication with coordinator.

- Cooperative protocol: participant sends a decision-request message to other participants.

Handling Timeout at Participants

- If B times out on TM and has voted yes, then execute termination protocol.

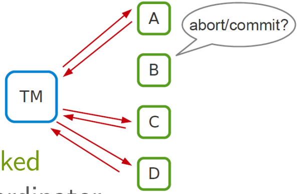


- Simple protocol: participant is remained blocked until it can establish communication with coordinator.

- Cooperative protocol: participant sends a decision-request message to other participants.
  - B sends status message to A

Handling Timeout at Participants

- If B times out on TM and has voted yes, then execute termination protocol.

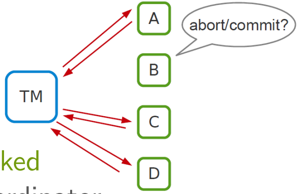


- Simple protocol: participant is remained blocked until it can establish communication with coordinator.

- Cooperative protocol: participant sends a decision-request message to other participants.
  - B sends status message to A
  - If A has received commit/abort from TM, ...

Handling Timeout at Participants

- If B times out on TM and has voted yes, then execute termination protocol.

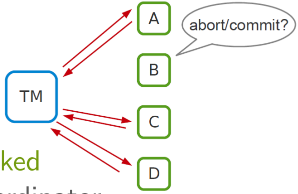


- Simple protocol: participant is remained blocked until it can establish communication with coordinator.

- Cooperative protocol: participant sends a decision-request message to other participants.
    - B sends status message to A
    - If A has received commit/abort from TM, ...
    - If A has not responded to TM, ...

Handling Timeout at Participants

- If B times out on TM and has voted yes, then execute termination protocol.



- Simple protocol: participant is remained blocked until it can establish communication with coordinator.

- Cooperative protocol: participant sends a decision-request message to other participants.
  - B sends status message to A
  - If A has received commit/abort from TM, ...
  - If A has not responded to TM, ...
  - If A has responded with no/yes ...

Handling Crash and Recovery (1/2)

- All nodes must log protocol progress.
  - Participants: prepared, uncertain, committed/aborted
  - Coordinator: prepared, committed/aborted, done

Handling Crash and Recovery (1/2)

- All nodes must log protocol progress.
  - Participants: prepared, uncertain, committed/aborted
  - Coordinator: prepared, committed/aborted, done
- Nodes cannot back out if commit is decided.

Handling Crash and Recovery (2/2)

- Coordinator crashes:
  - If it finds no commit on disk, it aborts.
  - If it finds commit, it commits.

Handling Crash and Recovery (2/2)

- Coordinator crashes:
  - If it finds no commit on disk, it aborts.
  - If it finds commit, it commits.

- Participant crashes:
  - If it finds no yes on disk, it aborts.
  - If it finds yes, runs termination protocol to decide.

Fault-Tolerance Limitations of 2PC

- Even with recovery enabled, 2PC is not really fault-tolerant (or live), because it can be blocked even when one (or a few) machines fail.

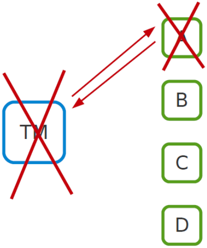
- Blocking means that it does not make progress during the failures.

Fault-Tolerance Limitations of 2PC

- Even with recovery enabled, 2PC is not really fault-tolerant (or live), because it can be blocked even when one (or a few) machines fail.
- Blocking means that it does not make progress during the failures.
- Any scenarios?

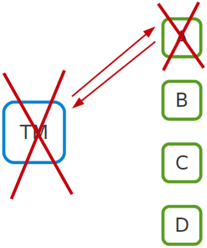
2PC Blocking Scenario

- TM sends `doCommit` to A, A gets it and commits, and then both TM and A die.



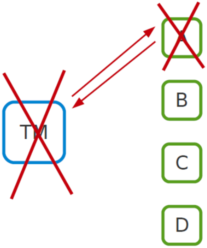
2PC Blocking Scenario

- TM sends `doCommit` to A, A gets it and commits, and then both TM and A die.
- B, C, D have already also replied `yes`, have locked their mutexes, and now need to wait for TM or A to reappear.
  - They cannot recover the decision with certainty until TM or A are online.



2PC Blocking Scenario

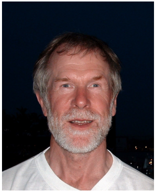
- TM sends `doCommit` to A, A gets it and commits, and then both TM and A die.
- B, C, D have already also replied `yes`, have locked their mutexes, and now need to wait for TM or A to reappear.
  - They cannot recover the decision with certainty until TM or A are online.
- This is why 2PC is called a blocking protocol: 2PC is safe, but not live.



Impossibility of Distributed Consensus with One Faulty Process

FLP

- Fischer-Lynch-Paterson (FLP)
  - M.J. Fischer, N.A. Lynch, and M.S. Paterson, Impossibility of distributed consensus with one faulty process, Journal of the ACM, 1985.



FLP Impossibility Result

- It is impossible for a set of processors in an asynchronous system to agree on a binary value, even if only a single process is subject to an unannounced failure.
- The core of the problem is asynchrony.

FLP Impossibility Result

- What FLP says: you cannot guarantee both safety and progress when there is even a single fault at an inopportune moment.

FLP Impossibility Result

- What FLP says: you cannot guarantee both safety and progress when there is even a single fault at an inopportune moment.
- What FLP does not say: in practice, how close can you get to the ideal (always safe and live)?

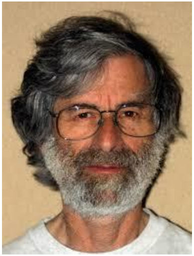
FLP Impossibility Result

- What FLP says: you cannot guarantee both safety and progress when there is even a single fault at an inopportune moment.
- What FLP does not say: in practice, how close can you get to the ideal (always safe and live)?
- So, Paxos ...

Paxos

Paxos

- The only known completely-safe and largely-live agreement protocol.
- L. Lamport, The part-time parliament, ACM Transactions on Computer Systems, 1998.



The Paxos Players

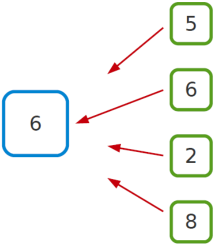
- Proposers
  - Suggests values for consideration by acceptors.
- Acceptors
  - Considers the values proposed by proposers.
  - Renders an accept/reject decision.
- Learners
  - Learns the chosen value.

The Paxos Players

- Proposers
  - Suggests values for consideration by acceptors.
- Acceptors
  - Considers the values proposed by proposers.
  - Renders an accept/reject decision.
- Learners
  - Learns the chosen value.
- A node can act as more than one roles (usually 3).

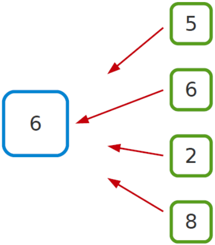
Single Proposal, Single Acceptor

- Use just one acceptor
  - Collects proposers' proposals.
  - Decides the value and tells everyone else.



Single Proposal, Single Acceptor

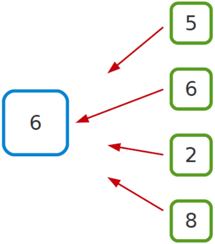
- Use just one acceptor
  - Collects proposers' proposals.
  - Decides the value and tells everyone else.



- Sounds familiar?

Single Proposal, Single Acceptor

- Use just one acceptor
  - Collects proposers' proposals.
  - Decides the value and tells everyone else.



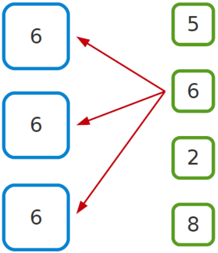
- Sounds familiar?
  - two-phase commit (2PC)
  - acceptor fails = protocol blocks

Single Proposal, Multiple Acceptors

- One acceptor is not fault-tolerant enough.
- Let's have multiple acceptors.

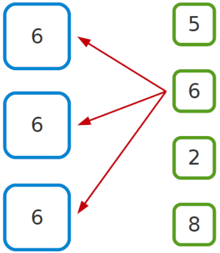
Single Proposal, Multiple Acceptors

- One acceptor is not fault-tolerant enough.
- Let's have multiple acceptors.



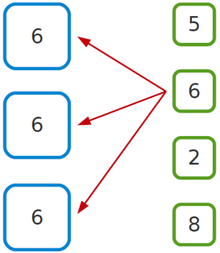
Single Proposal, Multiple Acceptors

- One acceptor is not fault-tolerant enough.
- Let's have multiple acceptors.
- From there, must reach a decision. How?



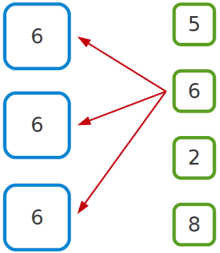
Single Proposal, Multiple Acceptors

- One acceptor is not fault-tolerant enough.
- Let's have multiple acceptors.
- From there, must reach a decision. How?
- Decision = value accepted by the majority.



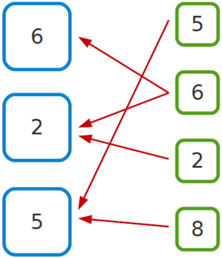
Single Proposal, Multiple Acceptors

- One acceptor is not fault-tolerant enough.
- Let's have multiple acceptors.
- From there, must reach a decision. How?
- Decision = value accepted by the majority.
- P1: an acceptor must accept first proposal it receives.



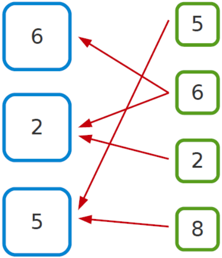
Multiple Proposals, Multiple Acceptors

- If there are multiple proposals, no proposal may get the majority.
  - 3 proposals may each get 1/3 of the acceptors.



Multiple Proposals, Multiple Acceptors

- If there are multiple proposals, no proposal may get the majority.
  - 3 proposals may each get 1/3 of the acceptors.



- Solution: acceptors can accept multiple proposals, distinguished by a unique proposal number.

Handling Multiple Proposals

- All chosen proposals must have the same value.

Handling Multiple Proposals

- All chosen proposals must have the same value.
- P2: If a proposal with value v is chosen, then every higher-numbered proposal that is chosen also has value v.

Handling Multiple Proposals

- All chosen proposals must have the same value.
- P2: If a proposal with value v is chosen, then every higher-numbered proposal that is chosen also has value v.
    - P2a: ... accepted ...

Handling Multiple Proposals

- All chosen proposals must have the same value.
- P2: If a proposal with value v is chosen, then every higher-numbered proposal that is chosen also has value v.
  - P2a: ... accepted ...
  - P2b: ... proposed ...

The Paxos Algorithm

- Phase 1a - prepare phase
- Phase 1b - promise phase
- Phase 2a - accept phase
- Phase 2b - accepted phase

Paxos Algorithm

proposer

Phase 1a: **"Prepare"**
Select proposal number* *N* and send a ***prepare(N)*** request to a majority of acceptors.

Phase 1b: **"Promise"**
If N > *number of any previous promises or acceptances*,
* promise to never accept any future proposal less than *N*,
\- send a ***promise(N, U)*** response
(where *U* is the highest-numbered proposal accepted so far (if any))

Phase 2a: **"Accept!"**
If proposer received promise responses from a majority,
\- send an ***accept(N, V)*** request to those acceptors
(where ***V*** is the value of the highest-numbered proposal among the ***promise*** responses, or any value if no ***promise*** contained a proposal)

acceptor

Phase 2b: **"Accepted"**
If N >= *number of any previous promise*,
* accept the proposal
\- send an ***accepted*** notification to the learner

Paxos Algorithm - Prepare Phase

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

Paxos Algorithm - Promise Phase

- If an acceptor receives a `prepare` request with number n greater than that of any prepare request it saw

Paxos Algorithm - Promise Phase

- If an acceptor receives a `prepare` request with number n greater than that of any prepare request it saw
  - It responses `yes` to that request with a promise not to accept any more proposals numbered less than n.

Paxos Algorithm - Promise Phase

- If an acceptor receives a `prepare` request with number n greater than that of any prepare request it saw
  - It responses `yes` to that request with a promise not to accept any more proposals numbered less than n.
  - It includes the highest-numbered proposal (if any) that it has accepted.

Paxos Algorithm - Accept Phase

- If the proposer receives a response `yes` to its `prepare` requests from a majority of acceptors

Paxos Algorithm - Accept Phase

- If the proposer receives a response `yes` to its `prepare` requests from a majority of acceptors
  - It sends an `accept` request to each of those acceptors for a proposal numbered n with a values v, which is the value of the highest-numbered proposal among the responses.

Paxos Algorithm - Accepted Phase

- If an acceptor receives an `accept` request for a proposal numbered n

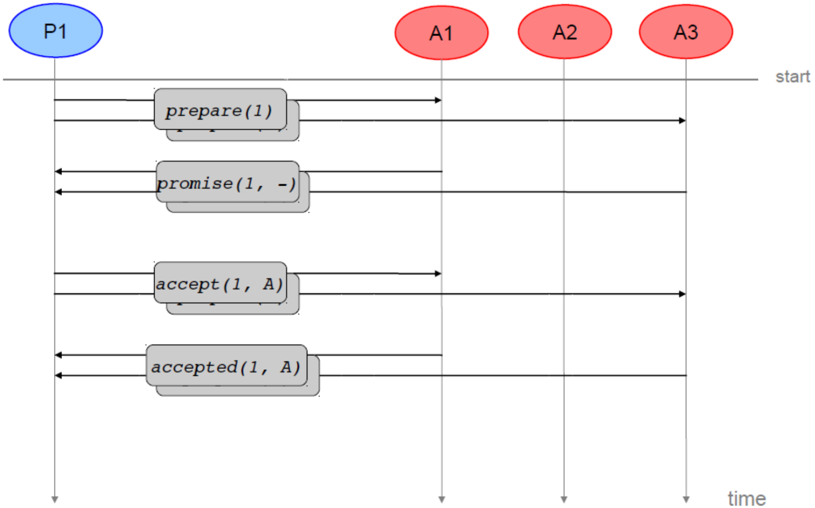
Paxos Algorithm - Accepted Phase

- 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.

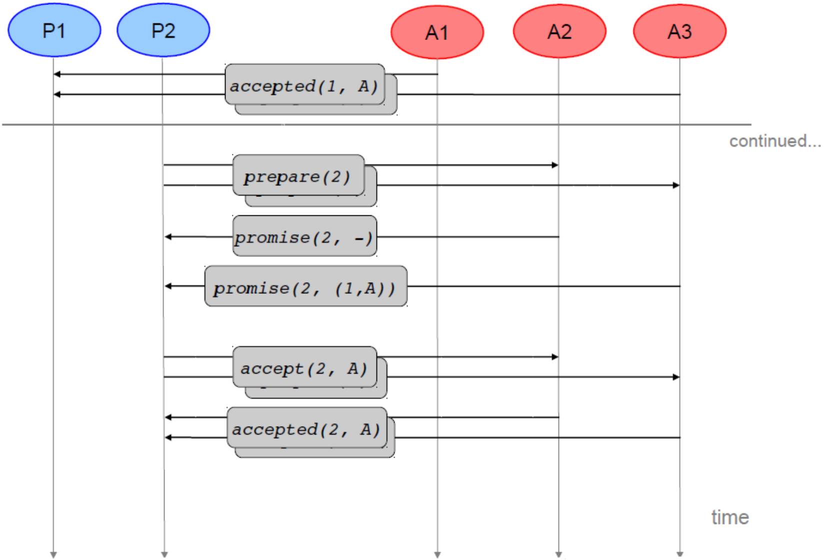
Definition of Chosen

- A value is chosen at proposal number n, iff majority of acceptors accept that value in phase 2 of the proposal number.

Paxos Example



Paxos Example



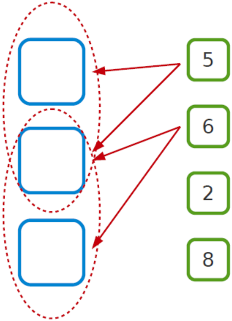
Paxos - Safety (1/3)

- If a value v is chosen at proposal number n, any value that is sent out in phase 2 of any later proposal numbers must be also v.

Paxos - Safety (2/3)

- Decision = Majority (any two majorities share at least one element)

- Therefore after the first round in which there is a decision, any subsequent round involves at least one acceptor that has accepted v.



Paxos - Safety (3/3)

- Now suppose our claim is not true, and let m is the first proposal number that is later than n and in 2nd phase, the value sent out is $w \neq v$.

- This is not possible, because if the proposer P was able to start 2nd phase for w, it means it got a majority to accept round for m (for $m > n$). So, either:
  - v would not have been the value decided, or
  - v would have been proposed by P

- Therefore, once a majority accepts v, that never changes.

Liveness

- If two or more proposers race to propose new values, they might step on each other toes all the time.
  - P_1: `prepare(`n_1`)`
  - P_2: `prepare(`n_2`)`
  - P_1: `accept(`n_1, v_1`)`
  - P_2: `accept(`n_2, v_2`)`
  - P_1: `prepare(`n_3`)`
  - P_2: `prepare(`n_4`)`
  - ...
  - $n_1 < n_2 < n_3 < n_4 < \cdots$

- With randomness, this occurs exceedingly rarely.

Paxos is Everywhere

- Google: Chubby (Paxos-based distributed lock service)
  - Most Google services use Chubby directly or indirectly
- Yahoo: Zookeeper (Paxos-based distributed lock service)
  - Zookeeper is open-source and integrates with Hadoop
- UW: Scatter (Paxos-based consistent DHT)

Summary

Summary

- Replicated State Machine
- 2PC: blocking
- FLP impossibility
- Paxos

References:

- M.J. Fischer, N.A. Lynch, and M.S. Paterson, Impossibility of distributed consensus with one faulty process, Journal of the ACM, 1985.
- L. Lamport, The part-time parliament, ACM Transactions on Computer Systems, 1998.
- L. Lamport, Paxos made simple, ACM Sigact News, 2001.
- J. Gray, and L. Lamport, Consensus on transaction commit, ACM Transactions on Database Systems, 2006.

Questions?

Acknowledgements

Some slides were derived from Jinyang Li slides (New York University).