9.2. 3 Paxos

Overall goals

- Safe
- Nonblocking, making progress if possible
- Resilient to failures, delays and network partitions

Design Ideas

- Combine Leader election with consensus protocol
  - Not an heuristic bolted on top
  - Roles may change on the fly
  - Safe with multiple leader(s), but may not make progress
- Acceptance with quorum/majority:
  - Progress possible if F+1 (out of 2F) voters agree
- Numbered sequence of proposals:
  - Embrace asynchrony

9.2. 3 Paxos

Overall goals

- Safe
- Nonblocking, making progress if possible
- Resilient to failures, delays and network partitions

Design Ideas

- Combine Leader election with consensus protocol
  - Not an heuristic bolted on top
  - Roles may change on the fly
  - Safe with multiple leader(s), but may not make progress
- Acceptance with quorum/majority:
  - Progress possible if F+1 (out of 2F) voters agree
- Numbered sequence of proposals:
  - Embrace asynchrony

Publication History

Several main works

Lamport, Leslie (1998) *The Part-Time Parliament* ACM Transactions on Computer Systems 16 (2): 133–169	Fake and lighthearted "Greek" history to explain the protocol. Not well-received by receivers, rejected and kept unpublished for years
Lamport, Leslie (2001) *Paxos Made Simple* ACM SIGACT News (Distributed Computing Column) 32, 4	Straight-forward write up of the same protocol by the same author in order to prove the simplicity of the algorithm
Tushar Chandra, Robert Griesemer, and Joshua Redstone (2007) *Paxos Made Live - An Engineering Perspective*: 26th ACM PODC	Discussing the sometimes convoluted issues to go from the algorithm to an actual working system

Followup works on optimizations, generalizations, applications, ...

Roles in Paxos

- Client
  - issues a *request* and waits for *response*
  - e.g. „write“-request on a distributed file server or a commit
- Acceptor
  - Acceptors work in *quorums*, a group which is voting on requests.
  - They issue responses and act like the fault-tolerant memory
  - accept only once.
- Proposer
  - tries to convince the Acceptors that the *request* is o.k.
  - coordinates conflicts
- Learner
  - act as replicators.
  - If a client request has been granted (and agreed upon) by the Acceptors, the learners take action
  - e.g. execute the request, send responses to the client
- Leader
  - is a distinguished Proposer
  - if more than one Proposer believe that they are leaders, this conflict needs to be resolved

Basic Idea

- One (or more) node decide(s) to be coordinator/proposer
- Proposes a value and requests acceptance from other nodes (acceptors)
- If it fails, it will try again
- Separate agreement on value from leader acceptance

Consequences and questions

- What happens if there are **multiple** proposers?
- What if these propose **different values**?
- What if there is a **network split**?
- What if a proposer **crashes** at any point?

Use proposal ordering and majorities!

Basic Idea

- One (or more) node decide(s) to be coordinator/proposer
- Proposes a value and requests acceptance from other nodes (acceptors)
- If it fails, it will try again
- Separate agreement on value from leader acceptance

Consequences and questions

- What happens if there are **multiple** proposers?
- What if these propose **different values**?
- What if there is a **network split**?
- What if a proposer **crashes** at any point?

Use proposal ordering and majorities!

Basic Idea

- One (or more) node decide(s) to be coordinator/proposer
- Proposes a value and requests acceptance from other nodes (acceptors)
- If it fails, it will try again
- Separate agreement on value from leader acceptance

Consequences and questions

- What happens if there are **multiple** proposers?
- What if these propose **different values**?
- What if there is a **network split**?
- What if a proposer **crashes** at any point?

Use proposal ordering and majorities!

Basic Idea

- One (or more) node decide(s) to be coordinator/proposer
- Proposes a value and requests acceptance from other nodes (acceptors)
- If it fails, it will try again
- Separate agreement on value from leader acceptance

Consequences and questions

- What happens if there are **multiple** proposers?
- What if these propose **different values**?
- What if there is a **network split**?
- What if a proposer **crashes** at any point?

Use proposal ordering and majorities!

Basic Idea

- One (or more) node decide(s) to be coordinator/proposer
- Proposes a value and requests acceptance from other nodes (acceptors)
- If it fails, it will try again
- Separate agreement on value from leader acceptance

Consequences and questions

- What happens if there are **multiple** proposers?
- What if these propose **different values**?
- What if there is a **network split**?
- What if a proposer **crashes** at any point?

Use proposal ordering and majorities!

Majority voting

Deficiences of existing approaches

- 2 PC (and 3 PC without errors) required positive votes from all participants
- 3 PC with failure in precommit: single node with knowledge sufficient

Majority and Quorums

Paxos requires F+1 nodes to agree out of 2 F acceptors
How does this help?

- **Half** of the acceptors can fail!
- No two **separate** majorities may exist at the **same time**, even if there are network splits
- If two majorities agree on two **distinct,successive** proposals (*a* and *b*), then there is **at least a single node** that is **in both** majorities. This is means that this node has seen and accepted both of them-
- If there there is a third (fourth, etc) majority (*c*,*d*..., then there will be a **set of nodes** that has seen and accepted **all these proposals**.

Majority voting

Deficiences of existing approaches

- 2 PC (and 3 PC without errors) required positive votes from all participants
- 3 PC with failure in precommit: single node with knowledge sufficient

Majority and Quorums

Paxos requires F+1 nodes to agree out of 2 F acceptors

How does this help?

- **Half** of the acceptors can fail!
- No two **separate** majorities may exist at the **same time**, even if there are network splits
- If two majorities agree on two **distinct,successive** proposals (*a* and *b*), then there is **at least a single node** that is **in both** majorities. This is means that this node has seen and accepted both of them-
- If there there is a third (fourth, etc) majority (*c*,*d*..., then there will be a **set of nodes** that has seen and accepted **all these proposals**.

Majority voting

Deficiences of existing approaches

- 2 PC (and 3 PC without errors) required positive votes from all participants
- 3 PC with failure in precommit: single node with knowledge sufficient

Majority and Quorums

Paxos requires F+1 nodes to agree out of 2 F acceptors
How does this help?

- **Half** of the acceptors can fail!
- No two **separate** majorities may exist at the **same time**, even if there are network splits
- If two majorities agree on two **distinct,successive** proposals (*a* and *b*), then there is **at least a single node** that is **in both** majorities. This is means that this node has seen and accepted both of them-
- If there there is a third (fourth, etc) majority (*c*,*d*..., then there will be a **set of nodes** that has seen and accepted **all these proposals**.

Majority voting

Deficiences of existing approaches

- 2 PC (and 3 PC without errors) required positive votes from all participants
- 3 PC with failure in precommit: single node with knowledge sufficient

Majority and Quorums

Paxos requires F+1 nodes to agree out of 2 F acceptors
How does this help?

- **Half** of the acceptors can fail!
- No two **separate** majorities may exist at the **same time**, even if there are network splits
- If two majorities agree on two **distinct,successive** proposals (*a* and *b*), then there is **at least a single node** that is **in both** majorities. This is means that this node has seen and accepted both of them-
- If there there is a third (fourth, etc) majority (*c*,*d*..., then there will be a **set of nodes** that has seen and accepted **all these proposals**.

Majority voting

Deficiences of existing approaches

- 2 PC (and 3 PC without errors) required positive votes from all participants
- 3 PC with failure in precommit: single node with knowledge sufficient

Majority and Quorums

Paxos requires F+1 nodes to agree out of 2 F acceptors
How does this help?

- **Half** of the acceptors can fail!
- No two **separate** majorities may exist at the **same time**, even if there are network splits
- If two majorities agree on two **distinct,successive** proposals (*a* and *b*), then there is **at least a single node** that is **in both** majorities. This is means that this node has seen and accepted both of them-
- If there there is a third (fourth, etc) majority (*c*,*d*..., then there will be a **set of nodes** that has seen and accepted **all these proposals**.

Majority voting

Deficiences of existing approaches

- 2 PC (and 3 PC without errors) required positive votes from all participants
- 3 PC with failure in precommit: single node with knowledge sufficient

Majority and Quorums

Paxos requires F+1 nodes to agree out of 2 F acceptors
How does this help?

- **Half** of the acceptors can fail!
- No two **separate** majorities may exist at the **same time**, even if there are network splits
- If two majorities agree on two **distinct,successive** proposals (*a* and *b*), then there is **at least a single node** that is **in both** majorities. This is means that this node has seen and accepted both of them-
- If there there is a third (fourth, etc) majority (*c*,*d*..., then there will be a **set of nodes** that has seen and accepted **all these proposals**.

Majority voting

Deficiences of existing approaches

- 2 PC (and 3 PC without errors) required positive votes from all participants
- 3 PC with failure in precommit: single node with knowledge sufficient

Majority and Quorums

Paxos requires F+1 nodes to agree out of 2 F acceptors
How does this help?

- **Half** of the acceptors can fail!
- No two **separate** majorities may exist at the **same time**, even if there are network splits
- If two majorities agree on two **distinct,successive** proposals (*a* and *b*), then there is **at least a single node** that is **in both** majorities. This is means that this node has seen and accepted both of them-
- If there there is a third (fourth, etc) majority (*c,d...*, then there will be a **set of nodes** that has seen and accepted **all these proposals**.

Proposal ordering - Idea

Challenges of single round and timeouts in 2PC

- Single proposer is a single point of failure (crash)
- Timeouts assume synchronous message propagation and processing
- Interaction of network splits, crashes and message delays not addressed

Multiple concurrent proposals

- Allow multiple proposers and proposals to be active at the same time
- Acceptors need to make their decision on which proposals to accept and which to reject
- We need need at least unique identifiers for proposals
- Providing a (global) order among proposal works even better

Proposal ordering - Idea

Challenges of single round and timeouts in 2PC

- Single proposer is a single point of failure (crash)
- Timeouts assume synchronous message propagation and processing
- Interaction of network splits, crashes and message delays not addressed

Multiple concurrent proposals

- Allow multiple proposers and proposals to be active at the same time
- Acceptors need to make their decision on which proposals to accept and which to reject
- We need need at least unique identifiers for proposals
- Providing a (global) order among proposal works even better

Naive approaches

Accept first, reject later proposals

- Only a single majority possible (⟹ safe)
- What about simultaneous or out-of-order proposals?
- What about proposer crashes?

Accept newer proposals

- Requires global ordering
- Withdraw support on older proposals
- Make progress on dead and alive proposers (⟹ life)
- What about proposers that are slow but not dead: multiple “winners”

Naive approaches

Accept first, reject later proposals

- Only a single majority possible ($\Longrightarrow$ safe)
- What about simultaneous or out-of-order proposals?
- What about proposer crashes?

Accept newer proposals

- Requires global ordering
- Withdraw support on older proposals
- Make progress on dead and alive proposers ($\Longrightarrow$ life)
- What about proposers that are slow but not dead: multiple “winners”

Paxos solution

Dealing with multiple proposals

- Use globally ordered proposal numbers,e.g., site-id:local-number
- Only accept the most recent proposal:
  - Acceptors will keep track on the highest proposal number it has accepted
  - Proposals with lower numbers will be rejected
- On acceptance, tell the new proposer the previously accepted value

Consequences

- Progress will be made towards more recent proposals
- Values with be learned/retained on leader change
- Value proposed by leader may not be the outcome of the consensus
- The set of consenting nodes will at least stay at the same size
- The proposer will find and preserve a consensus if it already exists

Paxos solution

Dealing with multiple proposals

- Use globally ordered proposal numbers,e.g., site-id:local-number
- Only accept the most recent proposal:
  - Acceptors will keep track on the highest proposal number it has accepted
  - Proposals with lower numbers will be rejected
- On acceptance, tell the new proposer the previously accepted value

Consequences

- Progress will be made towards more recent proposals
- Values with be learned/retained on leader change
- Value proposed by leader may not be the outcome of the consensus
- The set of consenting nodes will at least stay at the same size
- The proposer will find and preserve a consensus if it already exists

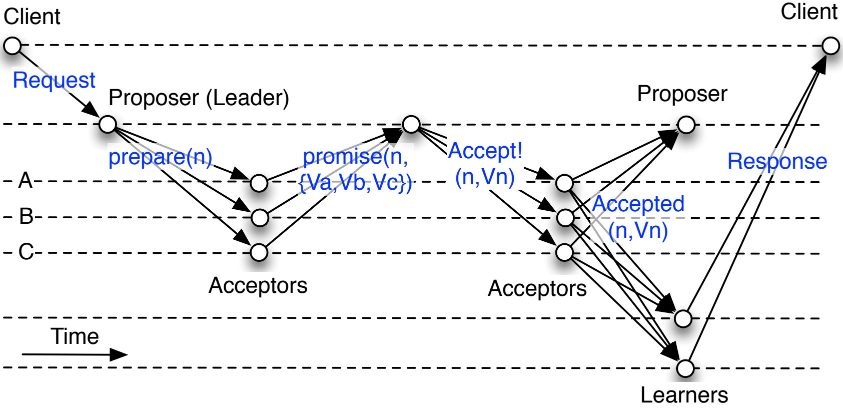
Basic Paxos - First Phase

- Phase 1a: Prepare
  - The Proposer (the Leader) selects a proposal number *n* and sends a **prepare** message to a Quorum of Acceptors
- Phase 1b: Promise
  - If the proposal number *n* is larger than any previous proposal
    - then each Acceptor promises not to accept proposals with a proposal number less than *n*
    - and sends a **promise** message including proposal number and value
  - otherwise the Acceptor sends a denial
  - Also each Acceptor sends the value and number of its last accepted or promised proposal to the Proposer

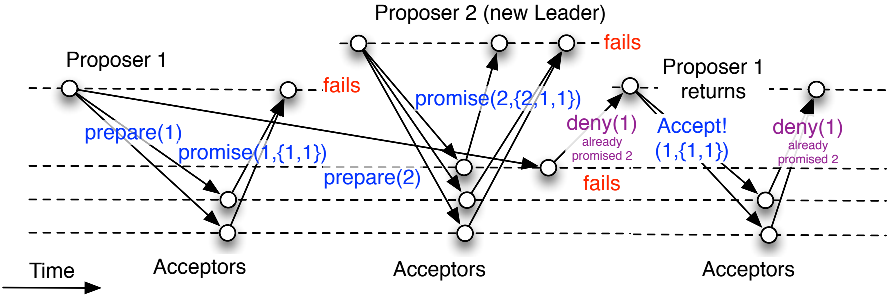
Basic Paxos - Second Phase

- Phase 2a: Accept!
    - If the Proposer receives (positive) responses from a Quorum of Acceptors
        - it may **choose** a value to be agreed upon
        - this value must be from the values of the Acceptors that have already accepted a value
        - otherwise the proposer can choose any value.
    - The Proposer sends an **accept!** message to a quorum of Acceptors including the chosen value
- Phase 2b: Accepted
    - If the Acceptor receives an **accept!** message for the most recent proposal it has promised,
        - it accepts the value
        - each Acceptor sends an **accepted** message to the proposer and every Learner.
    - otherwise it sends a denial and the last proposal number and value it has promised to accept

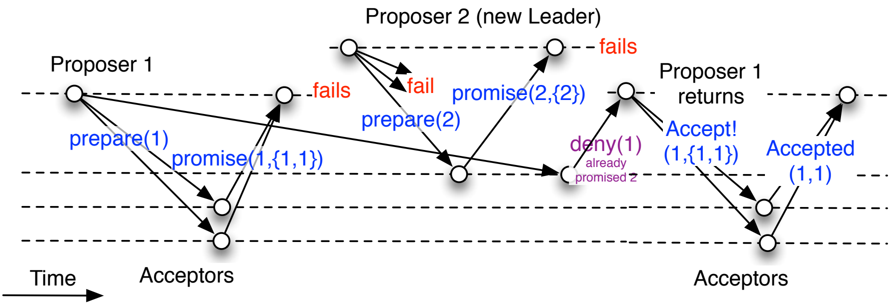
Basic Paxos — without Errors



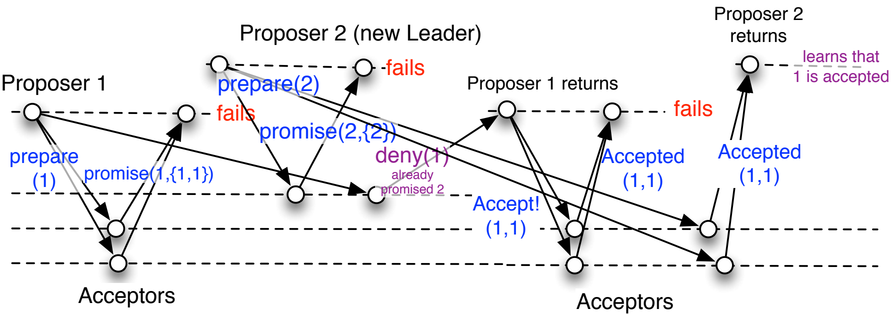
Basic Paxos — Failures and no Value Accepted



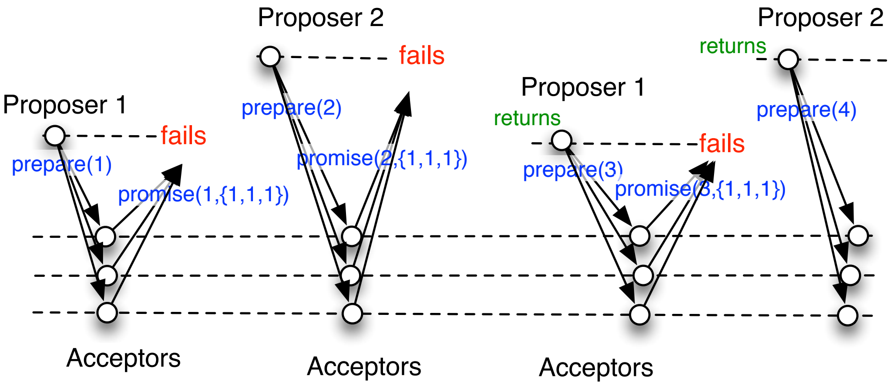
Basic Paxos — Failures and the First Value Accepted



Basic Paxos — Consistency in Time



Basic Paxos — Termination not Guaranteed



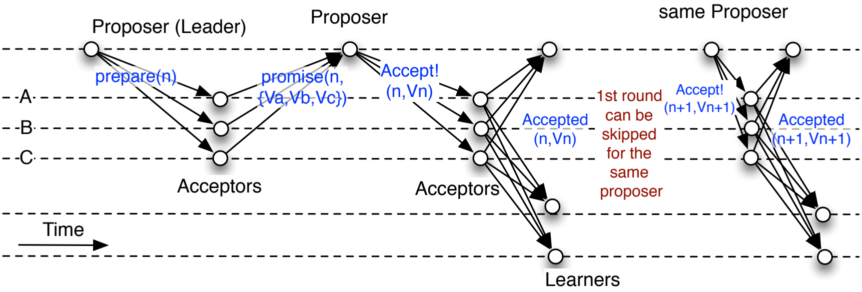
Leader election concerns

- Paxos is safe in the presence of multiple leaders
- Paxos is not guaranteed to make progress, since new leaders may ursurp and the block each other(livelock)
- Tradeoff in leader election:
  - Aggressive: livelock, costly in terms of messages
  - Reluctant: protocol stalled
- Partial solution: use suitable leader election timeout
- There is more we can do!

Multi-Paxos

- Paxos can be optimized regarding Message Complexity
- The first round can be skipped if the proposer stays the same.
- Then, the previous 2nd round plays the role of the following 1st round.
- Only the proposer is allowed to skip the 2nd round who succeeded in the 1st round.
- This way, the delay reduces to two round and the number of messages reduce to the quorum
- This implementation is called *Multi-Paxos*
- Leader in Multi-Paxos is often called *Master*

Multi-Paxos — Reducing the Delay and the Message Complexity



Further Optimizations

- Learners
  - A single distinguished Learner serves as relay and informs the other Learners when a value has been chosen
  - In most applications the role of the leader includes the role of the distinguished Learner
- Quorum communication
  - The leader may send *prepare* and *accept* only to a quorum
  - Other acceptors do not need to be bothered unless they are needed
- Hashing the value: Instead of sending the value, it suffices to send cryptographic secure hash values