Consensus in Distributed Systems

Gkountouvas Theodoros
tg294@cornell.edu

Advanced Systems (CS6410)
Department of Computer Science
Cornell University

October 25, 2012

Presentation

1. Definition of the Problem
2. Paxos Made Simple
3. Paxos Made Moderately Complex
4. Different Types of Paxos
5. Discussion

Consensus Meaning

- **In Real World:** A group of people reaches an agreement after discussion.
- **In Distributed Systems:** A group of process agrees on a specific value.

Safety Requirements

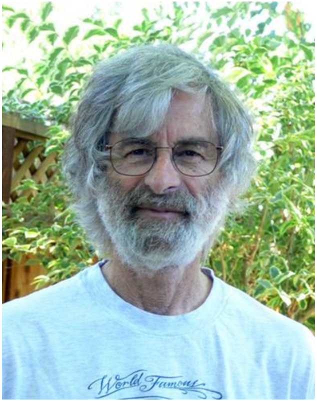
- Only a value that has been proposed may be chosen.
- Only a single value is chosen.
- The majority processes learn that the same value is chosen.

Assumptions

- Asynchronous environment
    - no bounds on timing characteristics
    - clocks run arbitrarily fast
    - message communication takes arbitrarily long
- Crash failures
    - processes just halt in case of failure
- Reliable links
    - messages will eventually be delivered
    - messages can be duplicated and reordered
    - communication is not corrupted

Paxos

Leslie Lamport: Researcher at Microsoft



Paxos Made Simple (2001): Simple description of Paxos protocol.

Classes of Agents

- **Proposers:** Propose values (possibly different) to acceptors.
- **Acceptors:** Choose a value amongst the proposed ones.
- **Learners:** Learn the correct chosen value from the acceptors.

* A process can act as a multi-agent.

Single Acceptor

- Proposers send proposals to a single Acceptor.
- The Acceptor chooses the first value it receives.
- **Problem:** If the Acceptor fails, further progress is impossible.
- **Solution:** Utilize multiple Acceptor agents.

Multi-Acceptors

- In a t fault-tolerant environment, 2t+1 Acceptors are needed.

- Proposers send their proposal to a set of processes, that consists of the majority of Acceptors.

- A value is chosen when at least t+1 Acceptors have accepted this value.

Proposal Format

- A proposal consists of a tuple (n, v), where n is a proposal id and v is the value assigned to this proposal.

- Each proposer has a unique set of proposal ids.

- Uniqueness is guaranteed for proposal ids.

Invariants

- **P1**: An Acceptor must accept the first proposal that it receives.

Invariants

- **P1**: An Acceptor must accept the first proposal that it receives.
- **Problem:** If an Acceptor accepts only one value, then there are scenarios where consensus is impossible.

Invariants

- **P1**: An Acceptor must accept the first proposal that it receives.
- **Problem:** If an Acceptor accepts only one value, then there are scenarios where consensus is impossible.
- **Solution:** An Acceptor must accept multiple values.

Invariants

- **P2**: If a proposal (n, v) is chosen, then for every proposal with id $n' > n$ chosen, the value must be v.

Invariants

- **P2**: If a proposal (n, v) is chosen, then for every proposal with id $n' > n$ chosen, the value must be v.

$$\Uparrow$$

- **P2a**: If a proposal (n, v) is chosen, then for every proposal with id $n' > n$ accepted, the value must be v.

Invariants

- **P2**: If a proposal (n, v) is chosen, then for every proposal with id $n' > n$ chosen, the value must be v.

$$\Uparrow$$

- **P2a**: If a proposal (n, v) is chosen, then for every proposal with id $n' > n$ accepted, the value must be v.

$$\Uparrow$$

- **P2b**: If a proposal (n, v) is chosen, then for every proposal with id $n' > n$ issued by any proposer the value must be v.

Invariants

- **P2c**: For any proposal (n, v), there is a set S consisting of a majority of Acceptors such that one of the following is true.
  (a) No Acceptor in S has accepted any proposal with number $n' < n$.
  (b) The value v is the value of the highest-numbered proposal among all proposals with number $n' < n$ accepted by the acceptors in S.

Invariants

- **P2c**: For any proposal (n, v), there is a set S consisting of a majority of Acceptors such that one of the following is true.
  (a) No Acceptor in S has accepted any proposal with number $n' < n$.
  (b) The value v is the value of the highest-numbered proposal among all proposals with number $n' < n$ accepted by the acceptors in S.

$$\Downarrow$$

P2

Synod Algorithm

- Phase 1: Prepare
  - (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 the greatest proposal number it has ever responded to, then it doesn't respond to proposals with number less than *n* and replies with the highest-numbered proposal that it has accepted.

Synod Algorithm

- Phase 2: Accept
  (a) If the proposer receives a response from majority of acceptors, it sends an *accept request* with (n, v), where v is the highest value in the responses or any value if none responded with a value.

  (b) If an Acceptor receives a *accept request* with number n it accepts the proposal unless it received a *prepare request* with number $n' > n$.

Learners

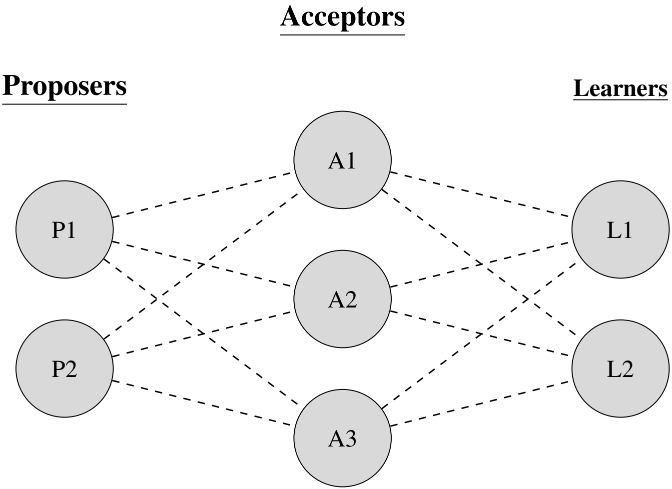
- Learners learn from Acceptors the accepted values and output the value that is proposed by the majority of them.

- In a t fault-tolerant environment, t+1 Learners are needed.

- **Broadcast:** All Acceptors forward to all Learners.

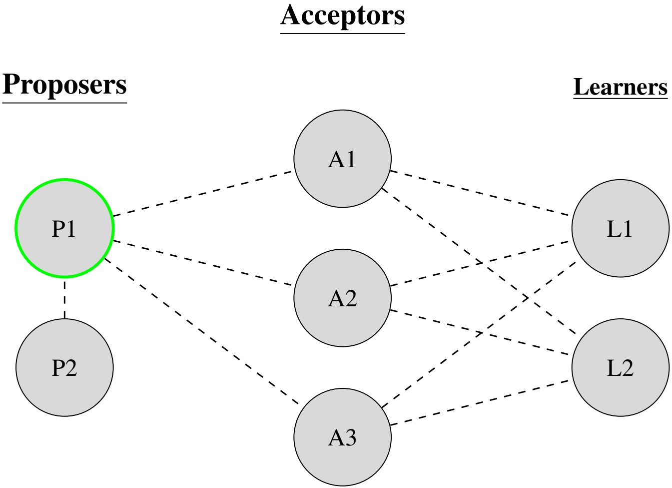
Optimizations

Basic Paxos



Optimizations

Basic Paxos with distinguished Proposer (Leader)



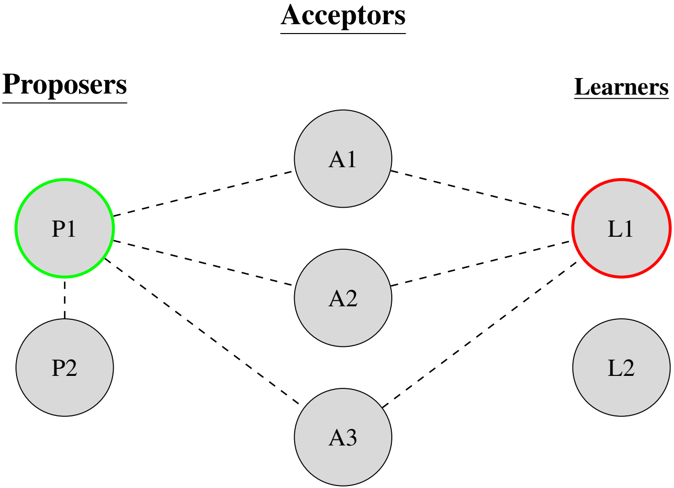
Optimizations

In case that Leader fails:

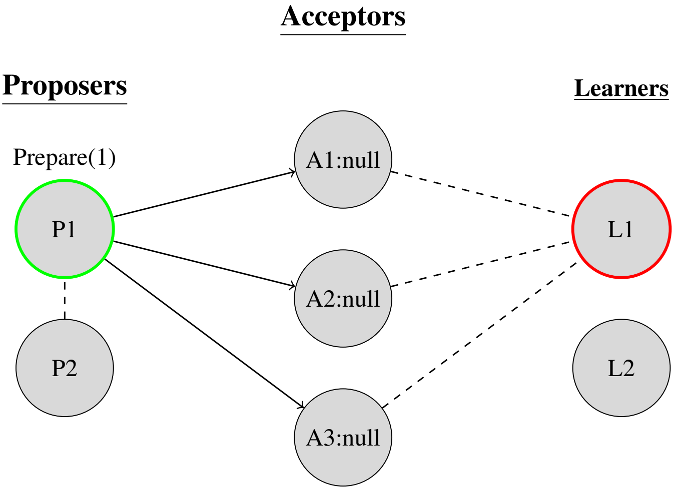
- The protocol must elect a new Leader. Is this another consensus problem?
- After the failed processor recovers it might continue to act as a Leader. This may lead to multiple Leaders.
- The protocol runs safely even with multiple Leaders

Optimizations

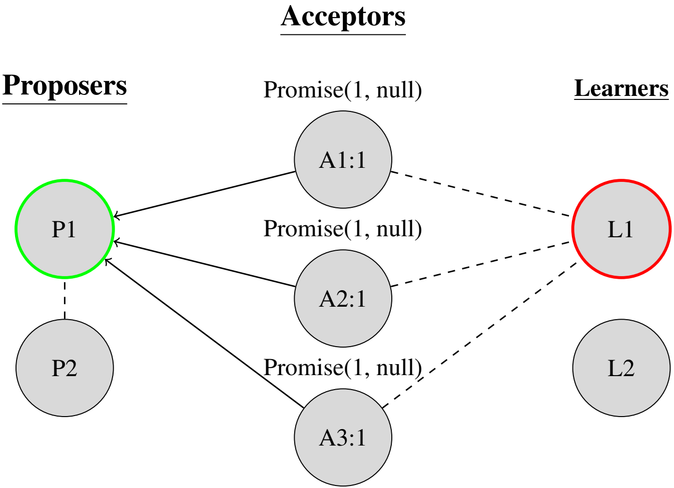
Basic Paxos with distinguished Learner (Leader)



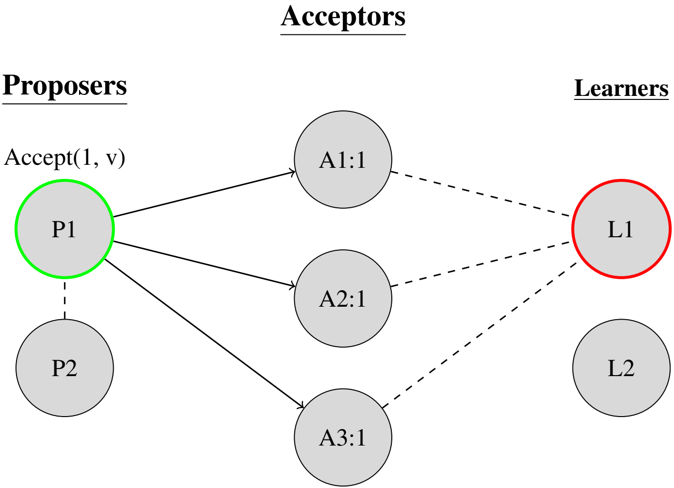
Example


Acceptors
Proposers
Learners
Prepare(1)
P1
P2
A1:null
A2:null
A3:null
L1
L2

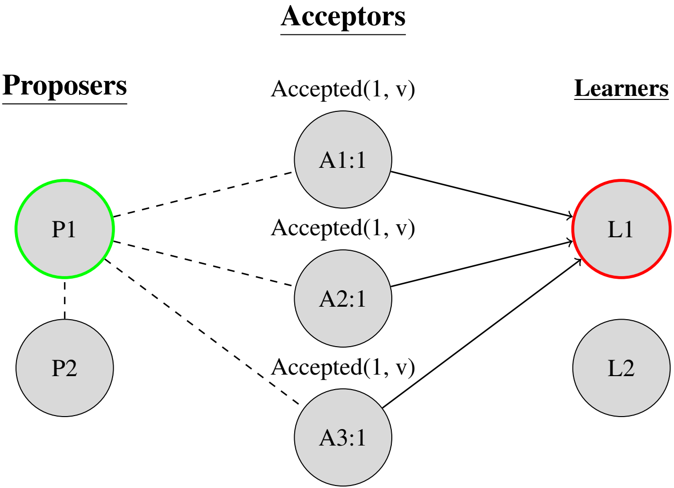
Example


Acceptors
Proposers
Learners
Promise(1, null)
Promise(1, null)
Promise(1, null)
P1
P2
A1:1
A2:1
A3:1
L1
L2

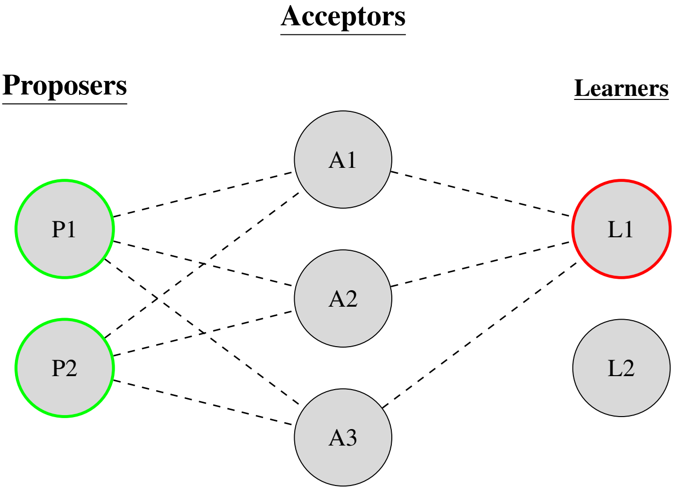
Example


Acceptors
Proposers
Learners
Accept(1, v)
P1
P2
A1:1
A2:1
A3:1
L1
L2

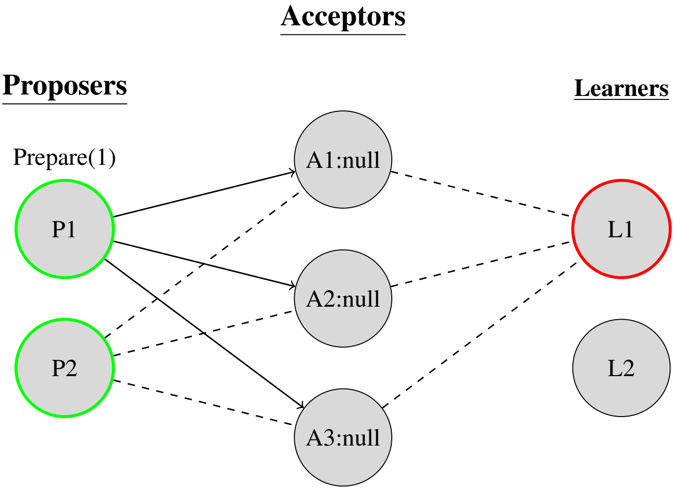
Example


Acceptors
Proposers
Accepted(1, v)
Learners
A1:1
P1
Accepted(1, v)
L1
A2:1
P2
Accepted(1, v)
L2
A3:1

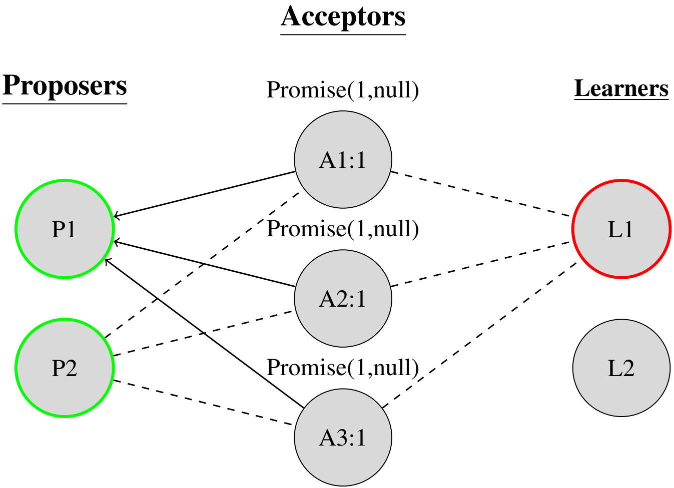
Progress



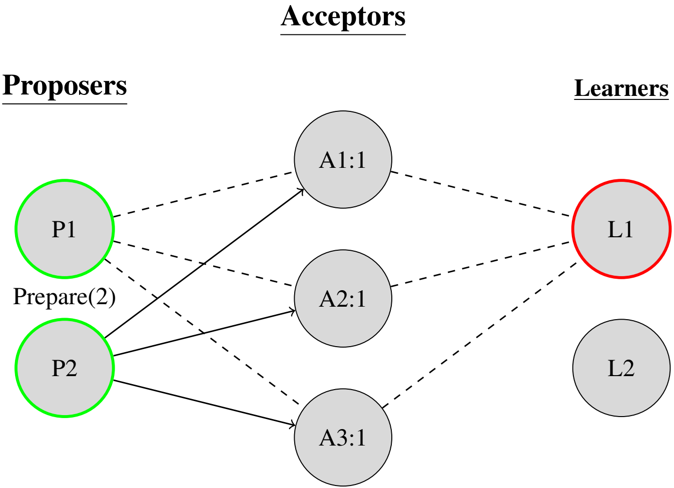
Progress


Acceptors
Proposers
Learners
Prepare(1)
P1
P2
A1:null
A2:null
A3:null
L1
L2

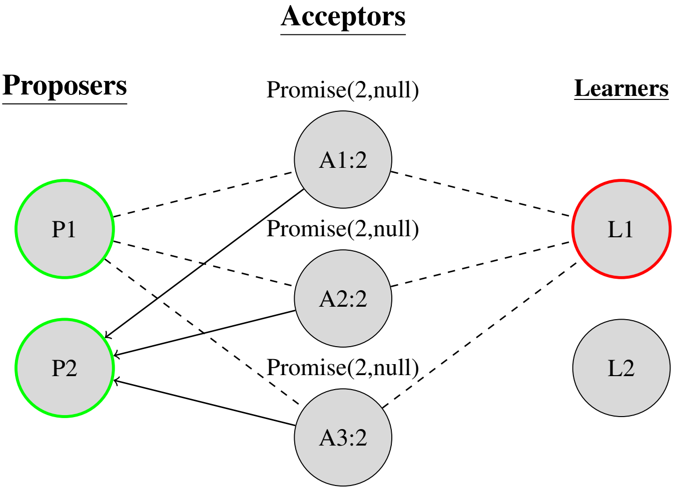
Progress


Acceptors
Proposers
Learners
Promise(1,null)
Promise(1,null)
Promise(1,null)
A1:1
A2:1
A3:1
P1
P2
L1
L2

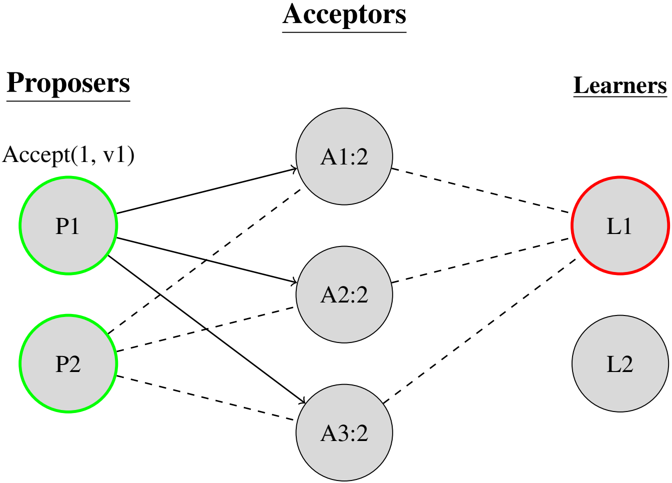
Progress


Acceptors
Proposers
Learners
A1:1
P1
L1
A2:1
Prepare(2)
P2
L2
A3:1

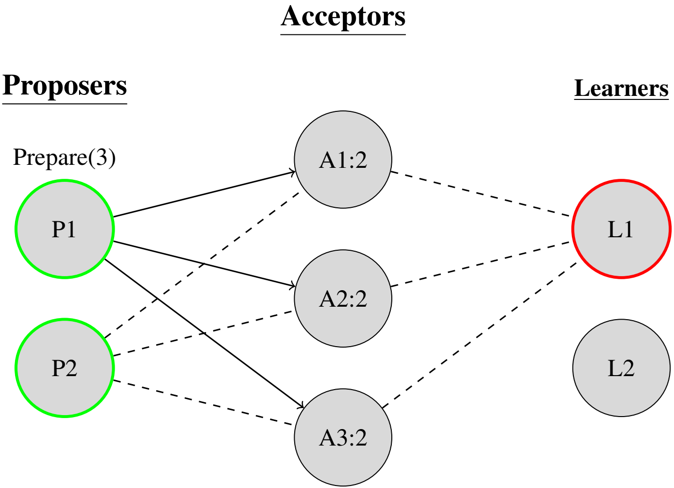
Progress


Acceptors
Proposers
Promise(2,null)
Learners
A1:2
P1
Promise(2,null)
L1
A2:2
P2
Promise(2,null)
L2
A3:2

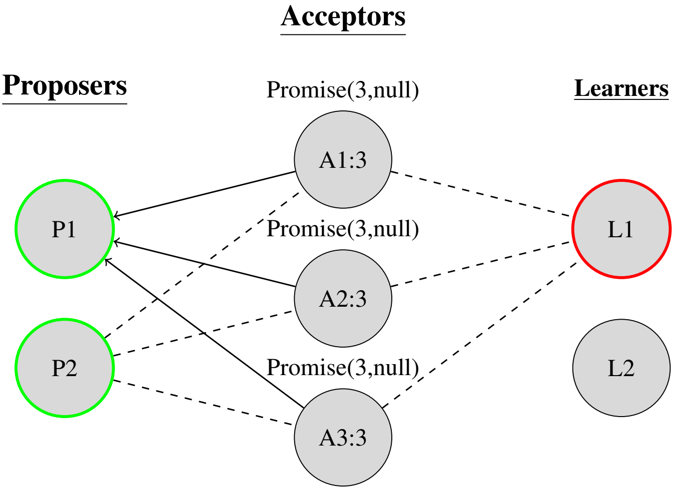
Progress


Acceptors
Proposers
Learners
Accept(1, v1)
P1
P2
A1:2
A2:2
A3:2
L1
L2

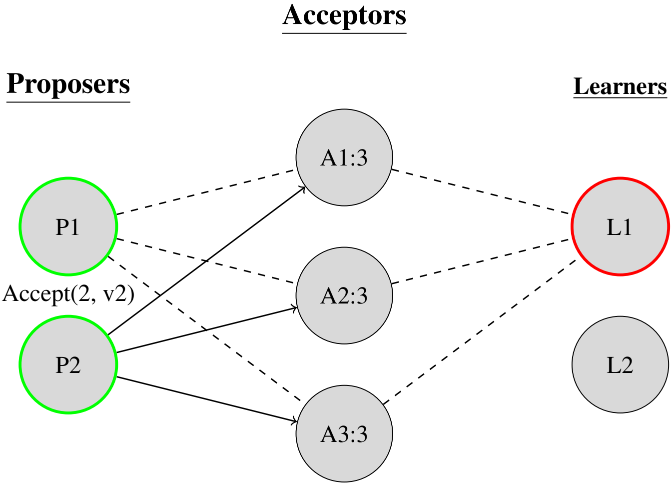
Progress


Acceptors
Proposers
Learners
Prepare(3)
P1
P2
A1:2
A2:2
A3:2
L1
L2

Progress


Acceptors
Proposers
Promise(3,null)
Learners
A1:3
P1
Promise(3,null)
L1
A2:3
P2
Promise(3,null)
L2
A3:3

Progress



Acceptors

Proposers

Learners

Progress

- **Theoretically:** Asynchronous environment and crash failure model lead to no Progress. *Impossibility of Distributed Consensus with One Faulty Process (1983)*
- **Practically:** Countermeasures can be taken to avoid this domino effect.
  - randomized timeouts
  - failure detection

Implementation of Paxos

- How the leaders are elected?
- What happens when multiple requests are spawned?
- How I get rid of redundant data?
- How do I achieve liveness requirement?

Paxos Made Moderately Complex

Robbert Van Renesse: Research Scientist at Cornell

Paxos Made Moderately Complex (2011): Difficulties in implementation of Paxos protocol.

State Machine

- Collection of states.
- Collection of transitions between states.
- Current state.

Deterministic: For any state and operation the transition is unique.

SMR: Masks failures via replication. It is assumed that at least one replica never crashes.

Problem

- Multiple clients

Problem

- Multiple clients

$$\Downarrow$$

- Multiple concurrent commands are executed with different order at the replicas.

Problem

- Multiple clients

$$\Downarrow$$

- Multiple concurrent commands are executed with different order at the replicas.

$$\Downarrow$$

- Replicas make different transitions and are inconsistent with each other.

Problem

- Multiple clients

$$\Downarrow$$

- Multiple concurrent commands are executed with different order at the replicas.

$$\Downarrow$$

- Replicas make different transitions and are inconsistent with each other.

Solution: Utilize Synod algorithm to agree on the order of commands.

Clients

- Clients make requests of type (k, cid, op).
  - *k* -> client unique id
  - *cid* -> command id
  - *op* -> operation to be performed
- They wait until they get a response.
- Clients should not be able to witness SMR model with failures. Instead, the system must behave like a single SM without failures.

Classes of agents

- **Replicas:** They are t+1 processes that guarantee t fault tolerance. They interact with the Clients.
- **Leaders:** They are placed between Replicas and Acceptors.
  - **Scouts:** execute first phase of Paxos.
  - **Commanders:** execute second phase of Paxos.
- **Acceptors:** They are 2t+1 processes. The majority is needed in order to reach a decision.

Slots and Ballots

Slots

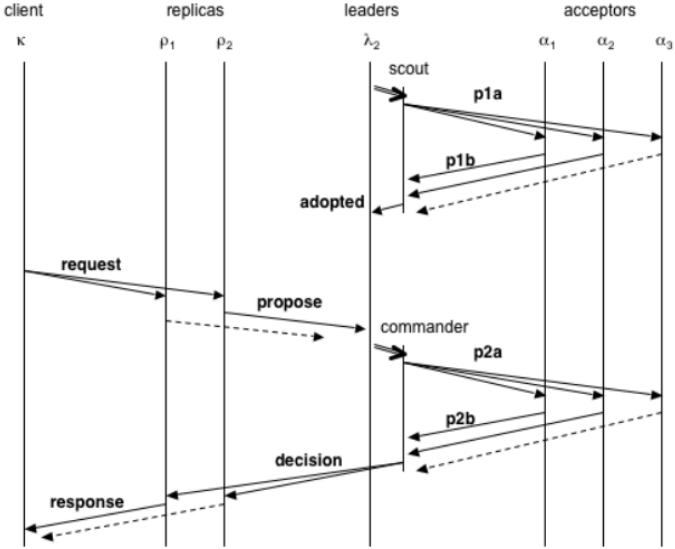
- contain commands in the order of execution
- each slot contains a unique command
- each command can be in multiple slots

Ballots

- there are tuples (λ, id) where λ is the Leader they belong to and id is a unique number for the ballot

PValues

- triple (b, s, p) where b is a ballot, s is a slot and p is the proposed command


client
replicas
leaders
acceptors
κ
ρ1
ρ2
λ2
α1
α2
α3
scout
p1a
p1b
adopted
request
propose
commander
p2a
p2b
decision
response

Liveness

- **Problem:** Liveness is not guaranteed.

- **Weaken Assumptions:** There is a bound
  - in clock drifts
  - in communication time between two non-faulty processes

- **Solutions:**
  - failure detection
  - TCP-like timeout mechanism

State Reduction

- Acceptors keep the highest PValues for each slot.
- Acceptors sent information only for slots that are undecided.
- Replicas can keep only the requests higher to their *slot_num*.
- Leaders spawn Commanders only for undecided slots.

Garbage Collection

- Acceptors do not need to keep PValues for slots that have been updated to all Replicas.

- A faulty Replica can stall the garbage collection.

- Have $2t + 1$ Replicas instead of $t + 1$. Acceptors erases the PValue when more than t Replicas have performed the corresponding command.

- A recovered Replica which is not able to learn a particular command will get a snapshot of the state of another Replica.

Co-location

- In practice, the Leaders are usually co-located with the Replicas.
- A Replica instead of broadcasting it sends the proposal to the local Leader. If Leader is active it spawns a Commander to handle the proposal. If not it sends the message to another active Leader (monitor).
- Avoid the expense of the Broadcast.
- Other scenarios of co-locations are possible, as well.

Read-only Commands

- Read operations do not change the state of Replicas. So, we don’t need consensus.
- Use leases mechanism in order to be certain that an update is not going to happen from the other Leader.
- If the Leader has the lease it can attach read-only commands to the highest slot number.

Multi-Paxos

- One Leader fairly stable.
- Skip prepare request after the first one.
- Instead of 4 messages delay we have 2 in the usual case.

Cheap-Paxos

- We have t+1 main Acceptors and t auxiliary Acceptors.
- Dynamic reconfiguration after failures.
- When system is stable the protocol is better.
- The system must halt when too many failures occur. (delay for reconfiguration)

Fast-Paxos

- Requests are made directly to all Acceptors.
- Response to requests goes to Learners and to a single Leader.
- The single Leader detects collisions and solves them with a new accept request.
- If there is not any collision, we have only 2 messages delay instead of 4.
- When collisions happen, we have 4 messages delay, which is the same with the basic Paxos.

Generalized-Paxos

- Partial order of events. Some operations can run concurrently.

- For some applications it is faster than Fast-Paxos algorithm.

Byzantine-Paxos

- Non-Byzantine processors assumption is erased.
- Extra replications are needed for guaranteed correctness.
- Fast-Paxos can be integrated to make it even faster (Fast-Byzantine-Paxos).

Many different versions of the protocol are proposed in literature.

Discussion

- Is Paxos implementation simple?
- Are there ways to weaken the assumptions realistically and obtain more performance gains?
- Is Paxos the only solution?

End of Presentation

Thank you!!!