Paxos Made Simple

Leslie Lamport, 2001
The Problem

- Reaching consensus on a proposed value, among a collection of processes

- Safety requirements:
  - Only a value that has been proposed may be chosen
  - Only a single value is chosen, and
  - A process never learns that a value has been chosen unless it actually has been.
Asynchronous, non-Byzantine model

- Processes communicate by sending messages
  - Processes operate at arbitrary speed
  - Processes may fail by stopping, and may restart
  - Information can be remembered by a process that has failed and restarted

- Messages are asynchronous
  - Might take arbitrarily long to be delivered
  - Might be duplicated and lost
  - Messages cannot be corrupted
Roles

- We have three different roles
  - Proposers
  - Acceptors
  - Learners

- A process may play one or more roles
Choosing a value

Two main requirements:
- P1
- P2

With extensions:
- P2^a
- P2^b
- P2^c
- P1^a
An acceptor must accept the first proposal it receives

Ensures that paxos works even if there is only one proposal
If a proposal with value $v$ is chosen, then every higher-numbered proposal that is chosen has value $v$.

Guarantees that only one value will be chosen.
If a proposal with value $v$ is chosen, then every higher-numbered proposal accepted by any acceptor has value $v$.

Ensures P2 will hold.
If a proposal with value \( v \) is chosen, then every higher-numbered proposal issued by any proposer has value \( v \)

Prevents proposers from ruining \( P2^a \) by issuing proposals with values different from \( v \)
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 $S$
Issuing proposals (1)

Algorithm for the proposer:
1. A proposer chooses a new proposal number $n$ and sends a request to a set of acceptors, asking them to respond with:
   a) A promise to never accept a proposal numbered less than $n$, and
   b) The proposal with the highest number less than $n$ that has been accepted, if any
Issuing proposals (2)

2. If the proposer receives the requested responses from a majority of the acceptors, it can issue a proposal with:
   a) number $n$ and value $v$, where $v$ is the value of the highest-numbered proposal among the responses, or
   b) any value if the responders reported no proposals
An acceptor can accept a proposal numbered $n$ if it has not responded to a *prepare* request having a number greater than $n$.

Responding to a prepare request includes promising not to accept any lower-numbered proposals. $P1^a$ makes sure this promise is kept.

By enforcing $P1^a$ we also implement $P1$.
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.
Consensus algorithm: Phase 2

A. If the proposer receives a response to its prepare request (numbered $n$) from a majority of acceptors, then it sends an accept request to each of those acceptors from 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 $n$. 
Example: Step 1

**Proposers**

- P1
  - Prepare for proposal 1
  - Reply to P1

**Acceptors**

- A1
  - Prepare for proposal 1
  - Reply to P1

- A2
  - Prepare for proposal 1

- A3
Example: Step 2

Proposers

P1

P2

Acceptors

A1

Accepted prepare for 1

A2

Accepted prepare for 1
Accepted prepare for 2

A3

Prepare for proposal 2

Reply to P2

Prepare for proposal 2
Example: Step 3

Proposers

Accept request proposal 1: \( v = \text{“pepperoni”} \)

Accept request proposal 1: \( v = \text{“pepperoni”} \)

Acceptors

A1

Accepted prepare for 1

A1: Accepts

A2

Accepted prepare for 2

A2: Declines. Promised A2 to not accept proposals < 2

P1

P2

A3
Example: Step 4 - The chosen value is “mushrooms”
Example: Step 5

Proposers

P1

Prepare for proposal 3

A1

A2

Prepare for proposal 3

Acceptors

A1: Last accepted proposal 1: “Pepperoni”
A2: Last accepted proposal 2: “Mushrooms”

P2

A3
Example: Step 6

Accept request proposal 3: \( v = \text{“Mushrooms”} \)

A1: Accepts
A2: Accepts

The value must match the one from proposal 2
Learning a chosen value

- The learner must find out that a value has been accepted by a majority of acceptors
- Each acceptor informs each learner
  - Requires many messages to be sent
- Distinguished learner
  - Single point of failure
- Compromise with set of distinguished learners?
  - Greater reliability at the cost of greater communication complexity
  - All distinguished learners must fail in order to cause a problem
Progress Isn't Guaranteed!

- Easy to construct a scenario in which proposal never is chosen
  - Proposer $p_1$ sends prepare for 1
  - Proposer $p_2$ sends prepare for 2
  - $p_1$ sends accept request for 1; no replies
    - Sends prepare for 3
  - $p_2$ sends accept request 2; no replies
    - Sends prepare for 4
  - $p_1$ sends accept request 3; no replies
    - Sends prepare for 5

...
Guaranteeing progress

- Solution: allow only a distinguished proposer to prepare and issue proposals
  - Proposers send their proposals to the distinguished proposer, who organizes them.
- Can we avoid single-point of failure?
  - New distinguished proposer is elected if the current one fails
  - Many processes may believe they are leaders. However the safety properties are still preserved in that case
  - Little mention of how to do the election...
Implementing a state machine

- Paxos can be used to run a distributed set of state machines
  - Values are executable commands
  - One instance of paxos per command
  - Each process plays all roles
  - Elect one leader to be distinguished proposer and learner
  - An infinite (!) number of instances are executed simultaneously.
Paxos is an algorithm for reaching consensus among processes
- Simple
- Robust
- More or less optimal network use
- Can be used to create distributed state machines