Paxos Made Simple

Link to paper

Paxos is a distributed consensus algorithm developed by an obnoxious twat who described his original algorithm with an extensive Greek parliament analogy. This is his paper coming to terms with the fact that he's a twat.

2. The Consensus Algorithm

2.1 The Problem

If a bunch of values are proposed by `N` processes, how do you choose a value such that:

1. Only a value that was proposed is chosen
2. Only one value is chosen
3. Processes never learn a value that wasn't chosen

We have three kinds of **agents** in a system like this: `proposers`, `acceptors`, and `learners`. Agents communicate via messages, asynchronously and non-Byzantine. Messages can be dropped, lost, take arbitrarily long, but are never corrupted. Likewise, agents can die and restart at will, but they must not lose their state ("what was chosen?") on a restart.

2.2 Choosing a Value

An acceptor (or set of acceptors, requiring a majority) receives a value from a proposer. Requirement 1:

> P1. An acceptor must accept the first proposal that it receives.

But what if multiple values are proposed simultaneously, and each acceptor in a system accepts a different value? No quorum. So new requirement as a consequence: an acceptor must be able to accept multiple proposals, each with a different, incrementing proposal number.

The long-and-short of this is that an acceptor must remember two things:

1. The highest-numbered proposal it's ever **accepted**
2. The highest-numbered prepare request it's ever **received**

A `prepare request` here is part of the two-phase aspect of this - a propose asks an acceptor to accept its proposal AND to send back a response promising to accept no lower-numbered proposals. The `commit` request comes later.

The actual two phases of 2PC are as such:

> Phase 1: a proposer sends a `prepare` request to a quorum of acceptors with proposal number `n` and value `v`. For a given acceptor, if the proposal number `n` is higher than any other proposal number that acceptor has received, it accepts the proposal and promises to the acceptor to never accept a proposal of a lower number.

> Phase 2: if a proposer receives a quorum of prepare responses from acceptors, it sends commit messages to all those acceptors containing a proposal number `n` and value `v` that is the **highest** of any prepare responses sent back to the proposer.

This second phase is how information gets updated!