**Lecture 20 — Paxos** **4/23/2014**

Review: Goal — highly available stateful server [slide]

Review: Replicated state machines [slides]

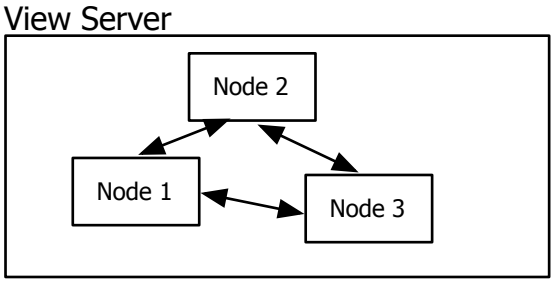
A general approach to making consistent replicas of a server:
- Start with the same initial state on each server.
- Provide each replica with the same input operations, in the same order.
- Ensure all operations are deterministic.
  E.g., no randomness, no reading of current time, etc.

**Challenge**: what if view server fails, or network is partitioned and can't be reached by all servers?

*Need a way to pick a new view server*

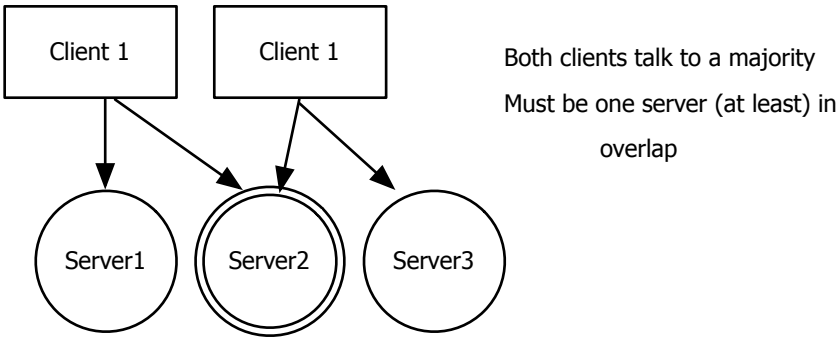
But have to be careful! Old view server may still be alive. Want view server to be one reachable by majority of nodes.

View server is actually n nodes, where n > 3. They run a protocol called Paxos (distributed consensus) that allows them to decide on some value (e.g., the id of the next view server.) They provide single copy consistency for current **view**




Why n>3?

**Majority rule**: to choose a new value, a majority must agree on the value. Has the property that any 2 majority sets must overlap by at least one server, so any majority will contain at least one node that heard about previous majority.



**Solution: Paxos [show goal slide]**

Advanced topic--won't quiz you on the details
Take 6.824 if you want to get into this topic
Paxos: fault-tolerant consensus.
Set of machines can agree on one value. [show properties slide]
E.g., X is the next operation that everyone will execute.
E.g., Y is the next coordinator.
Works despite node failures, network failures, delays.

Correctness:
All nodes agree on the same value.
The agreed-upon value was proposed by some node.
Fault-tolerance:
If less than N/2 nodes fail, rest should reach agreement w.h.p.
Liveness not guaranteed (may take an arbitrarily long time).
Assumes fail-stop machines.

[show setup slide, w/ three logical node types]
**Proposer**: tries to convince acceptors to agree on some value
(e.g., an operation it just received from a client).
**Acceptor**: persistently tracks proposals, vote on proposals.
**Learner**: does something with the value, once agreement is reached
(e.g., just the local RSM replica, which executes the chosen operation).

*Intuition for why fault-tolerant consensus difficult.*

**Strawman 1**: proposers agree on a single acceptor they will propose to.
Acceptor approves the first proposal it receives, and rejects all others.
Correct because only a single acceptor.
Problem: not fault-tolerant!

[draw diagram w/ single acceptor and multiple proposers]

**Strawman 2**: proposers send proposal to all acceptors.
Each acceptor approves the first proposal and rejects all others.
Proposers declare victory after approval from majority of acceptors.
Correct because there's only one majority, so at most one winner.
[diagram with multiple acceptors and proposers]

Problem 1: no one might get a majority, if we have 3 proposers!
Then what? Need to back out somehow, but acceptors are “locked in” to their votes

Problem: proposer might get majority and then crash!
Majority exists, but hasn’t been discovered

**Paxos**: two phases for consensus.

Proposer first tries to get a majority to
  **Prepare**: agree on a proposal number (value might not be decided yet).

If a majority prepare, a proposer can then attempt to get a majority to accept a value
  **Accept**: agree on a value, for given proposal number.

Proposers can fail at any time, and protocol can restart with a different proposer

  Think of proposal numbers as different attempts to reach consensus.
  If one attempt fails (e.g., 3 proposers can't get majority),
    try again with a larger proposal number.

[show paxos rule slide]
  Key requirement: if a majority if nodes in an earlier proposal number accepted a value,
    have to make sure later proposals accept the same value.

Paxos state maintained by acceptor (persistent across reboots):
  Np: largest proposal number seen in Prepare.
  Na: largest proposal number seen in Accept.
  Va: value accepted for proposal Na.
[show code]

some notes:

1) choosing unique N can be done in several ways, e.g., each proposal concatenates its unique node id on to a monotonic counter

2) when acceptor sends prepare, it is “promising” not to accept any proposals less than Np — ensures that proposer will definitely learn about any value accepted previously by a majority

3) value becomes “committed” when a majority of acceptors log the Na/Va to disk — any future proposer will learn this value if it succeeds in contacting a majority of nodes

**Example 1**: Paxos accepting value 'x' with one proposer. [show slides]

What is the commit point -- i.e., when is the value 'x' chosen?
  Majority of acceptors write the corresponding Na / Va to disk.

What if the Accept message to one of the acceptors is lost?
  Before majority of acceptors wrote to disk -> no consensus yet, fine.
  After consensus -> proposer can start a new round, will pick same Va.
  Why?  At least one Prepare_OK reply will include that Va.

**Example 2**: Paxos with two proposers, choosing between 'x' and 'y'. [show slides]
One proposer has N=10, another has N=11 (e.g., low bits are the node ID).

Key point: Accept() checks if it heard about a newer proposal.
If it has, then the older proposal number loses: Accept(10, x) will reject

**Example 3**: [show slides]
If it hasn't, will remember the accepted value: e.g., Prepare(11) is later.
Then, any newer Prepare() will learn about x.

**Example 4**: Paxos with proposer crashing after majority of Accepts are processed. [show slides]

Second proposer tries to agree on a new value y.
Gets Prepare_OK with first proposer's value.
Re-sends Accept messages with first proposer's value.

Key point: proposer conservatively uses a previous value from Prepare_OK().
Even if just one acceptor got it: could be the 1 overlap between majorities.

<u>Why do we need to log things?</u>

What happens if acceptor crashes after sending accept?
Other proposers should learn value that was accepted
Acceptors must record Na/Va persistently on disk.

What happens if acceptor fails after sending Prepare_OK?
Also need to remember Np persistently on disk.
Otherwise, example 2 fails if all acceptors reboot: should not accept (10, x).

<u>Let's review how we can use Paxos for Replicated State Machines:</u>
Variety of ways, e.g., could run a round of paxos agreement for each message number

That'd be pretty expensive - lots of communication

Instead, can use it to choose the view server.

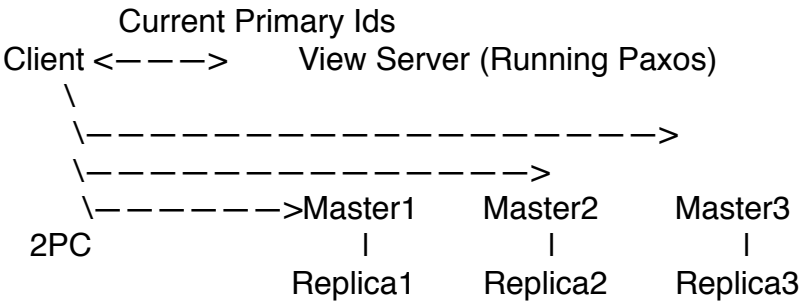
"View server" is really a collection of nodes running Paxos.
They use Paxos to agree on:
- The set of all replicas and the coordinator, for when we need to elect the next coordinator.
- The last operation executed by the current (dead) coordinator, so we don't lose track of it during switch over.

Can also add replicas at runtime: just agree on new replica set.
Need a way to transfer log state to the new replica.

Clients can ask any nodes in the view server group about the current coordinator

This is what is meant in spanner paper by a “paxos write” — a message to a coordinator, that is recovered by electing a new coordinator in the event of a failure.

Recap how spanner works:




[slide: summary]