6.033: Fault Tolerance: Replicated State Machines
Lecture 19
Katrina LaCurts, lacurts@mit.edu

0. Introduction
   - Have multi-site atomicity; want to improve availability via replication.
   - Goal: single-copy consistency
     - Property of the externally-visible behavior of a replicated system
     - Operations appear to execute as if there's only a single copy of the data
   - Is single-copy consistency always needed? No. DNS, PNUTS don't use it. But some systems need it.

1. The Problem
   - Main problem: messages can arrive at replicas in different orders resulting in inconsistent state.
     - See slides: C1 and C2 issue two writes, arrive in different orders
   - Note that the network causes this problem. Network will be problematic in general.

2. Replicated State Machines (RSMs)
   - RSMs ensure that each replica ends up with same final state by:
     - starting with the same initial state on each server
     - providing each replica with the same input operations, in the same order
     - ensuring that all operations are deterministic (e.g., no randomness, no reading of current time, etc.)
   - Assumption: failures are independent
     - Not always true!

3. Primary/Backup Model
   - [Clients] --> Coordinator --> Primary server --> Backup server
   - Primary does important stuff
     - Ensures that it sends all updates to the backup before ACKing the coordinator.
     - Chooses an ordering for all operations, so that the primary and backup agree.
     - Decides all non-deterministic values (e.g., random(), time())
   - What if primary fails?
     - Idea 1: Coordinator knows about both primary and backup, and decides which to use
       - Won't work: split brain syndrome. Multiple coordinators come to independent, and different, conclusions about who is primary when there are network partitions (see slides)
     - Idea 2: Have human decide when to switch from primary to backup
       - Not unreasonable for small webservices

4. View Servers
   - Add view server to primary/backup model
   - Basic functionality:
     1. The view server keeps a table that maintains a sequence of "view". Each view contains the view number, the primary server, and the backup server.
     2. The VS alerts each server as to whether it's the primary or the backup.
     3. Upon receiving any updates, the primary will receive an ACK from the backup before responding to the VS (just as before).
     4. Coordinators make requests to the VS asking who is primary. Coordinators then contact the primary.
   - To discover failures: replicas ping to the VS. If the VS misses N pings in a row, it deems a server to be dead.
   - Basic failure (actual worker crash):
     1. Primary fails; pings cease
     2. VS lets S2 know it's primary, and it handles any client requests.
        - Before S2 knowing it's the primary, it will simply reject requests that come directly from coordinators
     3. VS will eventually -- hopefully quickly -- recruit a new idle server to act as backup.

5. View Servers in the face of network partitions
   - We have a few rules in place
     1. Primary must wait for backup to accept each request.
     2. Non-primary must reject direct coordinator requests (that's what happened in the earlier failure, in the interim between the failure and S2 hearing that it was primary).
   - Add two more:
     3. Primary must reject forwarded requests (i.e., it won't accept an update from the backup).
     4. Primary in view i must have been primary or backup in view i-1.
   - Now consider S1 being partitioned from the VS (see slides)
     - Before S2 hears about View #2:
       - S1 can process operations from coordinators, S2 will accept forwarded requests.
       - S2 will reject operations from coordinators who have heard about view #2.
     - After S2 hears about View #2:
       - If S1 receives coordinator requests, it will forward. S2 will reject (not ACK), so S1 can no longer act as primary.
       - S1 will send error to coordinator, coordinator will ask VS for new view, learn about view #2, and coordinator will re-send to S2.
   - True moment of switch-over: when S2 hears about View #2

6. Recruiting new backups
   - When primary fails and backup is promoted, VS will eventually --

hopefully quickly -- find a new (idle) server to be backup
   - New backup copies state over from primary and is then ready to go
   - If primary fails during that copy?
     - Don't promote backup to primary; has incomplete state
     - Keep failed primary as primary; perhaps the failure is a network issue and it'll come back up
   - Moral of the story: having multiple backups is not a bad idea

7. Dealing with centralization
   - As described, view server is a central point of failure, which seems terrible. Also seems like it could be the bottleneck of the system.
   - Easy to distribute the view server: view servers 1 through N, each responsible for a different partition of replica sets.
   - Distribution != replication, though.
   - Can we replicate the view server in the same way? Nope; we'd need a view server for the view servers.
   - We need a mechanism for "distributed consensus"
     - RAFT: see in recitation tomorrow
     - Paxos: the original distributed consensus mechanism

Minor note: In many texts, you will see the word "client" used when describing RSMs where I used "coordinator". I am using "coordinator" because we're coming from the world of 2PC: clients (users) send request to the coordinators, and those coordinators deal with the internals of the system (contacting appropriate servers, etc.). I want you to understand how this whole picture fits together.

But a lot of times we use "client" as a generic term in the client/server setup. So in reading about RSMs, when you see client, it's just the part of the system sending requests to the servers for data. In the context of 2PC, the client in an RSM is the coordinator from 2PC.