• Distributed transactions
  • Availability
  • Replicated State Machines
goal: build reliable systems from unreliable components
the abstraction that makes that easier is transactions, which provide atomicity and isolation, while not hindering performance

atomicity

isolation

shadow copies (simple, poor performance) or logs (better performance, a bit more complex)
two-phase locking

we also want transaction-based systems to be distributed — to run across multiple machines — and to remain available even through failures
$C_1$ \text{ write}_1(X) \\

$C_2$ \text{ write}_2(X) \\

$S_1$ (replica of $S_1$)
problem: replica servers can become inconsistent
**attempt:** coordinators communicate with primary servers, who communicate with backup servers.
if primary fails, $c$ knows about $S_2$, and switches

**attempt:** coordinators communicate with primary servers, who communicate with backup servers
if primary fails, $c$ knows about $S_2$, and switches

**attempt:** coordinators communicate with primary servers, who communicate with backup servers
multiple coordinators + the network = problems

attempt: coordinators communicate with primary servers, who communicate with backup servers
multiple coordinators + the network = problems

attempt: coordinators communicate with primary servers, who communicate with backup servers
multiple coordinators + the network = problems

\[ C_1 \quad \text{network partition} \quad C_2 \]

\[ S_1 \quad \text{(primary)} \quad S_2 \quad \text{(backup, but primary for } C_2) \]

\[ C_1 \text{ and } C_2 \text{ are using different primaries; } S_1 \text{ and } S_2 \text{ are no longer consistent} \]

\textbf{attempt:} coordinators communicate with primary servers, who communicate with backup servers
use a **view server**, which determines which replica is the primary
use a **view server**, which determines which replica is the primary
use a **view server**, which determines which replica is the primary
use a **view server**, which determines which replica is the primary.
use a **view server**, which determines which replica is the primary
use a **view server**, which determines which replica is the primary
use a **view server**, which determines which replica is the primary
handling primary failure

lack of pings indicates to **VS** that **S₁** is down

1: **S₁**, **S₂**

**S₂** (backup)
handling primary failure

1: S1, S2
2: S2, --

VS

primary

S2

(dead)

(primary)
handling primary failure

C -> primary? S2
1: S1, S2
2: S2, --

VS

S2 (primary)

(dead)
handling primary failure

1: S1, S2
2: S2, --
before $S_2$ knows it’s primary, it will reject any requests from clients
(and if clients had contacted $S_1$ after it failed but before it was deemed dead, they would have received no response)
handling primary failure due to partition

lack of pings indicates to VS that S1 is down

S1 (dead)

network partition

1: S1, S2

S2 (backup)
handling primary failure due to partition

VS makes $S_2$ primary

1: $S_1, S_2$
2: $S_2, --$

network partition

$S_1$ (dead)

$S_2$ (primary)
handling primary failure due to partition

problem: what happens before $S_2$ knows it’s the primary?
handling primary failure due to partition

it’s okay! $S_2$ will act as backup
(accept updates from $S_1$, reject coordinator requests)
handling primary failure due to partition

problem: what happens after $S_2$ knows it’s the primary, but $S_1$ also thinks it is?
handling primary failure due to partition

also okay! $S_1$ won’t be able to act as primary
(can’t accept client requests because it won’t get ACKs from $S_2$)
problem: what if view server fails?

go to recitation tomorrow and find out!
• Replicated state machines (RSMs) provide single-copy consistency: operations complete as if there is a single copy of the data, though internally there are replicas.

• RSMs use a primary-backup mechanism for replication. The view server ensures that only one replica acts as the primary. It can also recruit new backups after servers fail.

• To extend this model to handle view-server failures, we need a mechanism to provide distributed consensus; see tomorrow’s recitation (on RAFT).