6.033 Spring 2018
Lecture #19

• 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** → **shadow copies** (simple, poor performance) or **logs** (better performance, a bit more complex)

- **isolation** → **two-phase locking**

we also want transaction-based systems to be **distributed** — to run across multiple 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** → **shadow copies** (simple, poor performance) or **logs** (better performance, a bit more complex)

**isolation** → **two-phase locking**

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

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

$S_1$

$S_2$

(replica of $S_1$)
problem: replica servers can become inconsistent
**attempt:** coordinators communicate with primary servers, who communicate with backup servers
if primary fails, \textbf{C} switches to backup
\textcolor{red}{(C} knows how to contact backup servers)
if primary fails, C switches to backup
(C knows how to contact backup servers)

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

Network partition

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

network partition

$C_1$ and $C_2$ are using different primaries;
$S_1$ and $S_2$ are no longer consistent

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

view server keeps a table that maintains a sequence of views
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

coordinators make requests to **view server** to find out who is primary

---

C → VS

VS: S1, S2

S1 (primary) → VS

S2 (backup)
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 $S_1$ is down

1: $S_1, S_2$

$S_2$ (backup)

VS (dead)
handling primary failure

1: S1, S2
2: S2, --

C

VS

S2

primary

(primary)

(dead)
handling primary failure

S1, S2
1: S1, S2
2: S2, --

C → primary?

VS

S2

S2 (primary)

(dead)
handling primary failure

1: \( S_1, S_2 \)
2: \( S_2, \) --

\( S_2 \)

(dead)

(primary)
handling primary failure due to partition

suppose a partition keeps $S_1$ from communicating with the view server
handling primary failure due to partition

lack of pings indicates to VS that $S_1$ is down

$1: S_1, S_2$
handling primary failure due to partition

VS makes $S_2$ primary

$S_1$ (presumed dead)

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

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

S₂ will act as backup
(accept updates from S₁, reject coordinator requests)
handling primary failure due to partition

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

S₁ won’t be able to act as primary
(can’t accept client requests because it won’t get ACKs from S₂)
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).