6.033 Spring 2019

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 — and to remain available even through failures
\begin{align*}
C_1 & \quad \text{write}_1(X) & S_1 \\
C_2 & \quad \text{write}_2(X) & S_2 & \text{(replica of } S_1) \end{align*}
problem: replica servers can become inconsistent
attempt: coordinators communicate with primary servers, who communicate with backup servers
if primary fails, \( C \) switches to backup
\( (C \) knows how to contact backup servers)
if primary fails, $\mathbf{C}$ switches to backup
($\mathbf{C}$ knows how to contact backup servers)

**attempt:** coordinators communicate with primary servers, who communicate with backup servers

$S_2$ (backup) (dead)
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

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.

view server keeps a table that maintains a sequence of *views*.

view #: primary, backup
1: S1, S2
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 $S_1$ is down.

View #: primary, backup
1: $S_1$, $S_2$
handling primary failure

view #: primary, backup
1: S1, S2
2: S2, --

S2

VS

C

(dead)

(primary)
handling primary failure

view #: primary, backup
1: S1, S2
2: S2, --

S2

primary?

C VS

6.033 | spring 2019 | lacurts@mit.edu
handling primary failure

(view #: primary, backup)

1: S1, S2
2: S2, --

S2

C

VS

(dead)
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

(view #: primary, backup
1: $S_1$, $S_2$)

(presumed dead)

S1

network partition

S2

(backup)
handling primary failure due to partition

VS makes $S_2$ primary

(view #: primary, backup
1: $S_1, S_2$
2: $S_2, --$

$S_1$ (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

$S_1$ won’t be able to act as primary
(can’t accept client requests because it won’t get ACKs from $S_2$)

view #: primary, backup
1: $S_1, S_2$
2: $S_2, --$
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).