Lecture #20: Replicated State Machines

high availability + single-copy consistency
As Use of A.I. Soars, So Does the Energy and Water It Requires

Generative artificial intelligence uses massive amounts of energy for computation and data storage and millions of gallons of water to cool the equipment at data centers. Now, legislators and regulators – in the U.S. and the EU – are starting to demand accountability.

BY DAVID BERREBY • FEBRUARY 6, 2024
6.1800 in the news

SpaceX’s Starship Kicked Up a Dust Cloud, Leaving Texans With a Mess

Residents of Port Isabel said that their city was covered in grime following SpaceX’s rocket launch on Thursday. The city said there was no “immediate concern for people’s health.”

The SpaceX Starship test flight caused dust and debris to travel miles from the launch site in Boca Chica, Texas, on Tuesday. Abraham Pineda-Jacome/EPA, via Shutterstock
6.1800 in the news

Beavers shut down town's internet for 36 hours after chewing through and stealing cables to build a dam


The Global Internet Is Being Attacked by Sharks, Google Confirms

6.1800 in the news

Beavers shut down town's internet for 36 hours after chewing through and stealing cables to build a dam


how does the physical infrastructure of our systems impact the environment?

when is it harmful? can it be helpful?

The Global Internet Is Being Attacked by Sharks, Google Confirms

BY WILL OREMUS


Sharks' attraction to undersea fiber-optic cables has been well-documented over the years.
our goal is to build **reliable systems from unreliable components**. we want to build systems that serve many clients, store a lot of data, perform well, all while keeping availability high.

transactions — which provide **atomicity** and **isolation** — make it easier for us to reason about failures.

our job in lecture is to understand how a system *implements* these two abstractions. how do our systems guarantee atomicity? how do they guarantee isolation?

**atomicity**: provided by **logging**, which gives better performance than shadow copies* at the cost of some added complexity; **two-phase commit** gives us multi-site atomicity.

**isolation**: provided by **two-phase locking**.

---

* shadow copies are used in some systems.
to increase availability, let’s try replicating data on two servers

attempt 1: nothing special, just two copies of the data

<table>
<thead>
<tr>
<th>client</th>
<th>coordinator</th>
<th>A-M server</th>
<th>A-M server</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>
to increase availability, let’s try replicating data on two servers

attempt 1: nothing special, just two copies of the data
to increase availability, let’s try replicating data on two servers

attempt 1: nothing special, just two copies of the data
to increase availability, let’s try replicating data on two servers

attempt 1: nothing special, just two copies of the data
to increase availability, let’s try replicating data on two servers

attempt 1: nothing special, just two copies of the data
to increase availability, let’s try replicating data on two servers

attempt 1: nothing special, just two copies of the data
to increase availability, let’s try replicating data on two servers

attempt 1: nothing special, just two copies of the data
to increase availability, let's try replicating data on two servers

attempt 1: nothing special, just two copies of the data
to increase availability, let’s try replicating data on two servers

attempt 1: nothing special, just two copies of the data
to increase availability, let’s try replicating data on two servers

attempt 1: nothing special, just two copies of the data

client  →  coordinator  →  A-M server  →  A-M server  →  coordinator  →  client

A=20  →  A=30
to increase availability, let’s try replicating data on two servers

attempt 1: nothing special, just two copies of the data
to increase availability, let’s try replicating data on two servers

attempt 1: nothing special, just two copies of the data

A-M server

A-M server

client

coordinator

A=20

A=30
to increase availability, let's try replicating data on two servers

attempt 1: nothing special, just two copies of the data

```
<table>
<thead>
<tr>
<th>client</th>
<th>coordinator</th>
<th>A-M server</th>
<th>A-M server</th>
<th>coordinator</th>
<th>client</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td></td>
<td>A=20</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
<td>A=30</td>
<td></td>
</tr>
</tbody>
</table>
```
to increase availability, let’s try replicating data on two servers

attempt 1: nothing special, just two copies of the data
To increase availability, let's try replicating data on two servers.

Attempt 1: Nothing special, just two copies of the data.

```
client -> coordinator --> A-M server <--> A-M server --> coordinator --> client
```

- A = 20
- ok
- A = 30
- ok
- Result: A = 30
- Result: A = 20
to increase availability, let’s try replicating data on two servers

attempt 1: nothing special, just two copies of the data

result: A=20

problem: replica servers can become inconsistent
to increase availability, let’s try replicating data on two servers

attempt 1: nothing special, just two copies of the data

Mosquito Capital @MosquitoCapital · Nov 18, 2022

50) Replication. Oh no. Um. You have, say… 5 primary regions. Each region has a copy of all mission-critical data. One day, some eng realizes that some data in A is different in B. This is *apocalyptically* bad. Which region is correct? How do you decide? How do you fix it?
to increase availability, let’s try replicating data on two servers

attempt 2: make one replica the **primary** replica, and have **coordinators** in place to help manage failures
to increase availability, let’s try replicating data on two servers

attempt 2: make one replica the **primary** replica, and have **coordinators** in place to help manage failures

clients communicate only with \( c \), not with replicas
to increase availability, let’s try replicating data on two servers

attempt 2: make one replica the primary replica, and have coordinators in place to help manage failures

clients communicate only with C, not with replicas

C sends requests to primary server

S₁ (primary)

S₂ (backup)
to increase availability, let’s try replicating data on two servers

attempt 2: make one replica the **primary** replica, and have **coordinators** in place to help manage failures

clients communicate only with **C**, not with replicas

**C** sends requests to **primary** server

**primary** chooses order of operations, decides all non-deterministic values
to increase availability, let’s try replicating data on two servers

attempt 2: make one replica the primary replica, and have coordinators in place to help manage failures

clients communicate only with C, not with replicas

C sends requests to primary server

primary chooses order of operations, decides all non-deterministic values

primary ACKs coordinator only after it’s sure that backup has all updates

primary server

S_1

(backup)

S_2

(backup)
to increase availability, let’s try replicating data on two servers

attempt 2: make one replica the primary replica, and have coordinators in place to help manage failures

primary chooses order of operations, decides all non-
deterministic values

primary ACKs coordinator only after it’s sure that backup has all updates

clients communicate only with C, not with replicas

A=2θ

C sends requests to primary server

primary

C

S1 (primary)

S2 (backup)

A=2θ

primary

A=2θ

C

S1 (primary)

S2 (backup)
to increase availability, let’s try replicating data on two servers

attempt 2: make one replica the primary replica, and have coordinators in place to help manage failures

clients communicate only with C, not with replicas

C sends requests to primary server

primary chooses order of operations, decides all non-deterministic values

primary ACKs coordinator only after it’s sure that backup has all updates

A=20

C

S₁

S₂

(backup)

(backup)
to increase availability, let’s try replicating data on two servers

attempt 2: make one replica the primary replica, and have coordinators in place to help manage failures

clients communicate only with C, not with replicas

C sends requests to primary server

A=2\(\theta\)

(primary)

S_1

S_2

(backup)

primary chooses order of operations, decides all non-deterministic values

primary ACKs coordinator only after it’s sure that backup has all updates
to increase availability, let's try replicating data on two servers

attempt 2: make one replica the primary replica, and have coordinators in place to help manage failures

clients communicate only with C, not with replicas

C sends requests to primary server

S1
A=20 (primary)

S2
A=20 (backup)

primary chooses order of operations, decides all non-deterministic values

primary ACKs coordinator only after it's sure that backup has all updates
to increase availability, let’s try replicating data on two servers

attempt 2: make one replica the primary replica, and have coordinators in place to help manage failures

clients communicate only with $C$, not with replicas

$C$ sends requests to primary server

primary chooses order of operations, decides all non-deterministic values

primary ACKs coordinator only after it’s sure that backup has all updates

primary replica, and have coordinators in place to help manage failures

A=20

A=20

attempt 2: make one replica the primary replica, and have coordinators in place to help manage failures

primary chooses order of operations, decides all non-
deterministic values

primary ACKs coordinator only after it’s sure that backup has all updates

primary replica, and have coordinators in place to help manage failures

primary chooses order of operations, decides all non-
deterministic values

primary ACKs coordinator only after it’s sure that backup has all updates

primary replica, and have coordinators in place to help manage failures

primary chooses order of operations, decides all non-
deterministic values

primary ACKs coordinator only after it’s sure that backup has all updates

primary replica, and have coordinators in place to help manage failures

primary chooses order of operations, decides all non-
deterministic values

primary ACKs coordinator only after it’s sure that backup has all updates
to increase availability, let's try replicating data on two servers

attempt 2: make one replica the primary replica, and have coordinators in place to help manage failures

clients communicate only with C, not with replicas

C sends requests to primary server

primary chooses order of operations, decides all non-deterministic values

primary ACKs coordinator only after it's sure that backup has all updates
to increase availability, let’s try replicating data on two servers

attempt 2: make one replica the **primary** replica, and have **coordinators** in place to help manage failures

clients communicate only with **C**, not with replicas

**C** sends requests to **primary** server

**primary** chooses order of operations, decides all non-deterministic values

**primary** ACKs coordinator only after it’s sure that **backup** has all updates

**primary** chooses order of operations, decides all non-deterministic values

**primary** ACKs **coordinator** only after it’s sure that **backup** has all updates
to increase availability, let’s try replicating data on two servers

attempt 2: make one replica the primary replica, and have coordinators in place to help manage failures

clients communicate only with C, not with replicas

C sends requests to primary server

C

S₁

A=2₀

S₂

A=2₀

primary chooses order of operations, decides all non-deterministic values

primary ACKs coordinator only after it’s sure that backup has all updates

primary replica
to increase availability, let’s try replicating data on two servers

attempt 2: make one replica the **primary** replica, and have **coordinators** in place to help manage failures

clients communicate only with \( C \), not with replicas

\( C \) sends requests to **primary** server

**primary** chooses order of operations, decides all non-deterministic values

**primary** ACKs coordinator only after it’s sure that **backup** has all updates
to increase availability, let’s try replicating data on two servers

attempt 2: make one replica the **primary** replica, and have **coordinators** in place to help manage failures

- **primary** chooses order of operations, decides all non-deterministic values
- **primary** ACKs coordinator only after it’s sure that **backup** has all updates

clients communicate only with **C**, not with replicas

**C** sends requests to **primary** server

**primary** sends ACKs to **C**

**S_1** and **S_2** are replicas, with **S_1** as the primary and **S_2** as the backup.

**A** = 20
to increase availability, let’s try replicating data on two servers

attempt 2: make one replica the **primary** replica, and have **coordinators** in place to help manage failures

clients communicate only with **C**, not with replicas

**C** sends requests to the **primary** server

**primary** chooses order of operations, decides all non-deterministic values

**primary** ACKs **coordinator** only after it’s sure that **backup** has all updates

**all** coordinators send requests to the **primary** server, which avoids the problem we saw in our first attempt
to increase availability, let’s try replicating data on two servers

attempt 2: make one replica the **primary** replica, and have **coordinators** in place to help manage failures

- if **primary** fails, **C** switches to **backup**
  - **C** knows how to contact backup server

**clients communicate only with **C**, not with replicas**

**C** sends requests to **primary** server

- **primary** chooses order of operations, decides all non-deterministic values
  - **primary** ACKs **coordinator** only after it’s sure that **backup** has all updates

**primary** knows how to contact backup server
to increase availability, let's try replicating data on two servers

attempt 2: make one replica the **primary** replica, and have **coordinators** in place to help manage failures

If **primary** fails, C switches to **backup**

- C knows how to contact backup server

Clients communicate only with C, not with replicas

C sends requests to **primary** server

S₂

(backup)

(failed)
to increase availability, let’s try replicating data on two servers

attempt 2: make one replica the primary replica, and have coordinators in place to help manage failures

if primary fails, C switches to backup

C knows how to contact backup server

clients communicate only with C, not with replicas

C sends requests to primary server

S2 (primary)

ideally, S1 recovers at some point, or we get some other replacement machine, and we go back to having both a primary and a backup. but for the purposes of this example, we’re just concerned about correctly switching over to the backup server.
to increase availability, let’s try replicating data on two servers

attempt 2: make one replica the **primary** replica, and have **coordinators** in place to help manage failures

for a single transaction, a client would communicate with a single **coordinator**
to increase availability, let’s try replicating data on two servers

attempt 2: make one replica the primary replica, and have coordinators in place to help manage failures

for a single transaction, a client would communicate with a single coordinator
to increase availability, let’s try replicating data on two servers

attempt 2: make one replica the primary replica, and have coordinators in place to help manage failures

for a single transaction, a client would communicate with a single coordinator
to increase availability, let’s try replicating data on two servers

attempt 2: make one replica the primary replica, and have coordinators in place to help manage failures

for a single transaction, a client would communicate with a single coordinator

suppose that all machines remain up, but that there is a network partition that effectively splits this network in half
to increase availability, let’s try replicating data on two servers

attempt 2: make one replica the primary replica, and have coordinators in place to help manage failures

for a single transaction, a client would communicate with a single coordinator

a network partition means that machines on the same side of this line can communicate with each other, but not with machines on the other side

suppose that all machines remain up, but that there is a network partition that effectively splits this network in half
to increase availability, let’s try replicating data on two servers

attempt 2: make one replica the **primary** replica, and have **coordinators** in place to help manage failures

for a single transaction, a client would communicate with a single **coordinator**

suppose that all machines remain up, but that there is a **network partition** that effectively splits this network in half

a network partition means that machines on the same side of this line can communicate with each other, but not with machines on the other side
to increase availability, let’s try replicating data on two servers

attempt 2: make one replica the **primary** replica, and have **coordinators** in place to help manage failures

---

**C₁** keeps using **S₁** as primary, with no backup

---

for a single transaction, a client would communicate with a single **coordinator**

---

a network partition means that machines on the same side of this line can communicate with each other, but not with machines on the other side

---

suppose that all machines remain up, but that there is a **network partition** that effectively splits this network in half
to increase availability, let’s try replicating data on two servers

attempt 2: make one replica the **primary** replica, and have **coordinators** in place to help manage failures

- **C**<sub>1</sub> keeps using **S**<sub>1</sub> as primary, with no backup

  ![Diagram](image)

  - for a single transaction, a client would communicate with a single coordinator

  - a network partition means that machines on the same side of this line can communicate with each other, but not with machines on the other side

- suppose that all machines remain up, but that there is a **network partition** that effectively splits this network in half

  ![Diagram](image)
to increase availability, let’s try replicating data on two servers

attempt 2: make one replica the primary replica, and have coordinators in place to help manage failures

for a single transaction, a client would communicate with a single coordinator

C₁ keeps using S₁ as primary, with no backup

C₁ keeps using S₁ as primary, with no backup

S₁

a network partition means that machines on the same side of this line can communicate with each other, but not with machines on the other side

C₂ begins using S₂ as primary, with no backup

C₂ begins using S₂ as primary, with no backup

S₂

suppose that all machines remain up, but that there is a network partition that effectively splits this network in half

for a single transaction, a client would communicate with a single coordinator
to increase availability, let’s try replicating data on two servers

attempt 2: make one replica the **primary** replica, and have **coordinators** in place to help manage failures

- **C₁** keeps using **S₁** as primary, with no backup
- **C₂** begins using **S₂** as primary, with no backup

because two different replicas both think that they are the **primary** replica, our data can become **inconsistent**

for a single transaction, a client would communicate with a single **coordinator**
to increase availability, let’s try replicating data on two servers

attempt 3: use a **view server** to determine which replica is primary, in hopes that we can deal with network partitions
to increase availability, let’s try replicating data on two servers

attempt 3: use a view server to determine which replica is primary, in hopes that we can deal with network partitions

view server keeps a table that maintains a sequence of views
to increase availability, let’s try replicating data on two servers

attempt 3: use a **view server** to determine which replica is primary, in hopes that we can deal with network partitions
to increase availability, let’s try replicating data on two servers

attempt 3: use a view server to determine which replica is primary, in hopes that we can deal with network partitions

view server keeps a table that maintains a sequence of views

<table>
<thead>
<tr>
<th>view #</th>
<th>primary</th>
<th>backup</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>S1</td>
<td>S2</td>
</tr>
</tbody>
</table>

view server alerts primary/backups about their roles
to increase availability, let’s try replicating data on two servers

attempt 3: use a view server to determine which replica is primary, in hopes that we can deal with network partitions

view server alerts primary/backups about their roles

view server keeps a table that maintains a sequence of views
to increase availability, let’s try replicating data on two servers

attempt 3: use a view server to determine which replica is primary, in hopes that we can deal with network partitions

view server alerts primary/backups about their roles

view server keeps a table that maintains a sequence of views

<table>
<thead>
<tr>
<th>view #</th>
<th>primary</th>
<th>backup</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>S1</td>
<td>S2</td>
</tr>
</tbody>
</table>
to increase availability, let’s try replicating data on two servers

attempt 3: use a view server to determine which replica is primary, in hopes that we can deal with network partitions

<table>
<thead>
<tr>
<th>view #</th>
<th>primary</th>
<th>backup</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>S1</td>
<td>S2</td>
</tr>
</tbody>
</table>

view server keeps a table that maintains a sequence of views
to increase availability, let’s try replicating data on two servers

attempt 3: use a view server to determine which replica is primary, in hopes that we can deal with network partitions

coordinators make requests to view server to find out which replica is primary

view server keeps a table that maintains a sequence of views

<table>
<thead>
<tr>
<th>view #</th>
<th>primary</th>
<th>backup</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>S1</td>
<td>S2</td>
</tr>
</tbody>
</table>

(backup) S2
(backup) S1
to increase availability, let’s try replicating data on two servers

attempt 3: use a **view server** to determine which replica is primary, in hopes that we can deal with network partitions

**coordinators** make requests to **view server** to find out which replica is primary

<table>
<thead>
<tr>
<th>view #</th>
<th>primary</th>
<th>backup</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>S1</td>
<td>S2</td>
</tr>
</tbody>
</table>

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

- (primary) 
  - S₁
- (backup)  
  - S₂
to increase availability, let’s try replicating data on two servers

attempt 3: use a **view server** to determine which replica is primary, in hopes that we can deal with network partitions

coordinators make requests to **view server** to find out which replica is primary

<table>
<thead>
<tr>
<th>view #</th>
<th>primary</th>
<th>backup</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>S1</td>
<td>S2</td>
</tr>
</tbody>
</table>

**view server** keeps a table that maintains a sequence of views
to increase availability, let’s try replicating data on two servers

attempt 3: use a **view server** to determine which replica is primary, in hopes that we can deal with network partitions

- coordinators make requests to **view server** to find out which replica is **primary**

<table>
<thead>
<tr>
<th>view #</th>
<th>primary</th>
<th>backup</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>S1</td>
<td>S2</td>
</tr>
</tbody>
</table>

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

**coordinators** contact **primary** (as before)

S1

S2

(primary)
to increase availability, let’s try replicating data on two servers

attempt 3: use a view server to determine which replica is primary, in hopes that we can deal with network partitions

Coordinators make requests to view server to find out which replica is primary.

View server keeps a table that maintains a sequence of views:

<table>
<thead>
<tr>
<th>view #</th>
<th>primary</th>
<th>backup</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>S1</td>
<td>S2</td>
</tr>
</tbody>
</table>

View server contacts coordinators to determine which replica is primary (as before).
to increase availability, let’s try replicating data on two servers

attempt 3: use a **view server** to determine which replica is primary, in hopes that we can deal with network partitions

**coordinators** make requests to **view server** to find out which replica is **primary**

<table>
<thead>
<tr>
<th>view #</th>
<th>primary</th>
<th>backup</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>S1</td>
<td>S2</td>
</tr>
</tbody>
</table>

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

**coordinators** contact **primary** (as before)

**primary** sends updates to, gets ACKs from **backup** (as before)
to increase availability, let’s try replicating data on two servers

attempt 3: use a **view server** to determine which replica is primary, in hopes that we can deal with network partitions

**coordinators** make requests to **view server** to find out which replica is **primary**

**coordinators** contact **primary** (as before)

<table>
<thead>
<tr>
<th>view</th>
<th>primary</th>
<th>backup</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>S1</td>
<td>S2</td>
</tr>
</tbody>
</table>

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

**primary** sends updates to, gets ACKs from **backup** (as before)
to increase availability, let’s try replicating data on two servers

attempt 3: use a **view server** to determine which replica is primary, in hopes that we can deal with network partitions

- coordinators make requests to **view server** to find out which replica is **primary**
  - coordinators contact **primary** (as before)
  - (**primary**) sends updates to, gets ACKs from **backup** (as before)

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

<table>
<thead>
<tr>
<th>view #</th>
<th>primary</th>
<th>backup</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>S1</td>
<td>S2</td>
</tr>
</tbody>
</table>

Katrina LaCurts | lacurts@mit.edu | 6.1800 2024
to increase availability, let’s try replicating data on two servers

attempt 3: use a view server to determine which replica is primary, in hopes that we can deal with network partitions

---

**coordinators** make requests to **view server** to find out which replica is primary

---

<table>
<thead>
<tr>
<th>view #</th>
<th>primary</th>
<th>backup</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>S1</td>
<td>S2</td>
</tr>
</tbody>
</table>

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

---

**coordinators** contact **primary** (as before)

---

**primary** sends updates to, gets ACKs from **backup** (as before)
to increase availability, let’s try replicating data on two servers

attempt 3: use a **view server** to determine which replica is primary, in hopes that we can deal with network partitions

**coordinators** make requests to **view server** to find out which replica is primary

**primary** (as before)

**primary/backup** ping **view server** so that **view server** can discover failures

**primary** sends updates to, gets ACKs from **backup** (as before)

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

<table>
<thead>
<tr>
<th>view #</th>
<th>primary</th>
<th>backup</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>S1</td>
<td>S2</td>
</tr>
</tbody>
</table>
to increase availability, let’s try replicating data on two servers

attempt 3: use a **view server** to determine which replica is primary, in hopes that we can deal with network partitions

- Coordinators make requests to view server to find out which replica is primary
- Coordinators contact primary (as before)
- Primary/backup ping view server so that view server can discover failures
- Primary sends updates to, gets ACKs from backup (as before)

- View server keeps a table that maintains a sequence of views
- View # | primary | backup
- 1     S1        S2

Katrina LaCurts | lacurts@mit.edu | 6.1800 2024
to increase availability, let’s try replicating data on two servers

attempt 3: use a **view server** to determine which replica is primary, in hopes that we can deal with network partitions

**coordinators** make requests to **view server** to find out which replica is **primary**

**coordinators** contact **(primary)** (as before)

**S1**

**S2**

**VS**

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

view # | primary | backup
1     S1    S2

**primary/backup ping view server** so that **view server** can discover failures

**primary** sends updates to, gets ACKs from **backup** (as before)

**question**: in our set-up, there is **one view server** for this entire system, whereas there can be multiple **coordinators**. why might having a single view server help us when failures (such as the examples you’ve already seen) occur?
to increase availability, let’s try replicating data on two servers

attempt 3: use a **view server** to determine which replica is primary, in hopes that we can deal with network partitions

what happens if the **primary** replica fails?
to increase availability, let’s try replicating data on two servers

attempt 3: use a **view server** to determine which replica is primary, in hopes that we can deal with network partitions

<table>
<thead>
<tr>
<th>view #</th>
<th>primary</th>
<th>backup</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>S1</td>
<td>S2</td>
</tr>
</tbody>
</table>

what happens if the **primary** replica fails?
to increase availability, let’s try replicating data on two servers

attempt 3: use a **view server** to determine which replica is primary, in hopes that we can deal with network partitions

<table>
<thead>
<tr>
<th>view #</th>
<th>primary</th>
<th>backup</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>S1</td>
<td>S2</td>
</tr>
</tbody>
</table>

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

what happens if the **primary** replica fails?
to increase availability, let’s try replicating data on two servers

attempt 3: use a **view server** to determine which replica is primary, in hopes that we can deal with network partitions

<table>
<thead>
<tr>
<th>view #</th>
<th>primary</th>
<th>backup</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>S1</td>
<td>S2</td>
</tr>
<tr>
<td>2</td>
<td>S2</td>
<td></td>
</tr>
</tbody>
</table>

lack of pings indicates to VS that S₁ is down

(failed)

what happens if the **primary** replica fails?
to increase availability, let’s try replicating data on two servers

attempt 3: use a **view server** to determine which replica is primary, in hopes that we can deal with network partitions

lack of pings indicates to **VS** that S₁ is down (failed)

what happens if the primary replica fails?
to increase availability, let’s try replicating data on two servers

attempt 3: use a **view server** to determine which replica is primary, in hopes that we can deal with network partitions

if C communicates with S₁, C won’t get a response; when C next asks VS who the primary is, VS will respond with S₂

lack of pings indicates to VS that S₁ is down

what happens if the **primary** replica fails?
to increase availability, let's try replicating data on two servers

attempt 3: use a **view server** to determine which replica is primary, in hopes that we can deal with network partitions

if C communicates with S₁, C won’t get a response; when C next asks VS who the primary is, VS will respond with S₂

lack of pings indicates to VS that S₁ is down

notice there is no longer a backup. once again, we’d hope to eventually bring S₁ back online, or find a new machine to act as a backup. but in this example, we’re only interested in safely making S₂ the new primary.

what happens if the **primary** replica fails?
to increase availability, let’s try replicating data on two servers

attempt 3: use a **view server** to determine which replica is primary, in hopes that we can deal with network partitions
to increase availability, let’s try replicating data on two servers

attempt 3: use a view server to determine which replica is primary, in hopes that we can deal with network partitions

what happens if a network partition prevents $S_1$ from communicating with $VS$?
to increase availability, let’s try replicating data on two servers
attempt 3: use a **view server** to determine which replica is primary, in hopes that we can deal with network partitions

what happens if a network partition prevents $S_1$ from communicating with $VS$?
to increase availability, let’s try replicating data on two servers

attempt 3: use a **view server** to determine which replica is primary, in hopes that we can deal with network partitions

what happens if a network partition prevents **S₁** from communicating with **VS**?

in a sense, this is the worst possible partition: **VS** is going to presume **S₁** has failed (and so switch to using **S₂** as a backup), while **S₁** can still communicate with everyone except **VS**
to increase availability, let’s try replicating data on two servers

attempt 3: use a **view server** to determine which replica is primary, in hopes that we can deal with network partitions

<table>
<thead>
<tr>
<th>view #</th>
<th>primary</th>
<th>backup</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>S1</td>
<td>S2</td>
</tr>
</tbody>
</table>

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

what happens if a network partition prevents **S1** from communicating with **VS**?

in a sense, this is the worst possible partition: **VS** is going to presume **S1** has failed (and so switch to using **S2** as a backup), while **S1** can still communicate with everyone except **VS**
to increase availability, let’s try replicating data on two servers

attempt 3: use a **view server** to determine which replica is primary, in hopes that we can deal with network partitions

---

What happens if a network partition prevents $S_1$ from communicating with $VS$?

In a sense, this is the worst possible partition: $VS$ is going to presume $S_1$ has failed (and so switch to using $S_2$ as a backup), while $S_1$ can still communicate with everyone except $VS$
to increase availability, let’s try replicating data on two servers

attempt 3: use a view server to determine which replica is primary, in hopes that we can deal with network partitions

What happens if a network partition prevents $S_1$ from communicating with VS?

In a sense, this is the worst possible partition: VS is going to presume $S_1$ has failed (and so switch to using $S_2$ as a backup), while $S_1$ can still communicate with everyone except VS.
to increase availability, let’s try replicating data on two servers

attempt 3: use a view server to determine which replica is primary, in hopes that we can deal with network partitions

what happens if a network partition prevents $S_1$ from communicating with VS?

in a sense, this is the worst possible partition: VS is going to presume $S_1$ has failed (and so switch to using $S_2$ as a backup), while $S_1$ can still communicate with everyone except VS
to increase availability, let’s try replicating data on two servers

attempt 3: use a view server to determine which replica is primary, in hopes that we can deal with network partitions

what happens if a network partition prevents $S_1$ from communicating with $VS$?

in a sense, this is the worst possible partition: $VS$ is going to presume $S_1$ has failed (and so switch to using $S_2$ as a backup), while $S_1$ can still communicate with everyone except $VS$
to increase availability, let’s try replicating data on two servers

attempt 3: use a view server to determine which replica is primary, in hopes that we can deal with network partitions

what happens if a network partition prevents $S_1$ from communicating with VS?

in a sense, this is the worst possible partition: VS is going to presume $S_1$ has failed (and so switch to using $S_2$ as a backup), while $S_1$ can still communicate with everyone except VS

at this stage, VS thinks $S_2$ is primary; $S_2$ and $S_1$ think $S_1$ is primary

if $S_1$ receives any requests from C, it will behave as primary with $S_2$ as backup
to increase availability, let’s try replicating data on two servers

at attempt 3: use a **view server** to determine which replica is primary, in hopes that we can deal with network partitions

what happens if a network partition prevents $S_1$ from communicating with $VS$?

- In a sense, this is the worst possible partition: $VS$ is going to presume $S_1$ has failed (and so switch to using $S_2$ as a backup), while $S_1$ can still communicate with everyone **except $VS$**

at this stage, $VS$ thinks $S_2$ is primary; $S_2$ and $S_1$ think $S_1$ is primary

if $S_1$ receives any requests from $C$, it will behave as primary with $S_2$ as backup

if $S_2$ receives any requests from $C$, it will reject them; it believes that it is the backup (and so does not communicate directly with $C$)

view # | primary | backup  
--- | --- | ---  
1 | $S_1$ | $S_2$  
2 | $S_2$ | (backup)

(presumed failed)
to increase availability, let’s try replicating data on two servers

attempt 3: use a view server to determine which replica is primary, in hopes that we can deal with network partitions

what happens if a network partition prevents $S_1$ from communicating with $VS$?

in a sense, this is the worst possible partition: $VS$ is going to presume $S_1$ has failed (and so switch to using $S_2$ as a backup), while $S_1$ can still communicate with everyone except $VS$.
to increase availability, let’s try replicating data on two servers

attempt 3: use a view server to determine which replica is primary, in hopes that we can deal with network partitions

what happens if a network partition prevents $S_1$ from communicating with VS?

in a sense, this is the worst possible partition: VS is going to presume $S_1$ has failed (and so switch to using $S_2$ as a backup), while $S_1$ can still communicate with everyone except VS
to increase availability, let’s try replicating data on two servers

attempt 3: use a **view server** to determine which replica is primary, in hopes that we can deal with network partitions

what happens if a network partition prevents \( S_1 \) from communicating with \( VS \)?

in a sense, this is the worst possible partition: \( VS \) is going to presume \( S_1 \) has failed (and so switch to using \( S_2 \) as a backup), while \( S_1 \) can still communicate with everyone except \( VS \)
to increase availability, let’s try replicating data on two servers

attempt 3: use a view server to determine which replica is primary, in hopes that we can deal with network partitions

what happens if a network partition prevents $S_1$ from communicating with $VS$?

in a sense, this is the worst possible partition: $VS$ is going to presume $S_1$ has failed (and so switch to using $S_2$ as a backup), while $S_1$ can still communicate with everyone except $VS$
to increase availability, let’s try replicating data on two servers

attempt 3: use a view server to determine which replica is primary, in hopes that we can deal with network partitions

what happens if a network partition prevents $S_1$ from communicating with $VS$?

in a sense, this is the worst possible partition: $VS$ is going to presume $S_1$ has failed (and so switch to using $S_2$ as a backup), while $S_1$ can still communicate with everyone except $VS$
to increase availability, let’s try replicating data on two servers

attempt 3: use a view server to determine which replica is primary, in hopes that we can deal with network partitions

what happens if a network partition prevents $S_1$ from communicating with VS?

in a sense, this is the worst possible partition: VS is going to presume $S_1$ has failed (and so switch to using $S_2$ as a backup), while $S_1$ can still communicate with everyone except VS

at this stage, VS and $S_2$ think $S_2$ is primary; $S_1$ thinks $S_1$ is primary

if $S_1$ receives any requests from C, it won’t be able to get an ACK from $S_2$, and so will reject

if $S_2$ receives any requests from C, it will respond as the primary (in line with what VS expects)
to increase availability, let’s try replicating data on two servers

attempt 3: use a view server to determine which replica is primary, in hopes that we can deal with network partitions

important rule: if a machine is primary in view \( n \), it must have been primary or backup in view \( n-1 \) (with the exception of view 1, when we’re just starting)

what happens if a network partition prevents \( S_1 \) from communicating with \( VS \)?

in a sense, this is the worst possible partition: \( VS \) is going to presume \( S_1 \) has failed (and so switch to using \( S_2 \) as a backup), while \( S_1 \) can still communicate with everyone except \( VS \)

once \( S_1 \) can communicate with \( VS \) again, \( VS \) will respond notifying it that it is not in the current view

at this stage, \( VS \) and \( S_2 \) think \( S_2 \) is primary; \( S_1 \) thinks \( S_1 \) is primary

if \( S_1 \) receives any requests from \( C \), it won’t be able to get an ACK from \( S_2 \), and so will reject

if \( S_2 \) receives any requests from \( C \), it will respond as the primary (in line with what \( VS \) expects)

what happens if a network partition prevents \( S_1 \) from communicating with \( VS \)?

in a sense, this is the worst possible partition: \( VS \) is going to presume \( S_1 \) has failed (and so switch to using \( S_2 \) as a backup), while \( S_1 \) can still communicate with everyone except \( VS \)
to increase availability, let's try replicating data on two servers

attempt 3: use a view server to determine which replica is primary, in hopes that we can deal with network partitions

**important rule:** if a machine is primary in view $n$, it must have been primary or backup in view $n-1$ (with the exception of view 1, when we're just starting)

once $S_1$ can communicate with $VS$ again, $VS$ will respond notifying it that it is *not* in the current view

at this stage, $VS$ and $S_2$ think $S_2$ is primary; $S_1$ thinks $S_1$ is primary

if $S_1$ receives any requests from $C$, it won't be able to get an ACK from $S_2$, and so will reject

if $S_2$ receives any requests from $C$, it will respond as the primary (in line with what $VS$ expects)

what happens if a network partition prevents $S_1$ from communicating with $VS$?

in a sense, this is the worst possible partition: $VS$ is going to presume $S_1$ has failed (and so switch to using $S_2$ as a backup), while $S_1$ can still communicate with everyone except $VS$
to increase availability, let’s try replicating data on two servers

attempt 3: use a view server to determine which replica is primary, in hopes that we can deal with network partitions
to increase availability, let’s try replicating data on two servers

to increase availability, let’s try replicating data on two servers

attempt 3: use a **view server** to determine which replica is primary, in hopes that we can deal with network partitions

what happens if **VS** fails?
to increase availability, let’s try replicating data on two servers

attempt 3: use a view server to determine which replica is primary, in hopes that we can deal with network partitions

what happens if VS fails?

find out in Tuesday's recitation
Replicated state machines

Coordinators make requests to view server to find out which replica is primary.

Coordinators contact primary (as before).

Primary/backup ping view server so that view server can discover failures.

Primary sends updates to, gets ACKs from backup (as before).

Primary must get an ACK from its backups before completing the update.

Backups will reject any requests that they get directly from coordinators; primary will reject any update that comes from a backup.

(view server) keeps a table that maintains a sequence of views.

<table>
<thead>
<tr>
<th>view #</th>
<th>primary</th>
<th>backup</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>S1</td>
<td>S2</td>
</tr>
</tbody>
</table>

If a machine is primary in view n, it must have been primary or backup in view n-1 (with the exception of view 1, when we're just starting).

(both of these events can happen in the case of certain types of failures)
our goal is to build **reliable systems from unreliable components**. we want to build systems that serve many clients, store a lot of data, perform well, all while keeping availability high

transactions — which provide **atomicity** and **isolation** — make it easier for us to reason about failures

our job in lecture is to understand how a system *implements* these two abstractions. how do our systems guarantee atomicity? how do they guarantee isolation?

**atomicity**: provided by **logging**, which gives better performance than shadow copies* at the cost of some added complexity; **two-phase commit** gives us multi-site atomicity

**isolation**: provided by **two-phase locking**
replicated state machines (RSMs) provide single-copy consistency: externally, it appears 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, and can recruit new backups if servers fail.

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