Scientists spy on Mount Etna with fiber-optic cables

Researchers detect volcanic activity by watching how light moves through a cable. (MATT SIMON, WIRED.COM - 4/10/2022, 7:00 AM)

Fiber optics work by transporting signals from point A to point B as pulses of light. But if the cable is disturbed by, say, an earthquake, a tiny amount of that light gets bounced back to the source. To measure this, scientists use an “interrogator,” which fires a laser through the fibers and analyzes what comes back. Because researchers know the speed of light, they can determine disturbances at various lengths along the cable: something happening 60 feet away will bounce back light that takes slightly longer to get to the interrogator than something happening at 50 feet.
6.033 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
AUG 15, 2014 • 3:23 PM

Sharks' attraction to undersea fiber-optic cables has been well-documented over the years.

Lecture #20: Replicated State Machines

high availability + single-copy consistency
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
to increase availability, let’s try replicating data on two servers

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

result: $A=30$

result: $A=20$

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

attempt 2: make one replica the **primary** replica, and have a coordinator 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

---

**Diagram:**

- Clients communicate only with **C**, not with replicas.
- **C** sends requests to **primary** server.
- **primary** sends requests to **S1** and **S2**.
- **S1** is the **primary** replica, **S2** is the **backup**.
- **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 a coordinator in place to help manage failures

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

ideally, $S_1$ 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 a coordinator 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 a coordinator in place to help manage failures

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

because two different replicas both think that they are the primary replica, our data can become inconsistent
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 (can we? let’s find out)

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 (can we? let’s find out)

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>

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 (can we? let’s find out)

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>

S1 (primary) sends updates to, gets ACKs from S2 (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 (can we? let’s find out)

do 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>

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).
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 (can we? let’s find out)

<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 (can we? let’s find out)

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 (can we? let’s find out)

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 (can we? let’s find out)

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

VS makes $S_2$ primary

at this stage, VS thinks $S_2$ is primary; $S_2$ and $S_1$ think $S_1$ is 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 (can we? let’s find out)

<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>

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

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 (can we? let’s find out)

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 (can we? let’s find out)

at this stage, VS and S₂ think S₂ is primary; S₁ thinks S₁ is primary

if S₁ receives any requests from C, it won’t be able to get an ACK from S₂, and so will reject

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 (can we? let's find out)

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

at this stage, VS and S₂ think S₂ is primary; S₁ thinks S₁ is primary

if S₁ receives any requests from C, it won't be able to get an ACK from S₂, and so will reject

if S₂ 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 (can we? let's find out)

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

<table>
<thead>
<tr>
<th>view #</th>
<th>primary</th>
<th>backup</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>( S_1 )</td>
<td>( S_2 )</td>
</tr>
<tr>
<td>2</td>
<td>( S_2 )</td>
<td>(primary)</td>
</tr>
</tbody>
</table>

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 (can we? let’s find out)

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.

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

View # | Primary | Backup
1     S1        S2

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

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

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**

* shadow copies are used in some systems

replicated state machines give us single-copy consistency even with replicated data.
**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.