Once these software updates have been given the OK on Earth, they are then compressed and separated into pieces for their journey through the Deep Space Network, the communication network of giant radio antennas dotted around the Earth that helps NASA transmit information to and from planetary spacecraft.

“The pieces of the software load are sent from the Jet Propulsion Laboratory, via the DSN, to one of the satellites currently orbiting Mars [and] are then relayed down to the Perseverance rover on the surface of Mars, where they are reassembled into the consolidated software package,” explains Lavery.

Once the software update has safely landed in Perseverance's hands, it is finally passed along to Ingenuity via the Helicopter Base Station, which Lavery says is a “dedicated controller in the rover which collects, stores, and configures data communications between the rover and the helicopter.”

The Mark 6 VLBI Data System (Whitney et al. 2013) is a packet recording system used to store the R2DBE output streams to hard drives. The recorder captures the two 8 Gbps streams from the R2DBE using commercial 10 Gbps Ethernet network interface cards. It strips the internet packet headers and stores the payload containing VDIF data frames. **For sustained recording at 16 Gbps, each Mark 6 recorder writes the data in time slices across 32 hard drives with a round-robin algorithm. The disks are mounted in groups of eight in four removable modules for ease of handling and shipping. For the EHT, four such recorders are configured in parallel to achieve an aggregate data capture rate of 64 Gbps.**

Due to the remote nature of many of its sites and the large data volume recorded, the EHT lacks real-time verification of fringes. Shipping data from remote stations to the central correlator and processing takes several days at minimum, and many months in the worst case (from the South Pole; see Appendix A.11). If a key system fails, it is possible for a site to take data that never result in fringes, making careful testing and retesting of subsystems throughout the observation absolutely crucial. Data from brief observations (10–30 s) can be transferred from sites with fast internet connections for near real-time fringe verification. While possible, this requires robust data transfer connections.
Results and logs from the setup and verification are centrally archived and available to the correlator centers for the interpretation of station issues and for fault diagnosis when data quality issues emerge.
The failure of a hard drive in one of the JCMT modules caused one-sixteenth of the data in the low band to be lost. The lost data affects all scans on the module approximately equally, as packets are scattered onto all hard drives at record time. This issue required no special handling because DiFX automatically adjusts data weights based on the amount of data in each AP.
6.033 Spring 2021
Lecture #18: Distributed Transactions
getting atomicity across machines
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.

isolation: provided by two-phase locking.

* shadow copies are used in some systems.

Katrina LaCurts | lacurts@mit.edu | 6.033 2021
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.

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

* shadow copies are used in some systems
transactions across multiple machines (no failures yet)

\text{transfer}(A, B, \text{amount})
transactions across multiple machines (no failures yet)

\text{transfer}(A, B, \text{amount})
transactions across multiple machines (no failures yet)

\[ \text{transfer}(A, B, \text{amount}) \]
transactions across multiple machines (no failures yet)

\[
\text{transfer}(A, B, \text{amount})
\]

| client | coordinator | A-M server |
transactions across multiple machines (no failures yet)

\[\text{transfer}(A, B, \text{amount})\]

- **client**
- **coordinator**
- **A-M server**
transactions across multiple machines (no failures yet)

\[\text{transfer}(A, B, \text{amount})\]
transactions across multiple machines (no failures yet)
transitions across multiple machines (no failures yet)

\[ \text{transfer}(A, B, \text{amount}) \]
transactions across multiple machines (no failures yet)
transactions across multiple machines (no failures yet)

client          coordinator          A-M server

begin
ok
A-amount
ok
transactions across multiple machines (no failures yet)

```plaintext
transfer(A, B, amount)```

```
client     coordinator     A-M server

begin →

ok ←

A-amount →

ok ←

B+amount →

ok ←
```
transactions across multiple machines (no failures yet)

```plaintext
transfer(A, B, amount)
```

```
client          coordinator          A-M server

begin

ok

A-amount

ok

B+amount

ok

commit

ok
```
transactions across multiple machines (no failures yet)

transfer(A, Z, amount)
transactions across multiple machines (no failures yet)

transfer(A, Z, amount)

client | coordinator | A-M server | N-Z server

begin
transactions across multiple machines (no failures yet)

\[ \text{transfer}(A, Z, \text{amount}) \]

- client
- coordinator
- A-M server
- N-Z server

begin
transactions across multiple machines (no failures yet)

transfer(A, Z, amount)

client          coordinator          A-M server          N-Z server

begin

---
transactions across multiple machines (no failures yet)

\[
\text{transfer}(A, Z, \text{amount})
\]

client \hspace{1cm} coordinator \hspace{1cm} A-M server \hspace{1cm} N-Z server

begin \hspace{2cm} ok
transactions across multiple machines (no failures yet)

\[
\text{transfer}(A, Z, \text{amount})
\]
transactions across multiple machines (no failures yet)

\[\text{transfer}(A, Z, \text{amount})\]

\begin{tabular}{|c|c|c|c|}
\hline
client & coordinator & A-M server & N-Z server \\
\hline
\hline
\text{begin} & & & \\
\hline
\text{ok} & & & \\
\hline
\text{A-amount} & & & \\
\hline
\text{ok} & & & \\
\hline
\end{tabular}
transactions across multiple machines (no failures yet)

$$\text{transfer}(A, Z, \text{amount})$$

client | coordinator | A-M server | N-Z server
---|---|---|---
begin
ok
A-amount
ok
Z+amount

(no failures yet)
transactions across multiple machines (no failures yet)

\[ \text{transfer}(A, Z, \text{amount}) \]

<table>
<thead>
<tr>
<th>client</th>
<th>coordinator</th>
<th>A-M server</th>
<th>N-Z server</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>begin</td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td>ok</td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td>A-amount</td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td>ok</td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td>Z+amount</td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td>ok</td>
<td></td>
<td></td>
</tr>
</tbody>
</table>
transactions across multiple machines (no failures yet)

transfer(A, Z, amount)

client  coordinator  A-M server  N-Z server

<table>
<thead>
<tr>
<th>begin</th>
<th>ok</th>
<th>ok</th>
<th>ok</th>
<th>ok</th>
</tr>
</thead>
<tbody>
<tr>
<td>A-amount</td>
<td>Z+amount</td>
<td>commit</td>
<td></td>
<td></td>
</tr>
</tbody>
</table>
transactions across multiple machines (no failures yet)

transfer(A, Z, amount)

client          coordinator          A-M server          N-Z server

begin  →  ok     ←  ok

A-amount  →  ok     ←  ok

Z+amount  →  ok

commit  →  ok
transactions across multiple machines (no failures yet)

transfer(A, Z, amount)

client | coordinator | A-M server | N-Z server

begin →

ok ←

A-amount →

ok ←

Z+amount →

ok ←

commit →
transactions across multiple machines (no failures yet)

\[
\text{transfer}(A, Z, \text{amount})
\]

<table>
<thead>
<tr>
<th>client</th>
<th>coordinator</th>
<th>A-M server</th>
<th>N-Z server</th>
</tr>
</thead>
<tbody>
<tr>
<td>begin</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>ok</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>A-amount</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>ok</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Z+amount</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>ok</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>commit</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>ok</td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>
transactions across multiple machines (now with failures)

transfer(A, Z, amount)

client | coordinator | A-M server | N-Z server

<table>
<thead>
<tr>
<th>begin</th>
<th>ok</th>
<th>ok</th>
<th>ok</th>
<th>ok</th>
</tr>
</thead>
<tbody>
<tr>
<td>A-amount</td>
<td>ok</td>
<td>Z+amount</td>
<td>ok</td>
<td>commit</td>
</tr>
</tbody>
</table>

X
transactions across multiple machines (now with failures)

transfer(A, Z, amount)

<table>
<thead>
<tr>
<th>client</th>
<th>coordinator</th>
<th>A-M server</th>
<th>N-Z server</th>
</tr>
</thead>
<tbody>
<tr>
<td>begin</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>ok</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>A-amount</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>ok</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Z+amount</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>ok</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>commit</td>
<td></td>
<td></td>
<td>X</td>
</tr>
</tbody>
</table>

problem: one server committed, the other did not
(we'd have a similar problem if the N-Z server crashed)
transactions across multiple machines (now with failures)

transfer(A, Z, amount)

<table>
<thead>
<tr>
<th>client</th>
<th>coordinator</th>
<th>A-M server</th>
<th>N-Z server</th>
</tr>
</thead>
<tbody>
<tr>
<td>begin</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td>ok</td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td>A-amount</td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td>ok</td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td>Z+amount</td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td>ok</td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td>commit</td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

goal: develop a protocol that can provide multi-site atomicity in the face of all sorts of failures (message loss, message reordering, worker failure, coordinator failure)

problem: one server committed, the other did not (we'd have a similar problem if the N-Z server crashed)
transactions across multiple machines (now with failures)

goal: develop a protocol that can provide multi-site atomicity in the face of all sorts of failures (message loss, message reordering, worker failure, coordinator failure)

message failures solved with reliable transport protocol (sequence numbers + ACKs)

problem: one server committed, the other did not (we'd have a similar problem if the N-Z server crashed)
two-phase commit: nodes agree that they’re ready to commit before committing

client

<table>
<thead>
<tr>
<th>coordinator</th>
</tr>
</thead>
<tbody>
<tr>
<td>A-M server</td>
</tr>
<tr>
<td>N-Z server</td>
</tr>
</tbody>
</table>

assume all of the parts of the transaction pre-commit are happening here

ok
**two-phase commit**: nodes agree that they’re ready to commit before committing.

- Assume all of the parts of the transaction pre-commit are happening here.
- At this point, the client is ready to commit the transaction.
two-phase commit: nodes agree that they’re ready to commit before committing

client → coordinator → A-M server → N-Z server

assume all of the parts of the transaction pre-commit are happening here

commit → ok

at this point, the client is ready to commit the transaction
**two-phase commit**: nodes agree that they’re ready to commit before committing

<table>
<thead>
<tr>
<th>client</th>
<th>coordinator</th>
<th>A-M server</th>
<th>N-Z server</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>assume all of the parts of the transaction pre-commit are happening here</td>
<td>prepare</td>
<td>prepare</td>
</tr>
<tr>
<td>ok</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>commit</td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

at this point, the client is ready to commit the transaction
**two-phase commit**: nodes agree that they’re ready to commit before committing

- **client**
  - commit
  - assume all of the parts of the transaction pre-commit are happening here
- **coordinator**
  - ok
- **A-M server**
  - prepare
- **N-Z server**
  - prepare

At this point, the client is ready to commit the transaction.
two-phase commit: nodes agree that they’re ready to commit before committing

Client | Coordinator | A-M server | N-Z server
--- | --- | --- | ---
commit | prepare | prepare | ok
prepare | ok | at this point, the client is ready to commit the transaction
**two-phase commit**: nodes agree that they’re ready to commit before committing

- **Client**
  - Assume all of the parts of the transaction pre-commit are happening here.
  - Commit
  - Ok

- **Coordinator**
  - Ok

- **A-M Server**
  - Prepare
  - Prepare
  - Commit
  - Ok

- **N-Z Server**
  - Commit
  - Ok

At this point, the client is ready to commit the transaction.
two-phase commit: nodes agree that they’re ready to commit before committing

Client

Assume all of the parts of the transaction pre-commit are happening here

- ok
- commit
- ok

Coordinator

Prepare

A-M server

Commit

Prepare

N-Z server

Commit

At this point, the client is ready to commit the transaction
two-phase commit: nodes agree that they’re ready to commit before committing

To understand why this protocol provides atomicity — and specifically why we need two phases — we need to examine how it behaves under a variety of different types of failures.
two-phase commit: nodes agree that they’re ready to commit before committing.
**two-phase commit**: nodes agree that they’re ready to commit before committing

- **client**
- **coordinator**
- **A-M server**
- **N-Z server**

---

- **worker failure before prepare phase**

---
**two-phase commit**: nodes agree that they’re ready to commit before committing

- **client**
- **coordinator**
- **A-M server**
- **N-Z server**

*worker failure before prepare phase*
two-phase commit: nodes agree that they’re ready to commit before committing

- worker failure before prepare phase

Diagram:

- Client sends an 'ok' request to the coordinator.
- The coordinator sends a prepare request to the A-M server.
- The coordinator sends an 'ok' request to the client.
- The coordinator sends an 'abort' request to the A-M server.
- The coordinator sends an 'ok' request to the N-Z server.
- The coordinator sends an 'abort' request to the client.

Diagram notes:
- A-M server sends an 'ok' request to the coordinator.
- N-Z server sends an 'ok' request to the coordinator.
two-phase commit: nodes agree that they’re ready to commit before committing

worker failure before prepare phase:
coordinator can safely abort
transaction without additional
communication to workers

you can assume that the coordinator detects failures with a HELLO
protocol, or something similar
two-phase commit: nodes agree that they’re ready to commit before committing

message loss at any stage: handled by reliable transport; coordinator will time out and resend message

worker failure before prepare phase: coordinator can safely abort transaction without additional communication to workers
two-phase commit: nodes agree that they’re ready to commit before committing

message loss at any stage: handled by reliable transport; coordinator will time out and resend message

worker failure before prepare phase: coordinator can safely abort transaction without additional communication to workers
two-phase commit: nodes agree that they’re ready to commit before committing

message loss at any stage: handled by reliable transport; coordinator will time out and resend message

worker failure before prepare phase: coordinator can safely abort transaction without additional communication to workers
two-phase commit: nodes agree that they’re ready to commit before committing

message loss at any stage: handled by reliable transport; coordinator will time out and resend message

worker failure before prepare phase: coordinator can safely abort transaction without additional communication to workers
**two-phase commit**: nodes agree that they’re ready to commit before committing.

- **Client**
- **Coordinator**
- **A-M server**
- **N-Z server**

- **message loss at any stage**: handled by reliable transport; coordinator will time out and resend message.
- **Worker failure before prepare phase**: coordinator can safely abort transaction without additional communication to workers.

Diagram:
- Client sends **commit** to coordinator.
- Coordinator sends **prepare** to A-M server, which sends **prepare** to N-Z server.
- If A-M server fails, coordinator sends **prepare X** to N-Z server.
- If message loss occurs, coordinator will time out and resend message.
two-phase commit: nodes agree that they’re ready to commit before committing

- **client**
  - commit
  - timeout; resend

- **coordinator**
  - ok
  - commit
  - prepare X
  - prepare

- **A-M server**
  - prepare X

- **N-Z server**

message loss at any stage: handled by reliable transport; coordinator will time out and resend message

worker failure before prepare phase: coordinator can safely abort transaction without additional communication to workers

Katrina LaCurts | lacurts@mit.edu | 6.033 2021
two-phase commit: nodes agree that they’re ready to commit before committing

client

coordinator

A-M server

N-Z server

commit

ok

prepare

prepare

X

prepare

prepare

timeout; resend

message loss at any stage: handled by reliable transport; coordinator will time out and resend message

worker failure before prepare phase: coordinator can safely abort transaction without additional communication to workers
two-phase commit: nodes agree that they’re ready to commit before committing

- **client**
  - commit

- **coordinator**
  - ok
  - commit
  - prepare

- **A-M server**

- **N-Z server**

**message loss at any stage:** handled by reliable transport; coordinator will time out and resend message

**worker failure before prepare phase:** coordinator can safely abort transaction without additional communication to workers
two-phase commit: nodes agree that they’re ready to commit before committing

- client
- coordinator
- A-M server
- N-Z server

message loss at any stage: handled by reliable transport; coordinator will time out and resend message

worker failure before prepare phase: coordinator can safely abort transaction without additional communication to workers
two-phase commit: nodes agree that they’re ready to commit before committing

Message loss at any stage: handled by reliable transport; coordinator will time out and resend message.

Worker failure before prepare phase: Coordinator can safely abort transaction without additional communication to workers.
two-phase commit: nodes agree that they’re ready to commit before committing

client

ok

commit
	timeout; resend

---

coordinator

---

A-M server

prepare

 tolerates

X

timeout; resend

prepare

N-Z server

message loss at any stage: handled by reliable transport; coordinator will time out and resend message

worker failure before prepare phase: coordinator can safely abort transaction without additional communication to workers
two-phase commit: nodes agree that they’re ready to commit before committing

message loss at any stage: handled by reliable transport; coordinator will time out and resend message

worker failure before prepare phase: coordinator can safely abort transaction without additional communication to workers

thanks to sequence numbers, A-M will ACK the second prepare message but not reprocess it
two-phase commit: nodes agree that they’re ready to commit before committing

message loss at any stage: handled by reliable transport; coordinator will time out and resend message

worker failure before prepare phase: coordinator can safely abort transaction without additional communication to workers

thanks to sequence numbers, A-M will ACK the second prepare message but not reprocess it
two-phase commit: nodes agree that they’re ready to commit before committing

message loss at any stage: handled by reliable transport; coordinator will time out and resend message

worker failure before prepare phase: coordinator can safely abort transaction without additional communication to workers

thanks to sequence numbers, A-M will ACK the second prepare message but not reprocess it
**two-phase commit:** nodes agree that they’re ready to commit before committing

- **Client** sends a commit message to the **coordinator**.
- The **coordinator** sends a prepare message to **A-M server** and **N-Z server**.
- The **servers** confirm prepare status and send an ok message back to the **coordinator**.
- The **coordinator** then sends a commit message to the **servers**.
- If there is message loss at any stage, handled by reliable transport; coordinator will time out and resend message.
- Worker failure before prepare phase: coordinator can safely abort transaction without additional communication to workers.
- Worker failure during prepare phase.
**two-phase commit**: nodes agree that they’re ready to commit before committing

- **client**
- **coordinator**
- **A-M server**
- **N-Z server**

**Message Loss at Any Stage**: handled by reliable transport; coordinator will time out and resend message.

**Worker Failure Before Prepare Phase**: coordinator can safely abort transaction without additional communication to workers.

**Worker Failure During Prepare Phase**
two-phase commit: nodes agree that they’re ready to commit before committing

- message loss at any stage: handled by reliable transport; coordinator will time out and resend message
- worker failure before prepare phase: coordinator can safely abort transaction without additional communication to workers
- worker failure during prepare phase
two-phase commit: nodes agree that they’re ready to commit before committing

message loss at any stage: handled by reliable transport; coordinator will time out and resend message

worker failure before prepare phase: coordinator can safely abort transaction without additional communication to workers

worker failure during prepare phase
**two-phase commit:** nodes agree that they’re ready to commit before committing

- **message loss at any stage:** handled by reliable transport; coordinator will time out and resend message
- **worker failure before prepare phase:** coordinator can safely abort transaction without additional communication to workers
- **worker failure during prepare phase**
two-phase commit: nodes agree that they’re ready to commit before committing

- **client**
  - commit
  - abort

- **coordinator**
  - ok
  - commit
  - prepare
  - prepare
  - abort

- **A-M server**

- **N-Z server**

- **message loss at any stage:** handled by reliable transport; coordinator will time out and resend message

- **worker failure before prepare phase:** coordinator can safely abort transaction without additional communication to workers

- **worker failure during prepare phase:** coordinator can safely abort transaction, will send explicit abort messages to live workers
two-phase commit: nodes agree that they’re ready to commit before committing

message loss at any stage: handled by reliable transport; coordinator will time out and resend message

worker failure before prepare phase: coordinator can safely abort transaction without additional communication to workers

worker failure during prepare phase: coordinator can safely abort transaction, will send explicit abort messages to live workers

worker failure during commit phase
**two-phase commit**: nodes agree that they’re ready to commit before committing

- **Message loss at any stage**: handled by reliable transport; coordinator will time out and resend message.

- **Worker failure before prepare phase**: coordinator can safely abort transaction without additional communication to workers.

- **Worker failure during prepare phase**: coordinator can safely abort transaction, will send explicit abort messages to live workers.

- **Worker failure during commit phase**
two-phase commit: nodes agree that they’re ready to commit before committing

message loss at any stage: handled by reliable transport; coordinator will time out and resend message

worker failure before prepare phase: coordinator can safely abort transaction without additional communication to workers

worker failure during prepare phase: coordinator can safely abort transaction, will send explicit abort messages to live workers

worker failure during commit phase

if workers fail after the commit point, we cannot abort the transaction. workers must be able to recover into a prepared state, and then commit
two-phase commit: nodes agree that they’re ready to commit before committing

message loss at any stage: handled by reliable transport; coordinator will time out and resend message

worker failure before prepare phase: coordinator can safely abort transaction without additional communication to workers

worker failure during prepare phase: coordinator can safely abort transaction, will send explicit abort messages to live workers

worker failure during commit phase: coordinator cannot abort the transaction
**two-phase commit**: nodes agree that they’re ready to commit before committing

- **message loss at any stage**: handled by reliable transport; coordinator will time out and resend message
  
- **worker failure before prepare phase**: coordinator can safely abort transaction without additional communication to workers
  
- **worker failure during prepare phase**: coordinator can safely abort transaction, will send explicit abort messages to live workers
  
- **worker failure during commit phase**: coordinator cannot abort the transaction

workers write **PREPARE** records once prepared. the recovery process — reading through the log — will indicate which transactions are prepared but not committed
**two-phase commit:** nodes agree that they’re ready to commit before committing

- **Message loss at any stage:** handled by reliable transport; coordinator will time out and resend message
- **Worker failure before prepare phase:** coordinator can safely abort transaction without additional communication to workers
- **Worker failure during prepare phase:** coordinator can safely abort transaction, will send explicit abort messages to live workers
- **Worker failure during commit phase:** coordinator *cannot* abort the transaction; prepared workers must commit the transaction during recovery

---

- **Client**
- **Coordinator**
- **A-M server**
- **N-Z server**

<table>
<thead>
<tr>
<th></th>
<th>Client</th>
<th>Coordinator</th>
<th>A-M Server</th>
<th>N-Z Server</th>
</tr>
</thead>
<tbody>
<tr>
<td>ok</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>commit</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>ok</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>ok</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>
**two-phase commit**: nodes agree that they’re ready to commit before committing

<table>
<thead>
<tr>
<th>client</th>
<th>coordinator</th>
<th>A-M server</th>
<th>N-Z server</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td></td>
<td>commit?</td>
<td></td>
</tr>
<tr>
<td>ok</td>
<td>prepare</td>
<td>prepare</td>
<td>commit</td>
</tr>
<tr>
<td>commit</td>
<td>commit</td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

message loss at any stage: handled by reliable transport; coordinator will time out and resend message

worker failure before prepare phase: coordinator can safely abort transaction without additional communication to workers

worker failure during prepare phase: coordinator can safely abort transaction, will send explicit abort messages to live workers

worker failure during commit phase: coordinator cannot abort the transaction; prepared workers must commit the transaction during recovery
**two-phase commit:** nodes agree that they’re ready to commit before committing

- **Message loss at any stage:** handled by reliable transport; coordinator will time out and resend message

- **Worker failure before prepare phase:** coordinator can safely abort transaction without additional communication to workers

- **Worker failure during prepare phase:** coordinator can safely abort transaction, will send explicit abort messages to live workers

- **Worker failure during commit phase:** coordinator cannot abort the transaction; prepared workers must commit the transaction during recovery
two-phase commit: nodes agree that they’re ready to commit before committing

**client**  
**coordinator**  
**A-M server**  
**N-Z server**

- **Commit**
  - **Prepare**
    - **Prepare**
      - **Commit?**
  - **Commit**

**Message loss at any stage:** handled by reliable transport; coordinator will time out and resend message.

**Worker failure before prepare phase:** coordinator can safely abort transaction without additional communication to workers.

**Worker failure during prepare phase:** coordinator can safely abort transaction, will send explicit abort messages to live workers.

**Worker failure during commit phase:** coordinator cannot abort the transaction; prepared workers must commit the transaction during recovery.
**two-phase commit:** nodes agree that they’re ready to commit before committing

- **message loss at any stage:** handled by reliable transport; coordinator will time out and resend message

- **worker failure before prepare phase:** coordinator can safely abort transaction without additional communication to workers

- **worker failure during prepare phase:** coordinator can safely abort transaction, will send explicit abort messages to live workers

- **worker failure during commit phase:** coordinator cannot abort the transaction; prepared workers must commit the transaction during recovery
**two-phase commit:** nodes agree that they’re ready to commit before committing

- **Message loss at any stage:** handled by reliable transport; coordinator will time out and resend message.

- **Worker failure before prepare phase:** coordinator can safely abort transaction without additional communication to workers.

- **Worker failure during prepare phase:** coordinator can safely abort transaction, will send explicit abort messages to live workers.

- **Worker failure during commit phase:** coordinator cannot abort the transaction; prepared workers must commit the transaction during recovery.
**two-phase commit:** nodes agree that they’re ready to commit before committing

- **Message loss at any stage:** handled by reliable transport; coordinator will time out and resend message

- **Worker failure before prepare phase:** coordinator can safely abort transaction without additional communication to workers

- **Worker failure or coordinator failure during prepare phase:** coordinator can safely abort transaction, will send explicit abort messages to live workers

- **Worker failure during commit phase:** coordinator cannot abort the transaction; prepared workers must commit the transaction during recovery
**two-phase commit**: nodes agree that they’re ready to commit before committing

- **Message loss at any stage**: handled by reliable transport; coordinator will time out and resend message.

- **Worker failure before prepare phase**: coordinator can safely abort transaction without additional communication to workers.

- **Worker failure or coordinator failure during prepare phase**: coordinator can safely abort transaction, will send explicit abort messages to live workers.

- **Worker failure during commit phase**: coordinator cannot abort the transaction; prepared workers must commit the transaction during recovery.
two-phase commit: nodes agree that they’re ready to commit before committing

message loss at any stage: handled by reliable transport; coordinator will time out and resend message

worker failure before prepare phase: coordinator can safely abort transaction without additional communication to workers

worker failure or coordinator failure during prepare phase: coordinator can safely abort transaction, will send explicit abort messages to live workers

worker failure during commit phase: coordinator cannot abort the transaction; prepared workers must commit the transaction during recovery
two-phase commit: nodes agree that they’re ready to commit before committing

Message loss at any stage: handled by reliable transport; coordinator will time out and resend message

Worker failure before prepare phase: coordinator can safely abort transaction without additional communication to workers

Worker failure or coordinator failure during prepare phase: coordinator can safely abort transaction, will send explicit abort messages to live workers

Worker failure during commit phase: coordinator cannot abort the transaction; prepared workers must commit the transaction during recovery
**two-phase commit**: nodes agree that they’re ready to commit before committing

- **Message loss at any stage**: handled by reliable transport; coordinator will time out and resend message

- **Worker failure before prepare phase**: coordinator can safely abort transaction without additional communication to workers

- **Worker failure or coordinator failure during prepare phase**: coordinator can safely abort transaction, will send explicit abort messages to live workers

- **Worker failure during commit phase**: coordinator *cannot* abort the transaction; prepared workers must commit the transaction during recovery
two-phase commit: nodes agree that they’re ready to commit before committing

message loss at any stage: handled by reliable transport; coordinator will time out and resend message

worker failure before prepare phase: coordinator can safely abort transaction without additional communication to workers

worker failure or coordinator failure during prepare phase: coordinator can safely abort transaction, will send explicit abort messages to live workers

worker failure or coordinator failure during commit phase: coordinator cannot abort the transaction; machines must commit the transaction during recovery
two-phase commit: nodes agree that they’re ready to commit before committing

message loss at any stage: handled by reliable transport; coordinator will time out and resend message

worker failure before prepare phase: coordinator can safely abort transaction without additional communication to workers

worker failure or coordinator failure during prepare phase: coordinator can safely abort transaction, will send explicit abort messages to live workers

worker failure or coordinator failure during commit phase: coordinator cannot abort the transaction; machines must commit the transaction during recovery
two-phase commit: nodes agree that they’re ready to commit before committing

message loss at any stage: handled by reliable transport; coordinator will time out and resend message

worker failure before prepare phase: coordinator can safely abort transaction without additional communication to workers

worker failure or coordinator failure during prepare phase: coordinator can safely abort transaction, will send explicit abort messages to live workers

worker failure or coordinator failure during commit phase: coordinator cannot abort the transaction; machines must commit the transaction during recovery
**two-phase commit**: nodes agree that they’re ready to commit before committing

- **message loss at any stage**: handled by reliable transport; coordinator will time out and resend message
- **worker failure before prepare phase**: coordinator can safely abort transaction without additional communication to workers
- **worker failure or coordinator failure during prepare phase**: coordinator can safely abort transaction, will send explicit abort messages to live workers
- **worker failure or coordinator failure during commit phase**: coordinator cannot abort the transaction; machines must commit the transaction during recovery
**two-phase commit:** nodes agree that they’re ready to commit before committing

---

**Problem:** in our example, when workers fail, some of the data (e.g., accounts A-M) is completely unavailable

---

**Message loss at any stage:** handled by reliable transport; coordinator will time out and resend message

**Worker failure before prepare phase:** coordinator can safely abort transaction without additional communication to workers

**Worker failure or coordinator failure during prepare phase:** coordinator can safely abort transaction, will send explicit abort messages to live workers

**Worker failure or coordinator failure during commit phase:** coordinator cannot abort the transaction; machines must commit the transaction during recovery
two-phase commit: nodes agree that they’re ready to commit before committing

- message loss at any stage: handled by reliable transport; coordinator will time out and resend message
- worker failure before prepare phase: coordinator can safely abort transaction without additional communication to workers
- worker failure or coordinator failure during prepare phase: coordinator can safely abort transaction, will send explicit abort messages to live workers
- worker failure or coordinator failure during commit phase: coordinator cannot abort the transaction; machines must commit the transaction during recovery

solution: replicate data. but to address this problem, we need to worry about keeping multiple copies of the same piece of data consistent, and what type of consistency we even want
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.

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

* shadow copies are used in some systems.
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**.

our lingering problem is that we aren’t replicating data across multiple machines.

* shadow copies are used in some systems.
two-phase commit allows us to achieve multi-site atomicity; transactions remain atomic even when they require communication with multiple machines.

In two-phase commit, failures prior to the commit point can be aborted. Failures after the commit point cannot; machines must commit the transaction in recovery.

Our remaining issue deals with availability and replication: we will replicate data across sites to improve availability, but must deal with keeping multiple copies of the data consistent.