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.* shadow copies are used in some systems

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

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

\begin{center}
\begin{tabular}{c|c|c}
client & coordinator & A-M server \\
\hline
begin & & \\
\hline
ok & & \\
\hline
\text{A-amount} & & \\
\hline
ok & & \\
\hline
\text{B+amount} & & \\
\hline
ok & & \\
\hline
commit & & \\
\hline
ok & & \\
\end{tabular}
\end{center}
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

ok
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

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

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

---

**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

client  coordinator  A-M server  N-Z server

- ok
- commit  timeout; resend
- prepare X
- prepare
- prepare

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

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

  - **Commit**
  - **Prepare**
  - **Abort**

  - **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

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

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**: 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
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

<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>ok</td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td>commit</td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td>prepare</td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td>coordinator recovers</td>
<td>abort</td>
<td>abort</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 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.

**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**.

*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.