Notes 6: Reliability
Reliable Distributed DB Management

Reliability
Failure models
Scenarios
Reliability

Correctness
Serializability
Atomicity
Persistence

Availability
Types of Failures

Processor failures
Halt, delay, restart, erratic execution

Storage failures
Volatile vs. non-volatile storage failures
Atomic write violations, transient errors, localized vs. global failures

Network failures
Lost message, out-of-order messages, partitions, bounded delay
Types of Failures

Unintended vs. malevolent failures

Single vs. multiple failures

Detectable vs. undetectable failures
Failure Models

Cannot protect against everything

Unlikely failures
  E.g., flooding in the Sahara
  *Ten of the Strangest Data Center Outages* [goo.gl/DcQysr]

Failures expensive to protect against
  E.g., earthquakes

Failures we know how to easily cope with
  E.g., using message sequence numbers
Failure Models

Events

- Desired
- Undesired

- Expected
- Unexpected
Node Models

1. Fail-stop nodes

- **perfect**
- **halted**
- **recovery**
- **perfect**

**volatile memory lost**

**stable storage ok**

**time**
Node Models

2. Byzantine nodes

At any given time, at most some fraction (e.g., 1/2 or 1/3) of nodes are failing
Network Models

1. Reliable network
   In order messages
   No spontaneous messages
   Timeout $T_D$
   No response within $T_D$ means destination is down (not paused)
   No lost messages except due to node failures
Network Models

Variation of reliable network
Persistent messages
  If destination down, network will eventually deliver message
  Simplifies node recovery, but inefficient
  Not considered here
Network Models

2. Partitionable network
   In order messages
   No spontaneous messages

No timeout
Nodes can have different views of the failures
Amazon Easter Outage

Amazon’s lengthy cloud outage shows the danger of complexity
Amazon has published a detailed description of the prolonged failure that …

by Peter Bright - Apr 30, 2011 3:12pm PDT

Misconfiguration → overloaded router → partition [goo.gl/z2nPq]
Scenarios

Reliable network
   Fail-stop nodes
      No data replication (1)
      Data replication (2)

Partitionable network
   Fail-stop nodes (3)
No Data Replication

Reliable network, fail-stop nodes

Basic idea: node $P_\alpha$ controls $X$
$P_\alpha$ does concurrency control for $X$
$P_\alpha$ does recovery for $X$
Single control point simplifies both
Process Models

Transaction $T$ wants to access $X$

$P_T$ is process that represents $T$ at this node
Process Models

Cohorts
Application code responsible for remote access
Application interacts with local DBMS

Transaction Manager
“System” handles distribution and remote access
Process Models

Cohorts

- USER
- Local DBMS
- Local DBMS
- Local DBMS

Symbols:
- Dashed line: Spawn Process
- Double arrows: Communication
- Single arrow: Data Access
Process Models

Transaction Manager

USER

Trans Mgr

Local DBMS

Trans Mgr

Local DBMS

Trans Mgr

Local DBMS

Data Access
Distributed Commit Problem

Transaction T

Actions $a_1, a_2$

Action $a_3$

Actions $a_4, a_5$
Centralized Two-Phase Commit

Coordinator

W
\[\begin{array}{c}
g o \\
= exec * \\
ok *\\
= commit *
\end{array}\]

A

C

Participant

W
\[\begin{array}{c}
exec \\
ok \\
ok \\
\text{commit} *
\end{array}\]

A

C

\[\text{incoming msg} \quad \text{outgoing msg} \quad * = \text{everyone}\]
Centralized Two-Phase Commit

No lost messages (for now)
   Reliable network
   Will discuss node failures next

When participant enters \( W \) state
   It must have acquired all resources
   It can only abort or commit if so instructed by the coordinator

Coordinator only enters \( C \) state if all participants are in \( W \)
   It is certain that all will eventually commit
Handling Node Failures

At failing node
Coordinator and participant logs are used to reconstruct state before failure
Handling Node Failures

Example
Participant log contains $W$ on recovery

<table>
<thead>
<tr>
<th>$T_1$</th>
<th>...</th>
<th>$T_1$</th>
<th>...</th>
<th>$T_1$</th>
</tr>
</thead>
<tbody>
<tr>
<td>X undo/redo info</td>
<td></td>
<td>Y undo/redo info</td>
<td></td>
<td>W state</td>
</tr>
</tbody>
</table>
Handling Node Failures

Example

Participant log contains $W$ on recovery

<table>
<thead>
<tr>
<th></th>
<th>$T_1$</th>
<th></th>
<th>$T_1$</th>
<th></th>
<th>$T_1$</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>$X$</td>
<td>undo/redo info</td>
<td>$Y$</td>
<td>undo/redo info</td>
<td>$W$</td>
</tr>
</tbody>
</table>

Recovery steps

1. Notice that $T_1$ is in $W$ state
2. Obtain $X, Y$ write locks (no read locks—why?)
3. Wait for message from coordinator (or ask about outcome)
Handling Node Failures

Other examples
No $W$ record on log $\Rightarrow$ abort $T_1$
Have $C$ record on log $\Rightarrow$ finish $T_1$
Handling Node Failures

At the protocol level
Add timeouts to cope with messages lost during failures
Add finish (F) state for coordinator ~ all done, can forget outcome
Coordinator

goto exec

ok commit

nok abort

ok cok

nok –
cok –

\( \text{cok} = \text{committed ok} \)
Coordinator

$t = \text{timeout}$
$c\text{ping} = \text{coordinator ping}$
Participant

I

exec
ok

exec
nok

W

commit
cok

A

abort
nok

C
Participant
Participant

done = either cok or nok for coordinator

equivalent to finish state
Presumed Abort Protocol

F and A states combined in coordinator
Saves persistent space (allows coordinator to forget sooner)

Presumed commit is analogous
Presumed Abort Protocol

Coordinator

I

W

go
exec *

nok, t
abort *

A/F

ping
abort

ok *
commit *

C

ping
commit

ping
commit

t
cping

Notes 6
Simplified Logging

All state transitions must be logged

Where to log participant state?
Both coordinator and participant
Participant only
Simplified Logging

Example
Coordinator tracking participant OKs

<table>
<thead>
<tr>
<th></th>
<th>T₁ start participants { a, b }</th>
<th>...</th>
<th>T₁ ok received from a</th>
<th>...</th>
</tr>
</thead>
</table>

After failure, we know we are still waiting for OK from node b

Alternative
Do not log receipt of OKs
Abort T₁ on recovery
Simplified Logging

Example
Logging receipt of cok messages

If logged then coordinator can recover state

If not logged
    resend commit *
    participants reply done if duplicate
Variants of 2PC

Linear

Hierarchical
Variants of 2PC

Distributed

Nodes broadcast all messages
Every node knows when to commit
2PC is Blocking

Sample scenario
2PC is Blocking

Case I

\[ P_1 \rightarrow W \]

coordinator sent commits

\[ P_1 \rightarrow C \]

Case II

\[ P_1 \rightarrow A \]

Surviving participants \( P_2, P_3, P_4 \) cannot safely abort or commit
Three-Phase Commit

Non-blocking commit protocol
Assumes that a failed node stays down forever

Key idea
Before committing the coordinator tells participants that everyone is OK
Three-Phase Commit

Coordinator

Participant

** = all non-failed nodes
3PC Recovery Rules

Termination protocol

Survivors try to complete transaction, based on their current states

Goal

If dead nodes committed or aborted, then survivors should not contradict; else survivors can do as they please
3PC Recovery Rules

Let \( \{ S_1, S_2, ..., S_n \} \) be survivor nodes

If one or more \( S_i = \text{COMMIT} \) \( \implies \) COMMIT \( T \)

If one or more \( S_i = \text{ABORT} \) \( \implies \) ABORT \( T \)

If one or more \( S_i = \text{PREPARE} \) \( \implies \) COMMIT \( T \)

\( T \) could not have aborted

If no \( S_i = \text{PREPARE} \) (or \( \text{COMMIT} \)) \( \implies \) ABORT \( T \)

\( T \) could not have committed
3PC Recovery Rules

Example 1

\[ ? \times \quad \bigcirc \ P \]

\[ ? \times \quad \bigcirc \ W \]

\[ \bigcirc \ W \]
3PC Recovery Rules

Example 2

\[ ? \times \quad \circ \quad I \]

\[ ? \times \quad \circ \quad W \]

\[ \circ \quad W \]
3PC Recovery Rules

Example 3

? ✗

? ✗

? ✗

P

P

C
3PC Recovery Rules

Example 4

? × P

? × W

? × A
3PC Recovery Rules

Once survivors make decision, they must select new coordinator to continue 3PC

<table>
<thead>
<tr>
<th>Time 1</th>
<th>Time 2</th>
<th>Time 3</th>
<th>Time 4</th>
</tr>
</thead>
<tbody>
<tr>
<td>W</td>
<td>P</td>
<td>P</td>
<td>C</td>
</tr>
<tr>
<td>W</td>
<td>P</td>
<td>C</td>
<td>C</td>
</tr>
<tr>
<td></td>
<td>P</td>
<td></td>
<td></td>
</tr>
<tr>
<td>Decide to commit</td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

Notes:
- P: Participant
- W: Witness
- C: Coordinator
3PC Recovery Rules

When survivors continue 3PC, failed nodes do not count

** = all non-failed nodes
3PC Recovery Rules

3PC is unsafe with partitions

abort

commit
3PC Node Recovery

After node recovers from failure

Do not participate in termination protocol
3PC Node Recovery

After node recovers from failure

Do not participate in termination protocol

? \( \times \) \( \rightarrow \) A

\( W \rightarrow A \)
3PC Node Recovery

After node recovers from failure

Do not participate in termination protocol
3PC Node Recovery

After node recovers from failure

**Waits** until receives commit or abort decision from another node

![Diagram showing the process of 3PC Node Recovery](image-url)
3PC Node Recovery

Waiting for commit or abort decision from others is ok
Unless all nodes fail
3PC Node Recovery

Waiting for commit or abort decision from others is ok
Unless all nodes fail

Two options
A. Wait for all nodes to recover
B. Perform majority commit
3PC Node Recovery

A. Wait for all nodes to recover

Recovering node waits for either
1. Commit or abort decision from another node or
2. If all other nodes are up and recovering then 3PC can continue

No danger that there is a failed node that had committed or aborted
3PC Node Recovery

B. Perform majority commit

Want a gang of failed but recovered nodes to be able to terminate the transaction, even when the rest are still failing

Nodes are assigned votes, total is $V$

Majority is $M \geq \text{round}\left(\frac{(V + 1)}{2}\right)$

E.g., $V = 6 \rightarrow M \geq 4$

To make state transitions, coordinator requires messages from notes with a majority of votes
3PC with Majority Votes

Example 1

Nodes $P_2$, $P_3$, $P_4$ enter $W$ state and fail
When they recover, coordinator and $P_1$ are down
Each node has one vote, $V = 5$, $M \geq 3$
3PC with Majority Votes

Example 1

Nodes \(P_2, P_3, P_4\) enter \(W\) state and fail
When they recover, coordinator and \(P_1\) are down
Each node has one vote, \(V = 5, M \geq 3\)

Since \(P_2, P_3, P_4\) have majority, they know coordinator could not have gone to \(P\) without at least one of their votes \(\Rightarrow T\) can be aborted
3PC with Majority Votes

Example 2

Nodes $P_3$ and $P_4$ enter $P$ and $W$ state, then fail
When they recover, coordinator, $P_1$ and $P_2$ are down
Each node has one vote, $V = 5$, $M \geq 3$
3PC with Majority Votes

Example 2

Nodes $P_3$ and $P_4$ enter $P$ and $W$ state, then fail
When they recover, coordinator, $P_1$ and $P_2$ are down
Each node has one vote, $V = 5$, $M \geq 3$

Nodes $P_3$ and $P_4$ have insufficient votes $\Rightarrow$ they do nothing
3PC with Majority Votes

Majority ensures that any decision (e.g., preparing, committing) will be known to any future group making subsequent decisions.
3PC with Majority Votes

Example

\[ \begin{align*}
? & \not\in & W \\
\neg P & \not\in & W \\
W & \rightarrow & A
\end{align*} \]
3PC with Majority Votes

Example

\[ \begin{align*}
? \times & \quad \circ W \rightarrow P \rightarrow C \\
\circ W & \\
\times W & \rightarrow A
\end{align*} \]
3PC with Majority Votes

Need prepare to abort state

** = all participants with majority votes
3PC with Majority Votes

Example revisited

? X

PC X

W

W

W
3PC with Majority Votes

Example revisited

Scenario 1

OK for all remaining nodes to enter PC and eventually commit
The transaction could not have aborted
3PC with Majority Votes

Example revisited

Scenario 2

Same outcome as scenario 1

Even though most recently failed node was already in PA
The PA node will have to *commit* when it eventually recovers
3PC with Majority Votes

Example revisited

Scenario 3

Remaining nodes initiated abort, some entered PA state
3PC with Majority Votes

Example revisited

Scenario 3

```
Cannot make a decision
The transaction could have committed or aborted
```

Exercise: work out sequences of steps (will revisit later)
3PC with Majority Votes

Example revisited

Scenario 3

Will have to wait for either

1. Some node to recover in A or C
2. All nodes to recover and confirm that none were in A or C
3PC with Majority Votes

If survivors have majority and all states $W \implies$ try to abort

If survivors have majority and states in $\{ W, PC, C \} \implies$ try to commit

If survivors have majority and states in $\{ W, PA, A \} \implies$ try to abort

Otherwise block
3PC Comparison

Basic 3PC

Only nodes that have not failed participate in decision

Any remaining subgroup can terminate (even one node)

If all nodes fail, must wait for all to recover
3PC Comparison

3PC with majority votes

A group of failed but recovering nodes can terminate transaction

Need majority to commit

Blocking protocol
3PC Logging

When a node recovers, it uses its log as usual to determine the status of each transaction

- If commit logged $\Rightarrow$ redo if necessary
- If abort logged (or wait is missing) $\Rightarrow$ rollback if necessary
3PC Logging

When a node recovers, it uses its log as usual to determine the status of each transaction

- If commit logged $\implies$ redo if necessary
- If abort logged (or wait is missing) $\implies$ rollback if necessary
- If wait logged (or pre state) $\implies$
  - Reclaim locks held by T before crash
  - Try to terminate T (with other nodes)

Can start normal processing once locks secured for recovering transactions
Summary

Failure models ✔
  Nodes
  Networks

Reliable network, fail-stop nodes, no replication ✔
  Two-phase commit (blocking)
  Three-phase commit
    Basic (non-blocking)
    With majority votes

Next: replication