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

   - time
   - perfect
   - halted
   - recovery
   - perfect
   - volatile memory lost
   - stable storage ok
Node Models

2. Byzantine nodes

A

perfect arbitrary failure recovery perfect

B

C

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

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 \( P_a \) controls \( X \)
- \( P_a \) does concurrency control for \( X \)
- \( P_a \) 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

Local DBMS

Locks

Log

\( X \)
Process Models

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

Transaction Manager
“System” handles distribution and remote access

Distributed Commit Problem

Actions $a_1, a_2$
Action $a_3$
Actions $a_4, a_5$
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>T1</th>
<th>T1</th>
<th>T1</th>
</tr>
</thead>
<tbody>
<tr>
<td>X</td>
<td>Y</td>
<td>W</td>
</tr>
<tr>
<td>undo/redo info</td>
<td>undo/redo info</td>
<td>state</td>
</tr>
</tbody>
</table>
Handling Node Failures

Example
Participant log contains W on recovery

<table>
<thead>
<tr>
<th></th>
<th>T₁</th>
<th></th>
<th>T₁</th>
<th></th>
<th>T₁</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>X</td>
<td>-</td>
<td>Y</td>
<td>-</td>
<td>W</td>
</tr>
<tr>
<td>undo/redo info</td>
<td>undo/redo info</td>
<td>state</td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

Recovery steps
1. Notice that T₁ 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 ⇒ abort T₁
Have C record on log ⇒ finish T₁

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
Coordinator

Participant

Participant

Participant
**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

---

**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>Ti</th>
<th>Tk</th>
</tr>
</thead>
<tbody>
<tr>
<td>Ti 1 start participants (a, b)</td>
<td>Tk 1 ok received from a</td>
</tr>
</tbody>
</table>

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

Alternative
Do not log receipt of OKs
Abort Ti 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**
- ok commit
- ok commit
- ok commit

**Hierarchical**

**Variants of 2PC**

**Distributed**

Nodes broadcast all messages
Every node knows when to commit

**2PC is Blocking**

Sample scenario

- Coordinator
- P1
- P2
- P3
- W
- W
- W
### 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

Coordinators
- go: execute
- ok: commit
- abort

Participants
- check ok
- pre ack
- abort
- pre
- commit

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

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

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

If one or more \( S_i =\text{PREPARE} \) \( \implies \) \( \text{COMMIT} \) \( T \) could not have aborted

If no \( S_i =\text{PREPARE} \) (or \( \text{COMMIT} \)) \( \implies \) \( \text{ABORT} \) \( T \) could not have committed

3PC Recovery Rules

Example 1

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

3PC Recovery Rules

Example 2

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

3PC Recovery Rules

Example 3

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

3PC Recovery Rules

Example 3

\[ ? \times \quad \bigcirc \quad \bigcirc \quad P \]
\[ ? \times \quad \bigcirc \quad C \]
3PC Recovery Rules

Example 4

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

When survivors continue 3PC, failed nodes do not count

3PC is unsafe with partitions
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

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

3PC Node Recovery
After node recovers from failure
Wait until receives commit or abort decision from another node

\[ ? \quad \boxtimes \quad W \]
\[ P \quad \boxtimes \quad W \]
\[ W \rightarrow A \]
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

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

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 \( M \geq \text{round}(V+1)/2 \)

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, $V = 5$, $M \geq 3$

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, $V = 5$, $M \geq 3$

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.

Example

W → P → C

Need prepare to abort state

Coordinator

Participant
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

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

Scenario 3

Remaining nodes initiated abort, some entered PA state
**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**

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

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

If survivors have majority and states in \{W, PA, A\} $\Rightarrow$ 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 ⇒ redo if necessary
If abort logged (or wait is missing) ⇒ 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 ⇒ redo if necessary
If abort logged (or wait is missing) ⇒ rollback if necessary
If wait logged (or pre state)
  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