Reliable distributed database management

- Reliability
- Failure models
- Scenarios

Reliability

- Correctness
  - Serializability
  - Atomicity
  - Persistence
- Availability

Types of failures

- Processor failures
  - Halt, delay, restart, bezerk, ...
- Storage failures
  - Volatile, non-volatile, atomic write, transient errors, spontaneous failures
- Network failures
  - Lost message, out-of-order messages, partitions, bounded delay

More Types of failures

- Malevolent failures
- Multiple failures
- Detectable failures

Failure models

- Cannot protect against everything
- Unlikely failures (e.g., flooding in the Sahara)
- Expensive to protect failures (e.g., earthquake)
- Failures we know how to protect against (e.g., message sequence numbers; stable storage)
**Failure model:**

Desired

Undesired

Expected

Unexpected

---

**Node models**

1. Fail-stop nodes

   - Perfect halted recovery perfect

   - Volatile memory lost
   - Stable storage ok

---

(1) Byzantine nodes

   A Perfect
   - Perfect
   - Arbitrary failure
   - Recovery

   B

   C

At any given time, at most some fraction \( f \) of nodes failed (typically \( f < 1/2 \) or \( f < 1/3 \))

---

**Network models**

1. Reliable network

   - In order messages
   - No spontaneous messages
   - Timeout \( T_0 \)

   If no ack in \( T_0 \) sec.

   Destination down
   (not paused)

   I.e., no lost messages, except for node failures

---

**Variation of reliable net:**

- Persistent messages
  - If destination down, net will eventually deliver message
  - Simplifies node recovery, but leads to inefficiencies (hides too much)
  - Not considered here

---

**Network models**

2. Partitionable network

   - In order messages
   - No spontaneous messages

   - No timeout; nodes can have different view of failures
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

"$P_\alpha$ controls X" means
- $P_\alpha$ does concurrency control for X
- $P_\alpha$ does recovery for X

Say transaction T wants to access X:

$P_T$ is process that represents T at this node

Process models

(A) Cohorts

- Spawn process
- Communication
- Data Access
Process models

(B) Transaction servers (manager)

- Cohorts: application code responsible for remote access
- Transaction manager: “system” handles distribution, remote access

Distributed commit problem

Centralized two-phase commit

- Notation: Incoming message
  Outgoing message
  (* = everyone)
  - When participant enters “W” state:
    - it must have acquired all resources
    - it can only abort or commit if so instructed by a coordinator
  - Coordinator only enters “C” state if all participants are in “W”, i.e., it is certain that all will eventually commit

Handling node failures

- Coordinator and participant logs are used to reconstruct state before failure
Example: after participant fails:

Log:

<table>
<thead>
<tr>
<th>T1</th>
<th>X</th>
<th>...</th>
<th>T1</th>
<th>Y</th>
<th>...</th>
<th>T1</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>undo/redo info</td>
<td></td>
<td></td>
<td>info</td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td>undo/redo info</td>
<td></td>
<td></td>
<td>info</td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td>undo/redo info</td>
<td></td>
<td></td>
<td>info</td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

At recovery:
- T1 is in “W” state
- Obtain X,Y write locks (no read locks!)
- Wait for message from coordinator (or ask coordinator for outcome)

Other examples:
- No “W” record on log ⇒ abort T1
- See “C” record on log ⇒ finish T1

Next
- Add timeouts to cope with messages lost during crashes
- Add finish (“F”) state for coordinator – all done, can forget outcome

Coordinator

Coordinator
Participant

I
exec
ok

W
exec
nok

A
abort

C
commit
c-ok

Presumed abort protocol

• “F” and “A” states combined in coordinator
• Saves persistent space (forget quicker)
• Presumed commit is analogous

Presumed abort-coordinator (participant unchanged)

Participant

I
exec
ok

W
exec
nok

A
abort

C
commit
c-ok

“done” message counts as either c-ok or n-ok for coordinator

Remember:
all state transitions must be logged

Example: tracking who has sent “OK” msgs
Log at coord:

\[
\begin{array}{c|c|c}
T_1 & T_1 & T_1 \\
\hline
\text{start} & \text{OK from a} & \text{RCV} \\
\text{part}=(a,b) & & \\
\hline
\end{array}
\]

• After failure, we know still waiting for OK from node b
• Alternative: do not log receipts of “OK”'s
  abort T_1
Example: logging receipt of C-OK messages
- If logged, can recover state
- If not logged:
  - resend commit *
  - participants reply “done” if duplicate

2PC is blocking
Sample scenario:
Coord P2
P1 W
P3 W
P4 W

Case I:
P1 \rightarrow “W”; coordinator sent commits
P1 \rightarrow “C”

Case II:
P1 \rightarrow NOK; P1 \rightarrow A
⇒ P2, P3, P4 (surviving participants)
cannot safely abort or commit transaction

Variants of 2PC
- Linear
- Hierarchical

Variants of 2PC
- Distributed
  - Nodes broadcast all messages
  - Every node knows when to commit

3PC = non-blocking commit
- Assume: failed node is down forever
- Key idea: before committing, coordinator tells participants everyone is ok
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...

Example:

- Let \( \{S_1, S_2, \ldots, S_n\} \) be survivor sites
- If one or more \( S_i = \text{COMMIT} \) \( \Rightarrow \text{COMMIT T} \)
- If one or more \( S_i = \text{ABORT} \) \( \Rightarrow \text{ABORT T} \)
- If one or more \( S_i = \text{PREPARE} \) \( \Rightarrow \text{T could not have aborted} \Rightarrow \text{COMMIT T} \)
- If no \( S_i = \text{PREPARE} \) (or \( \text{COMMIT} \)) \( \Rightarrow \text{T could not have committed} \Rightarrow \text{ABORT T} \)

Example:

- ? \( \times \) \( \bigcirc \) \( P \)
- ? \( \times \) \( \bigcirc \) \( W \)

Example:

- ? \( \times \) \( \bigcirc \) \( I \)
- ? \( \times \) \( \bigcirc \) \( W \)

Example:

- ? \( \times \) \( \bigcirc \) \( P \)
- ? \( \times \) \( \bigcirc \) \( C \)
Example:

\[
\begin{array}{c}
? \\
W
\end{array}
\begin{array}{cc}
P & A
\end{array}
\]

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

\[
\begin{array}{cccc}
P & W & C & C \\
\text{Time 1} & \text{Time 2} & \text{Time 3} & \text{Time 4}
\end{array}
\]

Note: when survivors continue 3PC, failed nodes do not count:

E.g., \( \text{ack**} \Rightarrow \text{ack***} \) when ack's received from all non-failed nodes.

Node recovery:

- After node N recovers from failure:
  - do not participate in termination protocol (why?)

\[
\begin{array}{c}
? \\
W
\end{array}
\begin{array}{c}
P \\
W
\end{array}
\begin{array}{c}
W \rightarrow A
\end{array}
\]

Note: 3PC unsafe with partitions!

Node recovery:

- After node N recovers from failure:
  - do not participate in termination protocol (why?)

\[
\begin{array}{c}
? \\
W
\end{array}
\begin{array}{c}
P \\
W
\end{array}
\begin{array}{c}
W \rightarrow A
\end{array}
\]

later on...
Node recovery:
• After node N recovers from failure:
  – do not participate in termination protocol
    (why?)
  – wait until it hears commit or abort
decision from operational node

Waiting for commit/abort decision from other node is ok, unless all fail:

Two options for all-failed problem:
(A) Wait for all to recover
(B) Majority commit

Option A
• Recovering node waits for either:
  (1) commit/abort outcome for T from
other node
  (2) all nodes that participated in T are
up and recovering:
    ⇒ then 3PC can continue
    (no danger that a failed node could have
aborted or committed)

Option B
• Nodes are assigned votes, total is V
  Majority is \( \frac{V+1}{2} \) e.g., \( \begin{align*}
    V &= 5 \\
    \text{Maj} &= 3
  \end{align*} \)
  \( \begin{align*}
    V &= 6 \\
    \text{Maj} &= 4
  \end{align*} \)
• To make state transitions, coordinator
requires messages from nodes with a
majority of votes
Example(1): Coord \(\times\) \(\exists\) \(\exists\) \(\exists\) \(P\) \(\rightarrow\) \(W\)
\(P_1\) \(\times\) \(\exists\) \(\exists\) \(\exists\) \(P\) \(\rightarrow\) \(W\)
\(P_2\) \(\times\) \(\exists\) \(\exists\) \(\exists\) \(P\) \(\rightarrow\) \(W\)
- Nodes \(P_2, P_3, P_4\) enter "W" state and fail
- When they recover, coord. and \(P_1\) are down
- Each node has 1 vote, \(V=5\), Maj=3

Example(1): Coord \(\times\) \(\exists\) \(\exists\) \(\exists\) \(P\) \(\rightarrow\) \(W\)
\(P_1\) \(\times\) \(\exists\) \(\exists\) \(\exists\) \(P\) \(\rightarrow\) \(W\)
\(P_2\) \(\times\) \(\exists\) \(\exists\) \(\exists\) \(P\) \(\rightarrow\) \(W\)
- Nodes \(P_2, P_3, P_4\) enter "W" state and fail
- When they recover, coord. and \(P_1\) are down
- Each node has 1 vote, \(V=5\), Maj=3
- Since \(P_2, P_3, P_4\) have majority, they know coord. could not have gone to "P" without at least one of their votes
- Therefore, \(T\) can be aborted!

Example(2): Coord \(\times\) \(\exists\) \(\exists\) \(\exists\) \(P\) \(\rightarrow\) "P"
\(P_1\) \(\times\) \(\exists\) \(\exists\) \(\exists\) \(P\) \(\rightarrow\) "W"
\(P_2\) \(\times\)
- Each node has 1 vote; \(V=5\), Maj=3
- Nodes fail after entering states shown; \(P_3, P_4\) recover

Example(2): Coord \(\times\) \(\exists\) \(\exists\) \(\exists\) \(P\) \(\rightarrow\) "P"
\(P_1\) \(\times\) \(\exists\) \(\exists\) \(\exists\) \(P\) \(\rightarrow\) "W"
\(P_2\) \(\times\)
- Each node has 1 vote; \(V=5\), Maj=3
- Nodes fail after entering states shown; \(P_3, P_4\) recover
- Termination rule says we can try to commit, but \(P_3, P_4\) do not have enough votes, so they do nothing!
- \(P_3, P_4\) doing nothing is good because later on, coord. \(P_1, P_2\) could abort \(T\)

Summary:
Majority rule ensures that any decision (e.g., Preparing, committing) will be known to any future group making a decision

Important Detail for Majority 3PC
- Example:
  \(\times\) \(\exists\) \(\exists\) \(\exists\) \(P\) \(\times\)
  \(\times\) \(\exists\) \(\exists\) \(\exists\) \(W\)
  \(\times\) \(\exists\) \(\exists\) \(\exists\) \(W\) \(\rightarrow\) \(A\)
Important Detail for Majority 3PC

- Example:

Need “Prepare To Abort” State

Example Revisited

Example Revisited

Example Revisited -II

Example Revisited -II
3PC with Majority Voting

- 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

Comparison

Option A: only nodes that have not failed participate in 3PC
- Any size group can terminate (even one node)
- If all nodes fail, must wait for all to recover

Comparison

Option B: Majority voting
- A group of failed + recovering nodes can terminate transaction (with majority of votes)
- Need majority for every commit
  - blocking protocol!

Reminder

- When node recovers, it uses its log in a normal fashion to determine status of transactions:
  - if commit found in log \( \Rightarrow \) redo if necessary
  - if abort found (or no “W” record) \( \Rightarrow \) rollback if necessary

Reminder - Continued

- if in “W” state (or “P” state):
  - reclaim locks held by \( T \) before crash
  - try to terminate \( T \) (with other nodes)
  - after locks claimed for “in doubt” transactions, start normal processing

Final note

- If nodes use 2P locking, global deadlocks possible

Local WFG:
- no cycles

Local WFG:
- no cycles!
• Need to “combine” WFGs to discover global deadlock

Problem: False deadlocks

Exercise

• Assume all waits are due to transaction lock requests
• Assume transactions well formed and 2PL; scheduler legal
• Show that false deadlocks are not possible

• Many deadlock solutions
  - Distributed vs. centralized
  - Detection vs. prevention
    • timeouts
    • wait-die
    • wound-wait
  • Covered in CS245