Outline

- Introduction
- Background
- Distributed DBMS Architecture
- Distributed Database Design
- Distributed Query Processing
- Distributed Transaction Management
  - Transaction Concepts and Models
  - Distributed Concurrency Control
  - Distributed Reliability
- Building Distributed Database Systems (RAID)
- Mobile Database Systems
- Privacy, Trust, and Authentication
- Peer to Peer Systems
Useful References


Termination Protocols

Message sent by an operational site

abort – If trans. state is abort
   (If in abort)

committable – If trans. state is committable
   (If in p or c)

non-committable – If trans. state is neither committable nor abort
   (If in initial or wait)

⇒ If at least one committable message is received, then commit the transaction, else abort it.
Problem with Simple Termination Protocol

Issue 1: Operational site fails immediately after making a commit decision.

Issue 2: Site does not know the current operational status (i.e., up or down) of other sites.

Simple termination protocol is not robust:

Site 1

Crashes before sending message to Site 3

Committable

Site 2

Commits and fails before sending message to Site 3

Noncommittable

Site 3

Site 3 does not know if Site 1 was up at beginning. Does not know it got inconsistent messages

Resilient protocols require at least two rounds unless no site fails during the execution of the protocol.
## Resilient Termination Protocols

First message round:

<table>
<thead>
<tr>
<th>Type of transaction state</th>
<th>Message sent</th>
</tr>
</thead>
<tbody>
<tr>
<td>Final abort state</td>
<td>abort</td>
</tr>
<tr>
<td>Committable state</td>
<td>committable</td>
</tr>
<tr>
<td>All other states</td>
<td>non-committable</td>
</tr>
</tbody>
</table>
Resilient Termination Protocols

Second and subsequent rounds:

<table>
<thead>
<tr>
<th>Message received from previous round</th>
<th>Message sent</th>
</tr>
</thead>
<tbody>
<tr>
<td>One or more abort messages</td>
<td>abort</td>
</tr>
<tr>
<td>One or more committable messages</td>
<td>committable</td>
</tr>
<tr>
<td>All non-committable messages</td>
<td>non-committable</td>
</tr>
</tbody>
</table>

Summary of rules for sending messages.
# Resilient Termination Protocols

The transactions is terminated if:

<table>
<thead>
<tr>
<th>Condition</th>
<th>Final state</th>
</tr>
</thead>
<tbody>
<tr>
<td>Receipt of a single abort message</td>
<td>abort</td>
</tr>
<tr>
<td>Receipt of all committable messages</td>
<td>commit</td>
</tr>
<tr>
<td>2 successive rounds of messages where all messages are non-committable</td>
<td>abort</td>
</tr>
<tr>
<td>(and no site failure)</td>
<td></td>
</tr>
</tbody>
</table>

Summary of commit and termination rules.
Rules for Commit and Termination

Commit Rule:
A transaction is committed at a site only after the receipt of a round consisting entirely of committable messages.

Termination Rule:
If a site ever receives two successive rounds of non-committable messages and it detects no site failures between rounds, it can safely abort the transaction.

Lemma: \( N_i(r+1) \subseteq N_i(r) \)
Set of sites sending non-committables to site i during round r.

Lemma: If \( N_i(r+1) = N_i(r) \), then all messages received by site i during \( r \) and \( r + 1 \) were non-committable messages.
## Worst Case Execution of the Resilient Transition Protocol

<table>
<thead>
<tr>
<th>MESSAGES RECEIVED</th>
<th>SITE 1</th>
<th>SITE 2</th>
<th>SITE 3</th>
<th>SITE 4</th>
<th>SITE 5</th>
</tr>
</thead>
<tbody>
<tr>
<td>initial state</td>
<td>Commit-able</td>
<td>Non-Committable</td>
<td>Non-Committable</td>
<td>Non-Committable</td>
<td>Non-Committable</td>
</tr>
<tr>
<td>Round 1</td>
<td>(1) CNNNN</td>
<td>-NNNN</td>
<td>-NNNN</td>
<td>-NNNN</td>
<td>-NNNN</td>
</tr>
<tr>
<td>Round 2</td>
<td>FAILED</td>
<td>(1) -CNNN</td>
<td>--NNN</td>
<td>--NNN</td>
<td>--NNN</td>
</tr>
<tr>
<td>Round 3</td>
<td>FAILED</td>
<td>FAILED</td>
<td>(1) --CNN</td>
<td>--NN</td>
<td>--NN</td>
</tr>
<tr>
<td>Round 4</td>
<td>FAILED</td>
<td>FAILED</td>
<td>FAILED</td>
<td>(1) --CN</td>
<td>--CN</td>
</tr>
<tr>
<td>Round 5</td>
<td>FAILED</td>
<td>FAILED</td>
<td>FAILED</td>
<td>FAILED</td>
<td>----C</td>
</tr>
</tbody>
</table>

**NOTE:** (1) site fails after sending a single message.
Worst Case Execution of the Resilient Transition Protocol

- The second issue can lead to very subtle problems. Again, consider the scenario where Site 1 sends a committable message to Site 2 and then crashes.

- Site 2 sends out non-committable messages, receives the committable message from Site 1, commits, and then promptly fails.

- Now, Site 3 receives a single non-committable message (from Site 2). Let us assume that Site 3 was not aware that Site 1 was up at the beginning of the protocol (a reasonable assumption).

- Then, Site 3 would not suspect that messages it received were inconsistent with those received by Site 2, and it would make an inconsistent commit decision.
Recovery Protocols

- Recovery Protocols:
  - Protocols at failed site to complete all transactions outstanding at the time of failure

- Classes of failures:
  - Site failure
  - Lost messages
  - Network partitioning
  - Byzantine failures

- Effects of failures:
  - Inconsistent database
  - Transaction processing is blocked
  - Failed component unavailable
Independent Recovery

A recovering site makes a transition directly to a final state without communicating with other sites.

Lemma:
For a protocol, if a local state’s concurrency set contains both an abort and commit, it is not resilient to an arbitrary failure of a single site.

\[ s_i^{\text{cannot}} \rightarrow \text{commit} \quad \text{because other site may be in abort} \]

\[ s_i^{\text{cannot}} \rightarrow \text{abort} \quad \text{because other site may be in commit} \]

Rule 1: s: Intermediate state
If C(s) contains a commit
⇒ failure transition from s to commit
otherwise failure transition from s to abort
Theorem for Single Site Failure

**Rule 2**: For each intermediate state $s_i$:
if $t_j$ in $s(s_i)$ & $t_j$ has a failure transition to a commit (abort),
then assign a timeout transition from $s_i$ to a commit (abort).

**Theorem**: Rules 1 and 2 are sufficient for designing protocols resilient to a single site failure.

\[
\begin{align*}
\text{p: consistent} \\
\text{site 1 fails} \\
\text{s}_1 \rightarrow f_1 \Rightarrow \text{s}_1 \text{ in } s(s_2) \\
\text{p': p + Failure + Timeout Transition} \\
\text{s}_2 = f_2 \Rightarrow f_2 \in C(s_i) \\
\Rightarrow \text{s}_i \text{ in } s(s_2) \\
f_2 \leftarrow \text{inconsistent}
\end{align*}
\]
Independent Recovery when Two Sites Fail?

**Theorem:** There exists no protocol using independent recovery that is resilient to arbitrary failures by two sites.

\[ G_0 \rightarrow \text{abort} \]
\[ G_1 \]
\[ G_{k-1} \rightarrow \text{site } j \text{ recovers to abort} \]
\[ \text{(only } j \text{ makes a transition)} \]
\[ \text{other sites recover to abort} \]
\[ G_k \rightarrow \text{site } j \text{ recovers to commit} \]
\[ G_m \rightarrow \text{commit} \]

Same state exists for other sites

First global state

Failure of \( j \) ⇒ recover to commit

Failure of any other site ⇒ recover to abort

Note: \( G_0, G_1, G_2, \ldots, G_{k-1}, G_k, \ldots \)
\( G_m \) are global state vectors.
Resilient Protocol when Messages are Lost

**Theorem**: There exists no protocol resilient to a network partitioning when messages are lost.

Rule 3: Isomorphic to Rule 1:

Rule 4: undelivered message $\leftrightarrow$ timeout

Rule 2: timeout $\leftrightarrow$ failure

**Theorem**: Rules 3 & 4 are necessary and sufficient for making protocols resilient to a partition in a two-site protocol.

**Theorem**: There exists no protocol resilient to a multiple partition.
Site Failures – 3PC Termination (see book)

- **Timeout in INITIAL**
  - Who cares
- **Timeout in WAIT**
  - Unilaterally abort
- **Timeout in PRECOMMIT**
  - Participants may not be in PRE-COMMIT, but at least in READY
  - Move all the participants to PRECOMMIT state
  - Terminate by globally committing
Timeout in ABORT or COMMIT

- Just ignore and treat the transaction as completed
- participants are either in PRECOMMIT or READY state and can follow their termination protocols
Timeout in INITIAL
- Coordinator must have failed in INITIAL state
- Unilaterally abort

Timeout in READY
- Voted to commit, but does not know the coordinator's decision
- Elect a new coordinator and terminate using a special protocol

Timeout in PRECOMMIT
- Handle it the same as timeout in READY state
New coordinator can be in one of four states: WAIT, PRECOMMIT, COMMIT, ABORT

- Coordinator sends its state to all of the participants asking them to assume its state.
- Participants “back-up” and reply with appropriate messages, except those in ABORT and COMMIT states. Those in these states respond with “Ack” but stay in their states.
- Coordinator guides the participants towards termination:
  - If the new coordinator is in the WAIT state, participants can be in INITIAL, READY, ABORT or PRECOMMIT states. New coordinator globally aborts the transaction.
  - If the new coordinator is in the PRECOMMIT state, the participants can be in READY, PRECOMMIT or COMMIT states. The new coordinator will globally commit the transaction.
  - If the new coordinator is in the ABORT or COMMIT states, at the end of the first phase, the participants will have moved to that state as well.
Site Failures – 3PC Recovery (see book)

- **Failure in INITIAL**
  - start commit process upon recovery

- **Failure in WAIT**
  - the participants may have elected a new coordinator and terminated the transaction
  - the new coordinator could be in WAIT or ABORT states
  - transaction aborted
  - ask around for the fate of the transaction

- **Failure in PRECOMMIT**
  - ask around for the fate of the transaction
Site Failures – 3PC Recovery (see book)

- **Failure in COMMIT or ABORT**
  - Nothing special if all the acknowledgements have been received; otherwise the termination protocol is involved
Site Failures – 3PC Recovery (see book)

- **Failure in INITIAL**
  - unilaterally abort upon recovery

- **Failure in READY**
  - the coordinator has been informed about the local decision
  - upon recovery, ask around

- **Failure in PRECOMMIT**
  - ask around to determine how the other participants have terminated the transaction

- **Failure in COMMIT or ABORT**
  - no need to do anything

Diagram:

- Participants
  - INITIAL
    - Prepare
      - Vote-commit
    - Prepare
      - Vote-abort
  - READY
    - Global-abort
      - Ack
    - Prepared-to-commit
      - Ready-to-commit
  - ABORT
  - PRE-COMMIT
    - Global commit
      - Ack
  - COMMIT