Commit Algorithms

Hamid Al-Hamadi
CS 5204
November 17, 2009
Agenda

• Fault Tolerance

• Transactional Model

• Commit Algorithms
  • 2-Phase Commit Protocol

  • Failure and Timeout Transitions

  • 3-Phase Commit Protocol

• Summary
Fault tolerance

Causes of failure in a distributed system:
- process failure
- machine failure
- network failure

How to deal with failures:
- transparent: transparently and completely recover from all failures
- predictable: exhibit a well defined failure behavior
Transaction Model

Transaction

- A sequence of actions (typically read/write), each of which is executed at one or more sites, the combined effect of which is guaranteed to be atomic.

A transaction is said to be ATOMIC when it satisfies the ACID properties:

- **Atomicity**: either all or none of the effects of the transaction are made permanent.
- **Consistency**: the effect of concurrent transactions is equivalent to some serial execution.
- **Isolation**: transactions cannot observe each other’s partial effects.
- **Durability**: once accepted, the effects of a transaction are permanent (until changed again, of course).
Commit Algorithms

What is a Commit Algorithm?

Possible definition: Algorithm run by all nodes involved in a distributed transaction s.t.:
- Either all nodes agree to commit (transaction as a whole commits) or
- All nodes agree to Abort (transaction as a whole Aborts).

Variations:
- blocking vs. non-blocking protocols (non-failed sites must wait (can continue) while failed sites recover)
- independent recovery (failed sites can recover using only local information)
- Type of failures which can be tolerated
Commit Algorithms

Environment
Each node is assumed to have:
- data stored in a partially/full replicated manner
- stable storage (information that survives failures)
- logs (a record of the intended changes to the data: write ahead, UNDO/REDO)
- locks (to prevent access to data being used by a transaction in progress)

Generals Paradox:
- 2 Generals need to agree to attack at the same time
- Each general needs to confirm that the other general has agreed to attack.
Since message loss is possible, confirmations can get lost—need to get confirmation
Result is that the 2 generals can never agree on attacking.
Goal:

Build a commit algorithm that is correct in the presence of failure such that either all nodes involved in the distributed transaction commit or they all abort.

Topology:

• n nodes:
  • 1 Coordinator
  • (n -1) Cohorts


**2-phase Commit Protocol**

**Coordinator**

- **Commit Request** msg sent to all cohorts
- One or more cohort(s) replied abort
- Abort msg sent to all cohorts

**Cohort i (i=2,3, ..., n)**

- **Commit Request** msg received
- Agreed msg sent to Coordinator
- Commit msg received from Coordinator
- Aborted msg received from Coordinator

**Failure causes** $w_i$ to block

1. Assume ABORT if there is a timeout
2. First, writes ABORT record to stable storage.
3. First, writes COMMIT record to stable storage.
4. Write COMPLETE record when all msgs confirmed.

1. First, write UNDO/REDO logs on stable storage.
2. Writes COMPLETE record; releases locks

**Cannot recover independently**
## Site Failures

<table>
<thead>
<tr>
<th>Who Fails</th>
<th>At what point</th>
<th>Actions on recovery</th>
</tr>
</thead>
<tbody>
<tr>
<td>Coordinator</td>
<td>before writing Commit</td>
<td>Send Abort messages</td>
</tr>
<tr>
<td>Coordinator</td>
<td>after writing Commit but before writing Complete</td>
<td>Send Commit messages</td>
</tr>
<tr>
<td>Coordinator</td>
<td>after writing Complete</td>
<td>None.</td>
</tr>
<tr>
<td>Cohort</td>
<td>before writing Undo/Redo</td>
<td>None. Abort will occur.</td>
</tr>
<tr>
<td>Cohort</td>
<td>after writing Undo/Redo</td>
<td>Wait for message from Coordinator.</td>
</tr>
</tbody>
</table>
Definitions

Synchronous
A protocol is synchronous if any two sites can never differ by more than one transition.

Concurrency Set
For a given state, s, at one site the concurrency set, C(s), is the set of all states in which all other sites can be.

Coordination

C(w1) = \{q2, w2, a2\}
Sender set
For a given state, s, at one site, the sender set, S(s), is the set of all other sites that can send messages that will be received in state s.

What causes blocking
Blocking occurs when a site’s state, s, has a concurrency set, C(s), that contains both commit and abort states.
Commit Algorithms

Blocking of 2-phase Commit Protocol

Coordinator

Commit_Request msg sent to all cohorts

$q_1$

One or more cohort(s) replied abort

Abort msg sent to all cohorts

$W_1$

All cohorts agreed

Send Commit msg to all cohorts

$c_1$

$a_1$

Cohort i (i=2,3, ..., n)

Commit_Request msg received

$q_i$

Agreed msg sent to Coordinator

$W_i$

Abort msg sent to Coordinator

$A_i$

Commit msg received from Coordinator

$c_i$

Commit_Request msg received

Solution:

Introduce additional states -> additional messages (to allow transitions to/from these new states). -> adding at least one more “phase”.

1. First, write UNDO/REDO logs on stable storage.

2. Writes COMPLETE record; releases locks

1. Assume ABORT if there is a timeout

2. First, writes ABORT record to stable storage.

3. First, writes COMMIT record to stable storage.

4. Write COMPLETE record when all msgs confirmed.
Added prepare states

Coordinator

\( q_1 \)
- Commit_Request msg sent to all cohorts

\( w_1 \)
- All cohorts agreed
- Send Prepare msg to all cohorts

\( a_1 \)
- All cohorts sent
- Ack msg
- Send Commit msg to all cohorts

\( p_1 \)

Cohort \( i \) (\( i=2,3, \ldots, n \))

\( q_i \)
- Commit_Request msg received
- Agreed msg sent to Coordinator
- Abort msg sent to Coordinator

\( w_i \)
- Prepare msg received
- Send Ack msg to Coordinator

\( a_i \)
- Abort msg received from Coordinator

\( p_i \)

\( c_i \)
- Commit msg received from Coordinator
Failure and Timeout Transitions

Failure Transition Rule

For every nonfinal state $s$, if $C(s)$ contains a commit, then add failure transition to a commit state; otherwise, add failure transition from $s$ to an abort state.
Adding a Failure Transition

Coordinator

Commit_Request msg sent to all cohorts

One or more cohort(s) replied abort
Abort msg sent to all cohorts

Commit_Request msg received

All cohorts agreed
Send Prepare msg to all cohorts

All cohorts sent Ack msg
Send Commit msg to all cohorts

Cohort i (i=2,3, ..., n)

Commit_Request msg received

Agreed msg sent to Coordinator

Abort msg sent to Coordinator

F

Prepare msg received from Coordinator

Abort msg sent to Coordinator

Commit_Request msg received

Cohort i

Cohort i

Commit msg received from Coordinator

Commit msg received

Send Prepare msg to all cohorts

Send Ack msg to Coordinator

Send Commit msg to all cohorts
Timeout Transition Rule

For every nonfinal state $s$, if $j$ is in $S(s)$ and $j$ has failure transition to commit (abort) state then add timeout transition from $s$ to commit (abort) state
Adding a Timeout Transition

**Coordinator**

- **q₁**
  - Commit_Request msg sent to all cohorts
  - One or more cohort(s) replied abort
  - Abort msg sent to all cohorts
  - F

- **w₁**
  - All cohorts agreed
  - Send Prepare msg to all cohorts

- **a₁**
  - Send Ack msg to Coordinator

- **p₁**
  - All cohorts sent Ack msg
  - Send Commit msg to all cohorts

- **c₁**
  - Commit msg received from Coordinator

**Cohort i (i=2,3, …, n)**

- **qᵢ**
  - Commit_Request msg received
  - Agreed msg sent to Coordinator

- **wᵢ**
  - Prepare msg received from Coordinator
  - Abort msg sent to Coordinator

- **aᵢ**
  - Commit_Request msg to Coordinator

- **pᵢ**
  - Send Ack msg to Coordinator

- **cᵢ**
  - Commit msg received from Coordinator
Adding a prepared state, and using Failure and Timeout transmissions in the 3PC protocol allows the protocol to be resilient to a **single site** failure.

After adding all transitions we get:
3-Phase Commit Protocol

Coordinator:

- **q1**: Commit_Request msg sent to all cohorts
  - F,T
- **w1**: Send Prepare msg to all cohorts
  - F,T
- **a1**: Abort msg sent to all cohorts
  - T
- **p1**: All cohorts agreed
  - Send Ack msg to Coordinator
  - F
- **c1**: All cohorts sent Ack msg
  - Send Commit msg to all cohorts
  - F

One or more cohort(s) replied abort
Abort msg sent to all cohorts

Cohort i (i=2,3, ..., n):

- **qi**: Commit_Request msg received
  - Agreed msg sent to Coordinator
  - F,T
- **wi**: Send Prepare msg received from Coordinator
  - F,T
- **ai**: Abort msg received from Coordinator
  - F,T
- **pi**: Abort msg received from Coordinator
  - F,T
- **ci**: Commit msg received from Coordinator
  - F,T

Transition:

- **T**: Timeout Transition
- **F**: Failure Transition
- **F,T**: Failure/Timeout Transition
Commit Algorithms

Summary

- Commit Algorithms are used to commit distributed transactions across multiple nodes S.T either all nodes commit or all abort.

- Commit algorithms differ in aspects of blocking, independent recovery, and types of failures which can be tolerated.

- 2-phase commit algorithm suffers from blocking and lacks independent recovery.

- 3-phase commit algorithm uses prepared states and applies transition rules, this gives it the properties of:
  - Non-blocking
  - Can recovery independently (→ only resilient to a single site failure).
Questions?