CS 347: Parallel and Distributed Data Management

Notes06: Reliable Distributed Database Management
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)
  - See: “Ten of the strangest data center outages”; http://goo.gl/DcQysr
- Expensive to protect failures (e.g., earthquake)
- Failures we know how to protect against (e.g., message sequence numbers; stable storage)
Failure model:

- Desired
- Expected
- Unexpected

Events

Desired

Undesired

Unexpected
Node models

(1) Fail-stop nodes

- perfect
- halted
- recovery
- perfect

Volatile memory lost
Stable storage ok
Node models

(2) Byzantine nodes

A

Perfect

Arbitrary failure

Recovery

B

Perfect

Arbitrary failure

Recovery

C

Perfect

Arbitrary failure

Recovery

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 $T_D$

If no ack in $T_D$ sec. $\rightarrow$ 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
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

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
No Data Replication

- Reliable network, fail-stop nodes
- Basic idea: node $P\alpha$ controls $X$
  - Single control point simplifies concurrency control, recovery
  - Not an availability hit:
    if $P\alpha$ down, $X$ unavailable too!
"$P_\alpha$ 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

USER

T1
Local DBMS

T2
Local DBMS

T3
Local DBMS
Process models

(B) Transaction servers (manager)
• Cohorts: application code responsible for remote access
• Transaction manager: “system” handles distribution, remote access
Distributed commit problem

Transaction T

Action: a₁, a₂

Action: a₃

Action: a₄, a₅
Centralized two-phase commit

Coordinator

Participant
• 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>$T_1$</th>
<th>$X$</th>
<th>undo/redo info</th>
<th>...</th>
<th>$T_1$</th>
<th>$Y$</th>
<th>info</th>
<th>...</th>
<th>$T_1$</th>
<th>&quot;W&quot;</th>
<th>state</th>
</tr>
</thead>
</table>

At recovery:

• $T_1$ 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 $\Rightarrow$ abort $T_1$
- See “C” record on log $\Rightarrow$ finish $T_1$
• Add timeouts to cope with messages lost during crashes
• Add finish ("F") state for coordinator – all done, can forget outcome
Coordinator

I

\(\text{go}\) exec*

\(\text{nok}\) abort*

W

\(\text{ok}\) commit*

\(\text{c-ok}\)*

C

F

A

\(\text{nok}\)*

\(\text{t=timeout}\)

\(\text{cping=coord. ping}\)
Participant

I

exec
ok

W

exec	nok

abort
nok

A

commit
c-ok

C
Participant

I

exec
ok

exec
nok

W

cping
done

commit
c-ok

C

cping
done

A

"done" message counts as either c-ok or n-ok for coordinator
Participant

I
exec
ok
exec
nok

W
commit
c-ok

C

cping
done
cping
t

cping
done

A
abort
nok

cping
done

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

equivalent to finish state
Presumed abort protocol

- “F” and “A” states combined in coordinator
- Saves persistent space (forget quicker)
- Presumed commit is analogous
Presumed abort-coordinator (participant unchanged)
Remember: all state transitions must be logged

Example: tracking who has sent “OK” msgs

Log at coord:

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

- 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

×

P1

×

P2

×

P3

W

P4

W
Case I: 
\[ \text{P}_1 \rightarrow \text{“W”}; \text{ coordinator sent commits} \]
\[ \text{P}_1 \rightarrow \text{“C”} \]

Case II: 
\[ \text{P}_1 \rightarrow \text{NOK}; \text{ P}_1 \rightarrow \text{A} \]
\[ \Rightarrow \text{ P}_2, \text{ P}_3, \text{ P}_4 \text{ (surviving participants)} \]

\[ \text{cannot safely abort or commit transaction} \]
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
 Coordinator

3PC

Participant

** means 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...
• Let \( \{S_1, S_2, \ldots, S_n\} \) be survivor sites
• If one or more \( S_i = \text{COMMIT} \) ⇒ COMMIT \( T \)
• If one or more \( S_i = \text{ABORT} \) ⇒ ABORT \( T \)
• If one or more \( S_i = \text{PREPARE} \) ⇒
  \( T \) could not have aborted ⇒ COMMIT \( T \)
• If no \( S_i = \text{PREPARE} \) (or COMMIT) ⇒
  \( T \) could not have committed ⇒ ABORT \( T \)
Example:

? × ○ ○ P

? × ○ ○ W

○ W
Example:

? ☒ ☐ I

? ☒ ☐ W

? ☒ ☐ W
Example:
Example:

? × 〇 P

? × 〇 W

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

 Decide to commit

 Time 1
 Time 2
 Time 3
 Time 4
Note: when survivors continue 3PC, failed nodes do not count

E.g.,

```
ack** ≡ when ack’s received from all non-failed nodes
```

Diagram:

```
P
  ↓
  ↘
  | ↘
  | ↘
  commit *
     ↓
     C
```
**Note:** 3PC unsafe with partitions!

Diagram showing the state of transactions in a distributed system, illustrating the issues with 3PC (Three-Phase Commit) protocol in the presence of partitions. The diagram highlights the potential for transactions to be in inconsistent states due to network partitions. The diagram includes arrows labeled 'abort' and 'commit' indicating the possible outcomes when transactions fail to communicate properly.
Node recovery:

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

? ×  ○ W

P ×  ○ W

○ W → A
Node recovery:

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

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

- Want a “gang” of failed but recovered nodes to be able to terminate a transaction even when rest are failed...
Option B

- Nodes are assigned votes, total is $V$
  Majority is $\frac{V+1}{2}$ e.g., \[
  \begin{cases}
    V=5 \\
    \text{Maj}=3 \\
    V=6 \\
    \text{Maj}=4
  \end{cases}
\]

- To make state transitions, coordinator requires messages from nodes with a majority of votes
Example(1):  Coord  ☒  ?  P_2 → W  
            P_1  ☒  ?  P_3 → W  
                  ?  p_4 → 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 $\times$ $?\ P_2 \rightarrow W$

$P_1 \times ?\ P_3 \rightarrow W$

$?\ p_4 \rightarrow W$

- 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 $\times$  $?$ $P_3 \rightarrow "P"$

$P_1$ $\times$ $?$ $P_4 \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 ⊗ ⊗ ⊗ ⊗ P₃ → "P"

P₁ ⊗ ⊗ ⊗ ⊗ P₄ → "W"

P₂ ⊗ ⊗

• Each node has 1 vote; V=5, Maj=3
• Nodes fail after entering states shown; P₃, P₄ recover
• Termination rule says we can try to commit, but P₃, P₄ do not have enough votes, so they do nothing!
• P₃, P₄ doing nothing is good because later on, coord. P₁, P₂ 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:

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

\[ P \quad \times \quad \bigcirc \quad \bigcirc \quad W \]

\[ \bigcirc \quad W \quad \rightarrow \quad A \]
Important Detail for Majority 3PC

• Example:
Need “Prepare To Abort” State

** means participants with majority votes.

Coordinator

I

W

PC

C

Participant

I

W

PC

C

A

PA

PA

Notes06
Example Revisited

?  W

PC  W

W  → PA
Example Revisited

OK to commit since transaction could not have aborted

? W → PC → C

PC ✔ W

W → PA
Example Revisited -II

\[ \begin{align*}
? & \quad \lor \quad W \\
PC & \quad \lor \quad W \rightarrow PA \\
& \quad \lor \quad W \rightarrow PA
\end{align*} \]
Example Revisited -II

No decision:
Transaction could have aborted or could have committed... Block!

PC

W ➔ PA

W ➔ PA
3PC with Majority Voting

- If survivors have majority and all states $W \Rightarrow$ try to abort
- If survivors have majority and states in $\{W, \text{PC}, \text{C}\} \Rightarrow$ try to commit
- If survivors have majority and states in $\{W, \text{PA}, \text{A}\} \Rightarrow$ 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 ⇒ redo if necessary
  – if abort found (or no “W” record) ⇒ 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

e.g., central detection node
Problem: False deadlocks

Time 1

info sent

T1
T2

T1
T2

Time 2

T1
T2

info sent

Time 3

at central site:

T1
T2
Problem: **False deadlocks**

1. Time 1
   - Information sent
   - T1
   - T2

2. Time 2
   - T1
   - T2
   - Information sent

3. Time 3
   - at central site:
   - T1
   - T2
   - Note that T2 is not 2PL; it releases lock, then asks for another lock
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