Distributed Databases

CS347
Lecture 16
June 6, 2001
Topics for the day

• Reliability
  – Three-phase commit (3PC)
  – Majority 3PC

• Network partitions
  – Committing with partitions
  – Concurrency control with partitions
Recall - 2PC is blocking

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

Case II: P₁ → NOK; P₁ → A

⇒ P₂, P₃, P₄ (surviving participants) cannot safely abort
  or commit transaction
3PC (non-blocking commit)

- Assume: failed node is down forever
- Key idea: before committing, coordinator tells participants everyone is ok

Diagram:

Coordinator (C) → Participant (P) → Coordinator (C)

Coordinator sends:
- go
- exec
- ok
- pre
- ack
- commit

Participant sends:
- nok
- abort

Coordinator: I
Participant: W
Coordinator: P
Participant: A
3PC recovery (termination protocol)

- Survivors try to complete transaction, based on their current states

- **Goal:**
  - If dead nodes committed or aborted, then survivors should not contradict!
  - Otherwise, survivors can do as they please...
**Termination rules**

- Let \( \{S_1, S_2, \ldots, S_n\} \) be survivor sites. Make decision on commit/abort based on following rules:
  
  - If one or more \( S_i = \text{COMMIT} \) \( \Rightarrow \) COMMIT \( T \)
  - If one or more \( S_i = \text{ABORT} \) \( \Rightarrow \) ABORT \( T \)
  - If one or more \( S_i = \text{PREPARE} \) \( \Rightarrow \) COMMIT \( T \)
    
    (\( T \) could not have aborted)
  - If no \( S_i = \text{PREPARE} \) (or COMMIT) \( \Rightarrow \) ABORT \( T \)
    
    (\( T \) could not have committed)
Examples

- ? X  O P  O W  O W
- ? X  O I  O W  O W
- ? X  O C  O P  O P
Points to Note

• Once survivors make a decision, they must **elect** a new coordinator and **continue** with 3PC.

\[
\begin{array}{cccc}
W & P & C & C \\
W & P & C & C \\
P & P & P & C \\
\end{array}
\]

Decide to commit

• When survivors continue 3PC, failed nodes do not count.
  – Example: OK* = OK from every non-failed participant
Points to Note

- 3PC unsafe with network partitions

Diagram:

W → abort
P → commit
Node recovery

- After node N recovers from failure, what must it do?
  - N must not participate in termination protocol
  - Wait until it hears commit/abort decision from operational nodes

Later on...
All-failed problem

Waiting for commit/abort decision is fine, unless all nodes fail.

Two possible solutions:

- Option A: Recovering node waits for either
  - commit/abort outcome for T from some other node.
  - all nodes that participated in T are up and running.

  Then 3PC can continue

- Option B: Use Majority 3PC
Majority 3PC

- Nodes are assigned votes. Total votes = V. For majority, need \( \lceil (V+1)/2 \rceil \) votes.

- **Majority rule**: For every state transition, coordinator requires messages from nodes with a majority of votes.

- Majority rule ensures that any decision (preparing, committing) is known to a future decision-making group.

![Diagram of decision sets and overlaps]

- Decision #1
  - Node 1
  - Node 1 & 2
  - Node 2

- Decision #2
  - Node 2
  - Node 2

decision # 1

decision # 2
Example 1

Each node has 1 vote, V=5
Nodes P2, P3, P4 enter “W” state and fail
When they recover, coordinator and P1 are down
Since P2, P3, P4 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

- Each node has 1 vote, V=5
- Nodes fail after entering states shown. P3 and P4 recover.
- Termination rule says \{P3,P4\} can commit. But \{P3,P4\} do not have majority – so block.
- Right thing to do, since \{Coordinator,P1,P2\} may later abort.
Problem!

- Previously, we disallowed recovering nodes from participating.
- Now any set of nodes with majority can progress.
- How do we fix the problem below?

Diagram:

- Question mark
- Node W
- Node P
- Node C
- Node P marked with a checkmark
- Node W marked with an X
- Node A marked with an X
Majority 3PC (introduce “prepare to abort” state)
Example Revisited

OK to commit since transaction could not have aborted

?- W \(\rightarrow\) PC \(\rightarrow\) C

PC - W \(\rightarrow\) PA
Example Revisited -II

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

[Diagram with arrows and symbols indicating the possible outcomes of a transaction process.]
Majority 3PC Rules

- If survivors have majority and states in \( \{W, PC, C\} \Rightarrow \text{try to commit} \)

- If survivors have majority and states in \( \{W, PA, A\} \Rightarrow \text{try to abort} \)

- Otherwise block

Blocking Protocol !!
Summarizing commit protocols

- **2PC**
  - Blocking protocol
  - Key: coordinator does not move to “C” state unless every participant is in “W” state

- **3PC**
  - Non-blocking protocol
  - Key: coordinator broadcasts that “all are ok” before committing. Failed nodes must wait.
  - Any set of non-failed nodes can terminate transaction (even a single node)
  - If all nodes fail, must wait for all to recover
Summarizing commit protocols

• Majority 3PC
  – Blocking protocol
  – Key: Every state transition requires majority of votes
  – Any majority group of active+recovered nodes can terminate transaction
Network partitions

• Groups of nodes may be isolated or may be slow in responding

• When are partitions of interest?
  – True network partitions (disaster)
  – Single node failure cannot be distinguished from partition (e.g., NIC fails)
  – Loosely-connected networks
    • Phone-in, wireless
Problems

- Partitions during commit

- Updates to replicated data in isolated partitions

T1

update

X

X

update

T2

X

X

X

A

A
Quorums

• **Commit and Abort Quorums:** Given set $S$ of nodes, define
  - Commit quorum $C \subseteq 2^S$, Abort quorum $A \subseteq 2^S$
  - $X \cap Y \neq \emptyset \ \forall \ X, Y$ such that $X \in C$ and $Y \in A$

• Example: $S = \{a,b,c,d\}$
  - $C = \{\{a,b,c\}, \{a,b,d\}, \{a,c,d\}, \{b,c,d\}\}$
  - $A = \{\{a,b\}, \{a,c\}, \{a,d\}, \{b,c\}, \{b,d\}, \{c,d\}\}$

• Quorums can be implemented with vote assignments
  - $V_a = V_b = V_c = V_d = 1$
  - To commit $\geq 3$ votes
  - To abort $\geq 2$ votes
Quorums

- However, not all quorums can be implemented with votes
  
  \[ C = \{\{a,b\}, \{c,d\}\} \quad A = \{\{a,c\}, \{a,d\}, \{b,c\}, \{b,d\}\} \]

- Commit protocol must enforce quorum

- Quorum condition is in **addition** to whatever rules the commit protocol might have

- If node knows transaction could have committed (aborted), if cannot abort (commit) even if abort (commit) quorum available

- With network partitions, all commit protocols are **blocking**.
3PC Example

- To make commit decision: commit quorum 
  (votes for commit \( V_C = 3 \))
- To make abort decision: abort quorum 
  (votes for abort \( V_A = 3 \))

Old coordinator could not have committed since all other nodes are in “W” state.

Surviving nodes have abort quorum

Attempt to abort
Another 3PC Example

\[ VC = 3; VA = 3 \]

1. P
2. w
3. w

- Old coordinator could not have aborted since one node is in “P” state.
- Surviving nodes have commit quorum

Note: When using 3PC with quorums, we must use the “Prepare to Abort” (PA) state as in majority commit (for the same reasons).
Yet Another 3PC Example

• Old coordinator could not have aborted since one node is in “P” state.
• However, surviving nodes do not have commit quorum

\[ V_c = 3; V_a = 3 \]

Block
Partitions and data replication

Options:
1. All copies required for updates
2. At most one group may update, at any time
3. Any group may update (potentially more than one can update simultaneously)
Coteries

• Used to enforce updates by at most one group
• Given a set $S$ of nodes at which an element $X$ is replicated, define a coterie $C$ such that
  ▪ $C \subseteq 2^S$
  ▪ $A_1 \cap A_2 \neq \emptyset$, for $\forall A_1, A_2 \in C$

Examples:

$C_1 = \{\{a,b,c\}, \{a,b,d\}, \{a,c,d\}, \{b,c,d\}\}$

$C_2 = \{\{a,b\}, \{a,c\}, \{a,d\}, \{b,c,d\}\}$

$C_3 = \{\{a,b\}, \{c,d\}\}$ not a valid coterie
Accessing Replicated Elements

- Element X replicated at a set $S$ of sites.

- Specify two sets $R$ (for “read”) and $W$ (for “write”) with the following properties:
  - $R, W \subseteq 2^S$
  - $W$ is a coterie over $S$
  - $R$ and $W$ are read and write quorums respectively over $S$
    i.e., $A \cap B \neq \emptyset \ \forall A,B$ such that $A \in R$ and $B \in W$

  (similar to commit and abort quorums)
Accessing replicated elements

- X replicated at $S = \{a,b,c,d\}$

- Example 1:
  \[W = \{\{a,b,c\}, \{a,b,d\}, \{a,c,d\}, \{b,c,d\}\}\]
  \[R = \{\{a,b\}, \{a,c\}, \{a,d\}, \{b,c\}, \{b,d\}, \{c,d\}\}\]

- Example 2:
  \[R = W = \{\{a,b\}, \{a,c\}, \{a,d\}, \{b,c,d\}\}\]

- Can be implemented using vote assignments. For example 1:
  - $V_a = V_b = V_c = V_d = 1$; Total = 4
  - $2V_w > T$
  - To write, get 3 votes ($V_w$)
  - $V_w + V_r > T$
  - To read, get 2 votes ($V_r$)
Missing Writes

Example: a 3 node system, 1 vote for each node

T₁ • a

T₁ • b

T₁ commits at \{a,b\}.

Partition changes. T₂ comes along. Verifies read and write quorum \{a,c\}.

T₂ reads at c. Writes and commits at \{a,c\}.

Unserializable
Solution

- Each node maintains list of committed transactions
- Compare list at read site with those at write sites
- Update sites that missed transactions

\[ \text{Exec-list} = \{T_0\} \]

\[ \text{Not OK!} \]
- Details are tricky
- Maintaining list of updates until \textbf{all} nodes have seen them  
  - interesting problem
- See resource ("Missing Writes" algorithm) for details
Partitions and data replication

Options:
1. All copies required for updates
2. At most one group may update, at any time
3. Any group may update

Separate Operational Groups

\[
\text{DB}_0 \sim \text{DB}_1 \sim \text{DB}_2 \sim \text{DB}_3 \sim \text{DB}_4
\]
Integrating Diverged DBs

1. Compensate transactions to make schedules equivalent
2. Data-patch: semantic fix
Compensation Example

- Assume $T_1$ commutes with $T_3$ and $T_4$ (for example, no conflicting operations)
- Also assume that it is possible to come up with $T_2^{-1}$ to undo the effect of $T_2$ on the database.
In general: Based on the characteristics of transactions, can "merge" schedules
Data Patch Example

- Forget schedules
- Integrate differing values via human-supplied “rules”
For X: site 1 wins
For Y: latest timestamp wins
For Z: add increments

Rules

for Z: 7 = 5 + site 1 increment + site 2 increment
      = 5 + 1 + 1
Resources

- “Concurrency Control and Recovery” by Bernstein, Hardzilacos, and Goodman
  - Available at
    http://research.microsoft.com/pubs/ccontrol/