Network Partitions

Causes
- Wired network disruptions
  - E.g., due to natural disaster
- Wireless network obstacles
- (Semi-)autonomous nodes
- Single node failures indistinguishable from partitions
  - E.g., network card fails

Partitions without Replication

If some data is unavailable then stuck
Even if all data is available, must cope with partition during commit protocol
**Quorums**

\[ C_1 = \{ \{ a, b, c \}, \{ a, b, d \}, \{ a, c, d \}, \{ b, c, d \} \} \]

\[ A_1 = \{ \{ a, b \}, \{ a, c \}, \{ a, d \}, \{ b, c \}, \{ b, d \}, \{ c, d \} \} \]

**Important property**
\[ X \in C \implies \forall Y \in A: X \cap Y \neq \emptyset \]
\[ Y \in A \implies \forall X \in C: X \cap Y \neq \emptyset \]

**Quorums**

Vote assignments

One way to implement quorums

\[ V_C + V_A > V_T \]

\[ V_C \text{ votes to commit} \]

\[ V_A \text{ votes to abort} \]

\[ V_T \text{ total votes} \]

\[ \text{to commit } V_C \geq 3 \]

\[ \text{to abort } V_A \geq 2 \]

Commit protocols must enforce quorum

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

All commit protocols are **blocking** (with network partitions)

**Quorums**

3PC with quorum

To make commit decision: commit quorum
To make abort decision: abort quorum

**Example 1**

Votes for commit \( V_C = 3 \)

Votes for abort \( V_A = 3 \)

bold
Quorums

Example 1

Coordinator could not have committed
Have abort quorum
→ Try to abort

Quorums

Example 2

Coordinator could not have aborted
Have commit quorum
→ Try to commit
**Quorums**

**Example 3**

Insufficient votes → Block

**Quorums**

**Problematic state**

Possible?
Could the transaction have aborted?
Could the transaction have committed?
What to do next?

Could the transaction have aborted?
Could the transaction have committed?
What to do next? Block

**Quorums**

**Scenario 1**

\{(PC, PA, W) after commit\}

\[\begin{array}{cccccc}
\circ \ W & PC & ? & ? & PC & ? \\
\circ \ W & W & W & W & W & PC & C \\
\circ \ W & W & W & W & W & PC & PC \\
\circ \ W & W & W & PA & ? & ? & PA \\
\circ \ W & W & W & W & PA & ? & ? & W \\
\end{array}\]
Quorums

Scenario 2
\{PC, PA, W\} after abort

Options
1. All copies are required for updates
2. Groups may update, but at most one at a time
3. Any group may update

Not all quorums can be implemented via votes

\[ C_2 = \{\{a, b\}, \{c, d\}\} \]
\[ A_2 = \{\{a, c\}, \{a, d\}, \{b, c\}, \{b, d\}\} \]
Coteries

Key idea
When operational groups change, at least one node should be shared across previous group and current group, so that it can carry over state.

\[ C_1 = \{ \{ a, b, c \}, \{ a, b, d \}, \{ a, c, d \}, \{ b, c, d \} \} \]
\[ C_2 = \{ \{ a, b \}, \{ a, c \}, \{ a, d \}, \{ b, c, d \} \} \]

Important property
\[ X \in C \Rightarrow \forall Y \in C: X \cap Y \neq \emptyset \]

Reading replicated data
Can relax coterie requirement

\[ C_1 = \{ \{ a, b, c \}, \{ a, b, d \}, \{ a, c, d \}, \{ b, c, d \} \} \]
\[ R_1 = \{ \{ a, b \}, \{ a, c \}, \{ a, d \}, \{ b, c \}, \{ b, d \}, \{ c, d \} \} \]
\[ C_2 = R_2 = \{ \{ a, b \}, \{ a, c \}, \{ a, d \}, \{ b, c, d \} \} \]

Votes
2 \( V_W \) \( > \) \( V_T \), \( V_W + V_R > V_T \)

To write get 3 votes (\( V_W \))
To read get 2 votes (\( V_R \))
Coteries

Votes
2 \( V_W > V_T \), \( V_W + V_R > V_T \)

To write get 3 votes (\( V_W \))
To read get 2 votes (\( V_R \))

\[ C_2 = R_2 = \{ \{ a, b \}, \{ a, c \}, \{ a, d \}, \{ b, c, d \} \} \]

Coteries

Which one is better?

\[ C_1 = \{ \{ a, b, c \}, \{ a, b, d \}, \{ a, c, d \}, \{ b, c, d \} \} \]
\[ R_1 = \{ \{ a, b \}, \{ a, c \}, \{ a, d \}, \{ b, c \}, \{ b, d \}, \{ c, d \} \} \]
\[ C_2 = R_2 = \{ \{ a, b \}, \{ a, c \}, \{ a, d \}, \{ b, c, d \} \} \]

Coteries

Not all coteries have vote assignments

Example
Nodes \{ a, b, c, d, e, f \}

\[ C = \{ \{ a, b \}, \{ a, c, d \}, \{ a, c, e \}, \{ a, d, f \}, \{ a, e, f \}, \{ b, c, f \}, \{ b, d, e \} \} \]
Update Propagation

Problem
Example

Now

\( T_1 \) is committed at \( a, b \)

\( T_1 \) reads at \( c \) (not seeing \( T_1 \))

Then writes and commits at \( a, c \)

Solution

Each node keeps list of committed transactions

Compare list at read site with those at write sites

Update sites that missed transactions

Example revisited
Update Propagation
Solution
Example revisited

Each node must keep updates for transactions until all nodes have seen them.
Multiple Operational Groups

Integration options
1. Compensate transactions to make schedules match
2. Data patch (i.e., apply semantic fix)

Compensation Example
Assume $T_1$ commutes with $T_3$ and $T_4$
E.g., no conflicting operations

Schedule $T_0, T_3, T_4$, $T_i$ is equivalent to $T_0, T_1, T_3, T_4$

Schedule $T_0, T_1, T_2, T_2^+, T_3, T_4$, $T_i$ is equivalent to $T_0, T_1, T_2, T_4$
Multiple Operational Groups

Compensation

In general, based on characteristics of transactions, schedules can be merged.

Data patch

Forget about schedules
Integrate differing values via rules

E.g., simple rules
For $x$ site 1 wins
For $y$ latest timestamp wins
For $z$ add increments
Multiple Operational Groups

Data patch

Summary

Partitions without replication
Abort, commit quorums
3PC with quorums

Partitions with replication
At most one operational group
Coteries
Update propagation
Multiple operational groups