Recap

- Consistency
  - Linearizability?
  - Sequential consistency?
- Chain replication
- Primary-backup (passive) replication
- Active replication

Linearizability vs. Sequential Consistency

- Both care about giving an illusion of a single copy.
  - From the outside observer, the system should (almost) behave as if there's only a single copy.
- Linearizability cares about time.
  - Steve writes on his facebook wall at 11am.
  - Atri writes on his facebook wall at 11:05am.
  - Everyone will see the posts in that order.
- Sequential consistency cares about program order.
  - Steve writes on his facebook wall at 11am.
  - Atri writes on his facebook wall at 11:05am.
  - It's not necessarily that the posts will be ordered that way (though everyone will see the same order).

Two More Consistency Models

- Even more relaxed
  - We don't even care about providing an illusion of a single copy.
- Causal consistency
  - We care about ordering causally related write operations correctly.
- Eventual consistency
  - As long as we can say all replicas converge to the same copy eventually, we're fine.

Causal Consistency

- Writes that are potentially causally related must be seen by all processes in the same order. Concurrent writes may be seen in a different order on different machines.
  - Weaker than sequential consistency
- How do we define “causal relations” between two writes?
  - (Roughly) One client reads something that another client has written; then the client writes something.

Example 1:

<table>
<thead>
<tr>
<th>Client</th>
<th>Action</th>
<th>Time</th>
</tr>
</thead>
<tbody>
<tr>
<td>P1</td>
<td>W(x)1</td>
<td>11am</td>
</tr>
<tr>
<td>P2</td>
<td>R(x)1</td>
<td>11am</td>
</tr>
<tr>
<td>P3</td>
<td>R(x)1</td>
<td>11am</td>
</tr>
<tr>
<td>P4</td>
<td>R(x)1</td>
<td>11am</td>
</tr>
</tbody>
</table>

Causally related

Concurrent writes

W(x)1  W(x)2  W(x)3

This sequence obeys causal consistency
Causal Consistency Example 2

- Causally consistent?
  - No!

P1: \( W(x) \)
P2: \( R(x) \)
P3: \( W(x) \)
P4: \( R(x) \)

Causally related

Causal Consistency Example 3

- Causally consistent?
  - Yes!

P1: \( W(x) \)
P2: \( R(x) \)
P3: \( W(x) \)
P4: \( R(x) \)

Eventual Consistency

- Popularized by the CAP theorem.
- The main problem is network partitions.

![Network partition diagram]

Dilemma

- In the presence of a network partition:
  - In order to keep the replicas consistent, you need to block.
    - From the outside observer, the system appears to be unavailable.
  - If we still serve the requests from two partitions, then the replicas will diverge.
    - The system is available, but no consistency.
- The CAP theorem explains this dilemma.

CAP Theorem

- Consistency
- Availability
  - Respond with a reasonable delay
- Partition tolerance
  - Even if the network gets partitioned
- In the presence of a partition, which one to choose? Consistency or availability?
- Brewer conjectured in 2000, then proven by Gilbert and Lynch in 2002.

Coping with CAP

- The main issue is the Internet.
  - As the system grows to span geographically distributed areas, network partitioning becomes inevitable.
- Then the choice is either giving up availability or consistency
- A design choice: What makes more sense to your scenario?
- Giving up availability and retaining consistency
  - E.g., use 2PC
    - Your system blocks until everything becomes consistent.
- Giving up consistency and retaining availability
  - Eventual consistency
Dealing with Network Partitions

- During a partition, pairs of conflicting transactions may have been allowed to execute in different partitions. The only choice is to take corrective action after the network has recovered.
  - Assumption: Partitions heal eventually
- Abort one of the transactions after the partition has healed
- Basic idea: allow operations to continue in one or some of the partitions, but reconcile the differences later after partitions have healed

Quorum Approaches

- Quorum approaches used to decide whether reads and writes are allowed
- There are two types: pessimistic quorums and optimistic quorums
- In the pessimistic quorum philosophy, updates are allowed only in a partition that has the majority of RMs
  - Updates are then propagated to the other RMs when the partition is repaired.

Static Quorums

- The decision about how many RMs should be involved in an operation on replicated data is called Quorum selection
- Quorum rules state that:
  - At least $r$ replicas must be accessed for read
  - At least $w$ replicas must be accessed for write
  - $r + w > N$, where $N$ is the number of replicas
  - $w > N/2$
  - Each object has a version number or a consistent timestamp

- What does $r + w > N$ mean?
  - The only way to satisfy this condition is that there’s always an overlap between the reader set and the write set.
  - There’s always some replica that has the most recent write.
- What does $w > N/2$ mean?
  - When there’s a network partition, only the partition with more than half of the RMs can perform write operations.
  - The rest will just serve reads with stale data.
- $R$ and $W$ are tunable:
  - E.g., $N=3, r=1, w=3$: High read throughput, perhaps at the cost of write throughput.

Optimistic Quorum Approaches

- An Optimistic Quorum selection allows writes to proceed in any partition.
  - “Write, but don’t commit”
    - Unless the partition gets healed in time.
  - Resolve write-write conflicts after the partition heals.
- Optimistic Quorum is practical when:
  - Conflicting updates are rare
  - Conflicts are always detectable
  - Damage from conflicts can be easily confined
  - Repair of damaged data is possible or an update can be discarded without consequences
  - Partitions are relatively short-lived
**View-based Quorum**

- An optimistic approach
- Quorum is based on views at any time
  - Uses group communication as a building block
- We define thresholds for each of read and write:
  - \( W \): regular writer quorum
  - \( R \): regular reader quorum
  - \( A_w \): minimum nodes in a view for write, e.g., \( A_w > N/4 \)
  - \( A_r \): minimum nodes in a view for read
    - E.g., \( A_w + A_r > N/2 \)
- Protocol
  - Try regular quorum first; if it doesn’t work, change the view.
    - If the minimum is satisfied, then proceed.
    - \( A_w \) \& \( A_r \) effectively determine which partition can proceed.

**Example: View-based Quorum**

- Consider: \( N = 5 \), \( w = 5 \), \( r = 1 \), \( A_w = 3 \), \( A_r = 1 \)

<table>
<thead>
<tr>
<th></th>
<th>V1.0</th>
<th>V2.0</th>
<th>V3.0</th>
<th>V4.0</th>
<th>V5.0</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>2</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>3</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>4</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>5</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

Initially all nodes are in V1.0

Network is partitioned

Read is initiated, quorum is reached

Write is initiated, quorum not reached

P1 changes view, writes & updates views

P5 initiates read, has quorum, reads stale data

P5 initiates write, no quorum, \( A_w \) not met, aborts.

Partition is repaired

P3 initiates write, notices repair

Views are updated to include P5; P5 is informed of updates

**Summary**

- Causal consistency & eventual consistency
- Quorums
  - Static
  - Optimistic
  - View-based

**Acknowledgements**

- These slides contain material developed and copyrighted by Indranil Gupta (UIUC).