CSE 486/586 Distributed Systems
Consistency --- 3

Steve Ko
Computer Sciences and Engineering
University at Buffalo

Recap
• Consistency
  – Linearizability
  – Sequential consistency
• Chain replication
• Primary-backup (passive) replication
• Active replication

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.

Relaxing the Guarantees
• Do we need sequential consistency?
  • Does everyone need to see these in this particular order? What kind of ordering matters? (Hint: causal)

Relaxing the Guarantees
• Sequential consistency
  – Still single-client, single-copy semantics, it's just that the single-client ordering does not strictly follow the actual-time order.
  – Every client should see the same write (update) order (every copy should apply all writes in the same order), since it works as if all clients read out of a single copy.
• E.g., writes are not applied in the same order:
  – P1: a.write(A)
  – P2: a.write(B)
  – P3: a.read() -> B a.read() -> A
  – P4: a.read() -> A a.read() -> B
• In the previous scenario,
  – Sequential consistency: All clients (all users’ browsers) will see all posts in the same order.

Relaxing the Guarantees
• For some applications, different clients (e.g., users) do not need to see the writes in the same order, but causality is still important (e.g., Facebook post-like pairs).
• Causal consistency
  – More relaxed than sequential consistency
  – Clients can read values out of order, i.e., it doesn’t behave as a single copy anymore.
  – Clients read values in-order for causally-related writes.
• How do we define “causal relations” between two writes?
  – (Roughly) One client reads something that another client has written; then the client writes something.
Causal Consistency

• Example 1:

<table>
<thead>
<tr>
<th>Event</th>
<th>P1</th>
<th>P2</th>
<th>P3</th>
<th>P4</th>
</tr>
</thead>
<tbody>
<tr>
<td>Write</td>
<td>W(x)1</td>
<td>W(x)2</td>
<td>R(x)1</td>
<td>R(x)2</td>
</tr>
<tr>
<td>Concurrent writes</td>
<td>W(x)3</td>
<td>R(x)3</td>
<td>R(x)2</td>
<td>R(x)3</td>
</tr>
</tbody>
</table>

This sequence obeys causal consistency

Causal Consistency Example 2

• Causally consistent?

<table>
<thead>
<tr>
<th>Event</th>
<th>P1</th>
<th>P2</th>
<th>P3</th>
<th>P4</th>
</tr>
</thead>
<tbody>
<tr>
<td>Write</td>
<td>W(x)1</td>
<td>W(x)2</td>
<td>R(x)1</td>
<td>R(x)1</td>
</tr>
<tr>
<td>Causally related</td>
<td>R(x)2</td>
<td>R(x)2</td>
<td>R(x)1</td>
<td>R(x)2</td>
</tr>
</tbody>
</table>

• No!

Causal Consistency Example 3

• Causally consistent?

<table>
<thead>
<tr>
<th>Event</th>
<th>P1</th>
<th>P2</th>
<th>P3</th>
<th>P4</th>
</tr>
</thead>
<tbody>
<tr>
<td>Write</td>
<td>W(x)1</td>
<td>W(x)2</td>
<td>R(x)2</td>
<td>R(x)1</td>
</tr>
<tr>
<td>Causally related</td>
<td>R(x)1</td>
<td>R(x)1</td>
<td>R(x)1</td>
<td>R(x)2</td>
</tr>
</tbody>
</table>

• Yes!

Implementing Causal Consistency

• We drop the notion of giving an illusion of a single copy.
  -- Writes can be applied in different orders across copies.
  -- Causally-related writes do need to be applied in the same order for all copies.
• Need a mechanism to keep track of causally-related writes.
• Due to the relaxed requirements, low latency is more tractable.

CSE 486/586 Administrivia

• Nothing really

Relaxing Even Further

• Let’s just do best effort to make things consistent.
• Eventual consistency
  -- Popularized by the CAP theorem.
  -- The main problem is network partitions.

Client + front end
withdraw(B, 4)
Client + front end
deposit(B, 3)
Network partition
Client + front end
Deposit managers
Client + front end
withdraw(B, 4)
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 sometimes happens.
• 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
**Static Quorums**

- \( r = 2, w = 2, N = 3 \): \( r + w > N, w > N/2 \)

![Diagram of Static Quorums](image)

- 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

**Summary**

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

**Acknowledgements**

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