Consistency and Replication

- Consistency models
  - Data-centric
  - Client-centric
- Consistency protocols
  - Primary-based protocols
  - Replicated-write protocols

Why Replicate?

- Data replication: common technique in distributed systems
- Reliability
  - If one replica is unavailable or crashes, use another
  - Protect against corrupted data
- Performance
  - Scale with size of the distributed system (replicated web servers)
  - Scale in geographically distributed systems (web proxies)
- Key issue: need to maintain consistency of replicated data
  - If one copy is modified, others become inconsistent
Object Replication

- Approach 1: application is responsible for replication
  - Application needs to handle consistency issues
- Approach 2: system (middleware) handles replication
  - Consistency issues are handled by the middleware
  - Simplifies application development but makes object-specific solutions harder

Replication and Scaling

- Replication and caching used for system scalability
- Multiple copies:
  - Improves performance by reducing access latency
  - But higher network overheads of maintaining consistency
  - Example: object is replicated \( N \) times
    - Read frequency \( R \), write frequency \( W \)
    - If \( R << W \), high consistency overhead and wasted messages
    - Consistency maintenance is itself an issue
      - Tight consistency requires globally synchronized clocks!
- Solution: loosen consistency requirements
  - Variety of consistency models possible
Data-Centric Consistency Models

- Consistency models (aka consistency semantics)
  - All models attempt to return the result of the last write for a read operation
    - Differ in how “last” write is determined/defined

Strict Consistency

- Any read always returns the result of the most recent write
  - Implicitly assumes the presence of a global clock
  - A write is immediately visible to all processes
- Difficult to achieve in real systems (due to network delays)
Sequential Consistency

- Sequential consistency: weaker than strict consistency
  - Assumes all operations are executed in some sequential order and each process issues operations in program order
    - Any valid interleaving is allowed
    - All agree on the same interleaving
    - Each process preserves its program order
    - Nothing is said about “most recent write”

(a) A sequentially consistent data store. (b) A data store that is not sequentially consistent.

Sequential Consistency Example

Four valid execution sequences for the three processes.
Causal Consistency

- Potentially causally related writes must be seen by all processes in the same order.
  - Concurrent writes may be seen in different orders on different machines

<table>
<thead>
<tr>
<th></th>
<th>P1: W(x)a</th>
<th>P2: R(x) ( \rightarrow ) W(x)b</th>
<th>P3: R(x) ( \rightarrow ) R(x)b</th>
<th>P4: R(x) ( \rightarrow ) R(x)b</th>
</tr>
</thead>
<tbody>
<tr>
<td>(a)</td>
<td>Not permitted</td>
<td>Permitted</td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

FIFO Consistency

- Writes from a process are seen by others in the same order. Writes from different processes may be seen in different order (even if causally related)
  - Relaxes causal consistency

<table>
<thead>
<tr>
<th></th>
<th>P1: W(x)a</th>
<th>P2: R(x) ( \rightarrow ) W(x)b</th>
<th>W(x)c</th>
<th>P3: R(x)b ( \rightarrow ) R(x) ( \rightarrow ) R(x)c</th>
<th>P4: R(x)a ( \rightarrow ) R(x)b ( \rightarrow ) R(x)c</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
<td>A valid sequence of events of FIFO consistency</td>
<td></td>
</tr>
</tbody>
</table>
Other Models

• Use granularity of critical sections instead of individual read/write
  – Send final result of critical section everywhere
  – Do not worry about propagating intermediate results
• Entry and release consistency
  – Assume shared data are made consistent at entry or exit points of critical sections
• Entry and release consistency are two forms of weak consistency where shared data are made consistent at synchronization time

Entry Consistency and Release Consistency

• Entry consistency: before enter critical section, block until all updates have reached my disk
• Release consistency: when exit the critical section, not release the lock until all updates I have made have been sent out and ACK’ed
Summary of Data-Centric Consistency Models

<table>
<thead>
<tr>
<th>Consistency</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>Strict</td>
<td>Absolute time ordering of all shared accesses matters.</td>
</tr>
<tr>
<td>Sequential</td>
<td>All processes see all shared accesses in the same order. Accesses are not ordered in time.</td>
</tr>
<tr>
<td>Causal</td>
<td>All processes see causally-related shared accesses in the same order.</td>
</tr>
<tr>
<td>FIFO</td>
<td>All processes see writes from each other in the order they were used. Writes from different processes may not always be seen in that order.</td>
</tr>
</tbody>
</table>

**a) Consistency models not using synchronization operations.**

<table>
<thead>
<tr>
<th>Consistency</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>Weak</td>
<td>Shared data can be counted on to be consistent only after a synchronization is done</td>
</tr>
<tr>
<td>Release</td>
<td>Shared data are made consistent when a critical section is exited</td>
</tr>
<tr>
<td>Entry</td>
<td>Shared data pertaining to a critical section are made consistent when a critical section is entered.</td>
</tr>
</tbody>
</table>

**b) Consistency models using synchronization operations.**