Consistency and Replication

• Today:
  – Introduction
  – Consistency models
    • Data-centric consistency models
    • Client-centric consistency models
  – Thoughts for the mid-term

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
      - What semantics to provide?
      - Tight consistency requires globally synchronized clocks!
- Solution: loosen consistency requirements
  - Variety of consistency semantics possible
Data-Centric Consistency Models

- Consistency model (aka *consistency semantics*)
  - Contract between processes and the data store
    - If processes obey certain rules, data store will work correctly
  - All models attempt to return the results 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 (network delays can be variable)
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”

Linearizability

• Assumes sequential consistency and
  – If TS(x) < TS(y) then OP(x) should precede OP(y) in the sequence
  – Stronger than sequential consistency
  – Difference between linearizability and serializbility?
    • Granularity: reads/writes versus transactions

• Example:

<table>
<thead>
<tr>
<th>Process P1</th>
<th>Process P2</th>
<th>Process P3</th>
</tr>
</thead>
<tbody>
<tr>
<td>x = 1;</td>
<td>y = 1;</td>
<td>z = 1;</td>
</tr>
<tr>
<td>print (y, z);</td>
<td>print (x, z);</td>
<td>print (x, y);</td>
</tr>
</tbody>
</table>
Linearizability Example

- Four valid execution sequences for the processes of the previous slide. The vertical axis is time.

\[
\begin{align*}
\text{x = 1;} & \quad \text{y = 1;} & \quad \text{z = 1;} \\
\text{print ((y, z);} & \quad \text{print (x, y);} & \quad \text{print (x, z);} \\
\text{y = 1;} & \quad \text{z = 1;} & \quad \text{z = 1;} \\
\text{print (x, z);} & \quad \text{print (x, y);} & \quad \text{print (x, y);} \\
\text{z = 1;} & \quad \text{x = 1;} & \quad \text{x = 1;} \\
\text{print (x, y);} & \quad \text{print (y, z);} & \quad \text{print (y, z)};
\end{align*}
\]

Prints: 001011 Prints: 101011 Prints: 010111 Prints: 111111
Signature: 001011 Signature: 101011 Signature: 110101 Signature: 111111

(a) (b) (c) (d)

Causal consistency

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

\[
\begin{align*}
\text{P1:} & \quad \text{W(x)a} & \quad \text{P1:} & \quad \text{W(x)a} \\
\text{P2:} & \quad \text{R(x)a} & \quad \text{P2:} & \quad \text{W(x)b} \\
\text{P3:} & \quad \text{R(x)b} & \quad \text{P3:} & \quad \text{R(x)b} \\
\text{P4:} & \quad \text{R(x)b} & \quad \text{P4:} & \quad \text{R(x)a}
\end{align*}
\]

(a) Not permitted (b) Permitted
Other models

- 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
  - Simple implementation: tag each write by (Proc ID, seq #)
- Even FIFO consistency may be too strong!
  - Requires all writes from a process be seen in order
- Assume use of critical sections for updates
  - Send final result of critical section everywhere
  - Do not worry about propagating intermediate results
    - Assume presence of synchronization primitives to define semantics

Other Models

- Weak consistency
  - Accesses to synchronization variables associated with a data store are sequentially consistent
  - No operation on a synchronization variable is allowed to be performed until all previous writes have been completed everywhere
  - No read or write operation on data items are allowed to be performed until all previous operations to synchronization variables have been performed.
- Entry and release consistency
  - Assume shared data are made consistent at entry or exit points of critical sections
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>Linearizability</td>
<td>All processes must see all shared accesses in the same order. Accesses are furthermore ordered according to a (nonunique) global timestamp.</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)

<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 region is exited.</td>
</tr>
<tr>
<td>Entry</td>
<td>Shared data pertaining to a critical region are made consistent when a critical region is entered.</td>
</tr>
</tbody>
</table>

(b)

Caching in WWW: Case Study

- Dramatic growth in world wide web traffic
- Web accesses are non-uniform in nature
  - Create hot-spots of server and network load, increase latency
- **Solution:** employ web proxy caches
  - Reduces user response times, server load, network load
Content Distribution Network

- Content distribution network (CDN)
  - Collection of proxies that act as intermediaries between servers and clients
  - Service a client request from “closest” proxy with the object
  - Similar benefits as single proxy environments, but larger scale
  - Example: Akamai CDN - 13,000+ proxies
- Caching in CDN => must maintain cache consistency

Consistency Mechanisms

- Time-to-live (TTL) values
  - Expiration time of cached document
  - Proxy must refresh from server after expiration
    - *Poll*: Use if-modified-since (IMS) HTTP requests
      - Weaker guarantees: document can change before expiration
  - Poll every time
    - Poll the server upon request for a cached object
    - Increases response time of requests
    - Provides stronger consistency guarantees
Consistency with Leases

- **Lease**: fixed duration contract between server and proxy
  - Server agrees to notify proxy of all updates to an object over duration $d$
  - “$d$” is the lease duration
  - Lease may be renewed upon expiry
- **Properties**:
  - Server needs to notify each proxy caching the object of an update
    - Excessive burden for popular objects
  - Leases require a server to maintain state
    - Overhead can be excessive for large CDNs
  - Leases provide stronger consistency guarantees
  - Push-based approach, server-initiated consistency

Mid-term Exam Comments

- Closed book, closed notes, 90 min
- Lectures 1-14 included on the test
  - Focus on things taught in class (lectures, in-class discussions)
  - Start with lecture notes, read corresponding sections from text
  - Supplementary readings are not directly included on the test.
- Exam structure: few short answer questions, mix of subjective and “design” questions
- Good luck!