Module 7 - Replication

Replication

- Why replicate?
  - Reliability
    - Avoid single points of failure
  - Performance
    - Scalability in numbers and geographic area

- Why not replicate?
  - Replication transparency
  - Consistency issues
    - Updates are costly
    - Availability may suffer if not careful
Logical vs Physical Objects

- There are physical copies of logical objects in the system.
- Operations are specified on logical objects, but translated to operate on physical objects.

![Diagram showing logical vs physical objects]

© Addison-Wesley Publishers 2000

Replication Architecture

![Diagram showing replication architecture]

© Addison-Wesley Publishers 2000
Object Replication (1)

Object Replication (2)

a) A remote object capable of handling concurrent invocations on its own.
b) A remote object for which an object adapter is required to handle concurrent invocations.
Object Replication (3)

a) A distributed system for replication-aware distributed objects.
b) A distributed system responsible for replica management

What will we study

- Consistency models - How do we reason about the consistency of the “global state”?
  - Data-centric consistency
    - Strict consistency
    - Linearizability
    - Sequential consistency
  - Client-centric consistency
    - Eventual consistency

- Update propagation - How does an update to one copy of an item get propagated to other copies?
- Replication protocols - What is the algorithm that takes one update propagation method and enforces a given consistency model?
Strict Consistency

- Any read\((x)\) returns a value corresponding to the result of the most recent write\((x)\).

Machine 1

\[
\begin{align*}
R(x) & \quad t_1 \\
\end{align*}
\]

Machine 2

\[
\begin{align*}
W_2(x) & \quad t_2 \\
R(x, b) & \quad \text{WRONG!}
\end{align*}
\]

- Relies on absolute global time; all writes are instantaneously visible to all processes and an absolute global time order is maintained.

- Cannot be implemented in a distributed system

\[
\begin{array}{c|c|c|c|c|c}
P_1: & W(x) & R(x) & R(x) & R(x) & R(x) \\
\hline
P_2: & W(x) & R(x) & R(x) & R(x) & R(x) \\
\end{array}
\]

\(\text{Strictly consistent}\)

\(\text{Not strictly consistent}\)

From: Tanenbaum and van Steen, Distributed Systems: Principles and Paradigms
© Prentice-Hall, Inc. 2002

Linearizability

- The result of the execution should satisfy the following criteria:
  - Read and write by all processes were executed in some serial order and each process’s operations maintain the order of specified;
  - If \(ts_{op_1}(x) < ts_{op_2}(y)\) then \(op_1(x)\) should precede \(op_2(y)\) in this sequence.
    This specifies that the order of operations in interleaving is consistent with the real times at which the operations occurred in the actual implementation.

- Requires synchronization according to timestamps, which makes it expensive.

- Used only in formal verification of programs.
Sequential Consistency

■ Similar to linearizability, but no requirement on timestamp order.
■ The result of execution should satisfy the following criteria:
  ● Read and write operations by all processes on the data store were executed in some sequential order;
  ● Operations of each individual process appear in this sequence in the order specified by its program.
■ These mean that all processes see the same interleaving of operations \(\Rightarrow\) similar to serializability.

\[
\begin{array}{c|c|c|c|c}
  P1: & W(a) & W(b) & W(c) & W(d) \\
  P2: & R(b) & R(c) & R(d) & R(e) \\
  P3: & & & & \\
  P4: & R(b) & R(c) & R(d) & R(e) \\
\end{array}
\]

Transactional Replica Consistency

■ Efficient implementation of sequential consistency requires transactions.
■ One-copy equivalence
  ● The effect of transactions performed by clients on replicated objects should be the same as if they had been performed on a single set of objects.
■ One-copy serializability
  ● The effect of transactions performed by clients on replicated objects should be the same as if they had been performed \textit{one at-a-time} on a single set of objects.
  ● This is done within transactional boundaries.
Client-Centric Consistency

- More relaxed form of consistency: only concerned with replicas being eventually consistent (eventual consistency).
- In the absence of any further updates, all replicas converge to identical copies of each other: only requires guarantees that updates will be propagated.
- Easy if a user always accesses the same replica; problematic if the user accesses different replicas.
  - Client-centric consistency: guarantees for a single client the consistency of access to a data store.

---

Client-Centric Consistency (2)

- Monotonic reads
  - If a process reads the value of a data item $x$, any successive read operation on $x$ by that process will always return that same value or a more recent value.

- Monotonic writes
  - A write operation by a process on a data item $x$ is completed before any successive write operation on $x$ by the same process.

- Read your writes
  - The effect of a write operation by a process on data item $x$ will always be seen by a successive read operation on $x$ by the same process.

- Writes follow reads
  - A write operation by a process on a data item $x$ following a previous read operation on $x$ by the same process is guaranteed to take place on the same or more recent value of $x$ that was read.
Replica Placement Alternatives

- **Permanent replicas**
  - Put a number of replicas at specific locations
  - Mirroring

- **Server-initiated replicas**
  - Server decides where and when to place replicas
  - Push caches

- **Client-initiated replicas**
  - Client caches

Update Propagation

- **What to propagate?**
  - Propagate only a notification
    - Invalidation
  - Propagate updated data
    - Possibly only logs
  - Propagate the update operation
    - Active replication

- **Who propagates?**
  - Server: push approach
  - Client: pull approach

- **Epidemic protocols**
  - Update propagation in eventual-consistency data stores.
Pull versus Push Protocols

<table>
<thead>
<tr>
<th>Issue</th>
<th>Push-based</th>
<th>Pull-based</th>
</tr>
</thead>
<tbody>
<tr>
<td>State of server</td>
<td>List of client replicas and caches</td>
<td>None</td>
</tr>
<tr>
<td>Messages sent</td>
<td>Update (and possibly fetch update later)</td>
<td>Poll and update</td>
</tr>
<tr>
<td>Response time at client</td>
<td>Immediate (or fetch-update time)</td>
<td>Fetch-update time</td>
</tr>
</tbody>
</table>

From: Tanenbaum and van Steen, Distributed Systems: Principles and Paradigms
© Prentice-Hall, Inc. 2002

Replication Protocols

- We focus on those that enforce sequential consistency.
- Primary-based protocols
  - Remote-Write protocols
  - Local-Write protocols
- Replicated Write protocols
  - Active replication
  - Quorum-based protocols
- Read-one-Write-All (ROWA)
Primary Copy Remote-Write Protocol

- W1. Write request
- W2. Forward request to primary
- W3. Tell backups to update
- W4. Acknowledge update
- W5. Acknowledge write completed

Primary Copy Local-Write Protocol

- W1. Write request
- W2. Move item x to new primary
- W3. Acknowledge write completed
- W4. Tell backups to update
- W5. Acknowledge update
Active Replication

- Requires a process, for each replica, that can perform the update on it
- How to enforce the update order?
  - Totally-ordered multicast mechanism needed
  - Can be implemented by Lamport timestamps
  - Can be implemented by sequencer
- Problem of replicated invocations
  - If an object \( A \) invokes another object \( B \), all replicas of \( A \) will invoke \( B \) (multiple invocations)

Replicated Invocations Problem

![Diagram showing replicated invocations](image-url)
Solution to Replicated Invocations

Quorum-Based Protocol

- Assign a vote to each copy of a replicated object (say $V_i$) such that $\sum V_i = V$
- Each operation has to obtain a read quorum ($V_r$) to read and a write quorum ($V_w$) to write an object
- Then the following rules have to be obeyed in determining the quorums:
  - $V_r + V_w > V$: an object is not read and written by two transactions concurrently
  - $V_w > V/2$: two write operations from two transactions cannot occur concurrently on the same object
Quorum Example

Three examples of the voting algorithm:

a) A correct choice of read and write set

b) A choice that may lead to write-write conflicts

c) ROWA