Consistency and Replication

DR. YINGWU ZHU
Reasons for Replication

• Data are replicated to increase the reliability of a system.

• Replication for performance
  ▪ Scaling in numbers
  ▪ Scaling in geographical area

• Pros & Cons
  ▪ Gain in performance, reliability
  ▪ Cost of increased bandwidth for maintaining replication (consistent replicas)
Topics

- **Data-centric consistency models**
  - System-wide consistent view on data store
  - Simultaneous updates

- **Consistency protocols to implement data-centric consistency models**

- **Client-centric consistency models**
  - Often lack in simultaneous updates
  - Or once they occur, they can be easily resolved

- **Replica placement & content distribution**
The general organization of a logical data store, physically distributed and replicated across multiple processes.
Consistency Model

- A contract between processes and the data store
- If the contract is followed, the store promises to work correctly
- Normally, a read operation expects to return a value that shows the results of the last write operation on that data
  - Hard to implement w/o a global clock!
  - Using locks to implement it
  - Practical: weaker consistency models!
Data-Centric Consistency Models

- Providing **system-wide consistent view** on a data store
- Assume that concurrent processes may be simultaneously update data (simultaneous updates are the norm)
- Two models
  - Sequential consistency
  - Causal consistency
Sequential Consistency (1)

Behavior of two processes operating on the same data item. The horizontal axis is time.

- Propagation of updates from one copy to another
A data store is **sequentially consistent** when:

- The result of any execution is the same as if the (read and write) operations by all processes on the data store …
  
  - were executed in some sequential order, i.e., all processes see the same interleaving of operations
  - the operations of each individual process appear in this sequence in the order specified by its program.
Sequential Consistency (3)

(a) A sequentially consistent data store.
(b) A data store that is not sequentially consistent.
### Sequential Consistency (4)

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

- Three concurrently-executing processes.
Sequential Consistency (5)

| x ← 1; print(y, z); y ← 1; print(x, z); z ← 1; print(x, y);  |
|-----------------|-----------------|-----------------|-----------------|
| x ← 1; y ← 1; print(x, z); z ← 1; print(x, y);           |
| y ← 1; print(x, z); print(x, y);                          |
| z ← 1; print(y, z); print(y, z);                          |
| print(x, y); print(x, y);                                 |

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

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

Four valid execution sequences (90 in total) for the processes in the preceding slide, must be accepted by all processes! Follow contract!
Causal Consistency (1)

Weaker than sequential consistency

For a data store to be considered causally consistent, it is necessary that the store obeys the following condition:

- **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.
Causal Consistency (2)

- $W_2(x)b$ and $W_1(x)c$ are concurrent
- This sequence is allowed with a causally-consistent store, but not with a sequentially consistent store.
Causal Consistency (3)

<table>
<thead>
<tr>
<th>P1:</th>
<th>W(x)a</th>
</tr>
</thead>
<tbody>
<tr>
<td>P2:</td>
<td>R(x)a</td>
</tr>
<tr>
<td>P3:</td>
<td></td>
</tr>
<tr>
<td>P4:</td>
<td></td>
</tr>
</tbody>
</table>

(a) A violation of a causally-consistent store.

(b) A correct sequence of events in a causally-consistent store.
Some facts...

- Performance and strong consistency can **not** be achieved simultaneously.
- Some applications (e.g., DNS, Web) trade consistency for performance!
  - Why?
  - Group discussion.
Some facts...

- Performance and strong consistency cannot be achieved simultaneously.
- Some applications (e.g., DNS, Web) trade consistency for performance!
  - Often lack in simultaneous updates.
  - Or once they occur, they can be easily resolved.
Topics

- **Data-centric consistency models**
  - System-wide consistent view on data store
  - Simultaneous updates

- Consistency protocols to implement data-centric consistency models

- **Client-centric consistency models**
  - Often lack in simultaneous updates
  - Or once they occur, they can be easily resolved

- Replica placement & content distribution
Consistency Protocols

- Describes an implementation of a specific consistency model
- Primary-based protocols (writes on the primary)
  - Remote-write
  - Local-write
  - Straightforward implementation of sequential consistency
- Replicated-write protocols (writes performed on multiple replicas)
  - Quorum-based
Remote-Write Protocols

- The principle of a primary-backup protocol: each update goes to the primary first, the primary performs the update and then forwards the update to the backups which in turn perform the update and send back ack. After all backup have updated their local copy, ack. back to the initiator.
Primary-backup protocol in which the primary migrates to the process wanting to perform an update.

Useful for successive local writes!
Quorum-Based Protocols

- Read quorum $N_R$
- Write quorum $N_W$
- Meet 2 constraints:
  - $N_R+N_W > N$ (avoid R-W conflicts)
  - $N_W > N/2$ (avoid W-W conflicts)
Quorum-Based Protocols

- 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) A correct choice, known as ROWA (read one, write all).
- **Data-centric consistency models**
  - System-wide consistent view on data store
  - Simultaneous updates

- **Consistency protocols to implement data-centric consistency models**

- **Client-centric consistency models**
  - Often lack in simultaneous updates
  - Or once they occur, they can be easily resolved

- **Replica placement & content distribution**
Client-Centric Consistency Models

- Distinct from data centric consistency models
- Assumptions
  - Often lack in simultaneous updates
  - Or once they occur, they can be easily resolved
  - Examples: DNS and Web
Eventual Consistency

- If no updates take place for a long time, all replicas will gradually become consistent.
- It works as long as clients always access the same replica (updates are guaranteed to propagate to all replicas).
- However, it poses (inconsistency) problems when clients access different replicas over a short period of time.
Eventual Consistency

Updates have not propagated to all replicas yet!

Client moves to other location and (transparently) connects to other replica

Replicas need to maintain client-centric consistency

Wide-area network

Read and write operations

Distributed and replicated database

The principle of a mobile user accessing different replicas of a distributed database. → inconsistency, solved by client-centric consistency
Client-Centric Consistency

- Alleviate the problem
- Provides guarantees for a single client concerning the consistency of accesses to a data store by that client
- No guarantees are given for concurrent accesses by different clients!
Monotonic Reads (1)

A data store is said to provide monotonic-read consistency if the following condition holds:

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.

Example: a distributed email database

The email read in A location can also appear in B location.
Monotonic Reads (2)

WS(X): the set of write operations on item X
WS(X1,X2): the set of write operations at X1 also performed in X2, part of WS(X2)

L1: \( WS(x_1) \)  
\[ R(x_1) \]  
L2: \( WS(x_1;x_2) \)  
\[ R(x_2) \]  
(a) Indicates order

- The read operations performed by a single process P at two different local copies of the same data store. (a) A monotonic-read consistent data store.
The read operations performed by a single process P at two different local copies of the same data store.

(b) A data store that does not provide monotonic reads.
In a monotonic-write consistent store, the following condition holds:

A write operation by a process on a data item $x$ is completed before any successive write operation on $x$ by the same process.
The write operations performed by a single process $P$ at two different local copies of the same data store. (a) A monotonic-write consistent data store.
The write operations performed by a single process P at two different local copies of the same data store. (b) A data store that does not provide monotonic-write consistency.
A data store is said to provide read-your-writes consistency, if the following condition holds:

- 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.
(a) A data store that provides read-your-writes consistency.
(b) A data store that does not.
For web server (which updates the web pages) and web browsers, does the web browsing provide “read-your-writes” consistency in the real world?
A data store is said to provide writes-follow-reads consistency, if the following holds:

- 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 a more recent value of $x$ that was read.
(a) A writes-follow-reads consistent data store.
(b) A data store that does not provide writes-follow-reads consistency.
Topics

- **Data-centric consistency models**
  - System-wide consistent view on data store
  - Simultaneous updates
- **Consistency protocols to implement data-centric consistency models**
- **Client-centric consistency models**
  - Often lack in simultaneous updates
  - Or once they occur, they can be easily resolved
- **Replica placement & content distribution**
Replication Examples

- Web applications
The logical organization of different kinds of copies of a data store into three concentric rings.

Replicated web servers, server-initiated replicas on demand, client-initiated replicas by caching.
Permanent Replicas

- Web servers
  - Mirror sites
  - A small number of servers in each site (load balancing)
Server-Initiated Replicas

- Counting access requests from different clients (and where they are from).
- Assuming that P is closest to C1 and C2
Client-Initiated replicas

- Driven by clients
- Close to client side
  - Browser cache
  - Proxy cache (hierarchy?)
如何传播更新的内容？

1. 仅传播更新的通知（无效化协议）：减少带宽成本
2. 传输数据从一个副本到另一个副本。
3. 传播更新操作到其他副本：最小化带宽成本？

考虑reads : writes ratio
**Content Distribution**

- **Design issue**: whether updates are **pulled** or **pushed**

<table>
<thead>
<tr>
<th>Issue</th>
<th>Push-based</th>
<th>Pull-based</th>
</tr>
</thead>
<tbody>
<tr>
<td>State at 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>

- **Lease** (hybrid): push guaranteed within lease; otherwise poll & pull
- **Timeout**(TTL) in web caches