HY-559
Infrastructure Technologies for Large-Scale Service-Oriented Systems

Kostas Magoutis
magoutis@csd.uoc.gr
http://www.csd.uoc.gr/~hy559
Geo-replicated storage

Image courtesy of: L. Wyatt et al. Don’t Settle for Eventual Consistency, CACM’14
Geo-replicated storage

CAP theorem, cannot simultaneously achieve
- Consistency
- Availability
- Partition tolerance

Choose any two
(e.g., CA/CP -> Linearizability)

Real-world services opt for
- Availability
- Partition tolerance
Desirable properties in geo-replicated services

- **ALPS**
  - Availability
  - Low latency
  - Partition tolerance
  - Scalability

- Linearizability (strong consistency) is not partition-tolerant

- One way to achieve ALPS: Eventual consistency
  - Writes to one data center (DC) eventually appear at other DCs
  - If all DCs receive the same set of writes, they will have the same values for all data

- This can lead to problems
A collection of operations is linearizable if each operation appears to occur instantaneously and exactly once at some point-in-time between its invocation and its completion.

An example with replicated data

Data type: 4-location byte-valued read/write snapshot register

A multi-location read-write memory has
- a set of locations (or addresses)
- operations such as
  - read\((a)\)
  - write\((a, w)\)
  - snapshot()
- snapshot() returns a set of values, one for each location

<table>
<thead>
<tr>
<th>location</th>
<th>value</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>0</td>
</tr>
<tr>
<td>2</td>
<td>0</td>
</tr>
<tr>
<td>3</td>
<td>0</td>
</tr>
<tr>
<td>4</td>
<td>0</td>
</tr>
</tbody>
</table>
Linearizable execution

Implementation rules:
• each read or snapshot is done on one replica
• each write is done on both replicas
• different writes are done in the same order at the replicas
• a write doesn’t return to the client until acked

Note:
• Writes indeed applied in the same order on all replicas in this example
• Mechanism for achieving order between writes is not shown here

The order of operations as they occur in the sequence must not contradict any order information visible to an observer of the system execution.

From “Consistency Models for Replicated Data”, A. D. Fekete, Krithi Mamamritham

B. Charron-Bost, F. Pedone, and A. Schiper (Eds.): Replication, LNCS 5959, pp. 1–17, 2010. © Springer-Verlag Berlin Heidelberg 2010
Sequential consistency

Implementation rules:

- each read or snapshot is done on one replica
- each write is done on both replicas
- different writes are done in the same order at the replicas
- a write returns to the client as soon as messages sent out

From “Consistency Models for Replicated Data”, A. D. Fekete, Krithi Mamamritham
Weak consistency

Implementation rules:
- each read or snapshot is done on one replica
- each write is done on both replicas
- different writes are done in the same order at the replicas
- a write returns to the client as soon as messages sent out

Cannot find a legal history that would satisfy either linearizability or SC conditions
Desirable properties in geo-replicated services

- **ALPS**
  - Availability
  - Low latency
  - Partition tolerance
  - Scalability

- Linearizability (strong consistency) is not partition-tolerant

- One way to achieve ALPS: Eventual consistency
  - Writes to one data center (DC) eventually appear at other DCs
  - If all DCs receive the same set of writes, they will have the same values for all data

- This can lead to problems
Problem 1: Comment reordering

Alice: “I’ve lost my wedding ring”
Alice: “Whew found it upstairs!”
Bob: “I’m glad to hear that”

HY-559 University of Crete
Problem 2: Double money withdrawal
General architecture of modern web services

Images courtesy of: L. Wyatt et al. Don’t Settle for Eventual: Scalable Causal Consistency for Wide-Area Storage with COPS, SOSP'11
Example of causal relationships

1. **Execution Thread.** If \( a \) and \( b \) are two operations in a *single thread of execution*, then \( a \sim b \) if operation \( a \) happens before operation \( b \).

2. **Gets From.** If \( a \) is a put operation and \( b \) is a get operation that returns the value written by \( a \), then \( a \sim b \).

3. **Transitivity.** For operations \( a, b, \) and \( c \), if \( a \sim b \) and \( b \sim c \), then \( a \sim c \).

**Client 1**
\[
\text{put}(x, 1) \rightarrow \text{put}(y, 2) \rightarrow \text{put}(x, 3)
\]

**Client 2**
\[
\text{get}(y)=2 \rightarrow \text{put}(x, 4)
\]

**Client 3**
\[
\text{get}(x)=4 \rightarrow \text{put}(z, 5)
\]
(m:1) means “key m, version 1”
# Alice’s Photo Upload
ctx_id = createContext()  // Alice logs in
put(Photo, "Portuguese Coast", ctx_id)
put(Album, "add &Photo", ctx_id)
deleteContext(ctx_id)  // Alice logs out

# Bob’s Photo View
ctx_id = createContext()  // Bob logs in
"&Photo" ← get(Album, ctx_id)
"Portuguese Coast" ← get(Photo, ctx_id)
deleteContext(ctx_id)  // Bob logs out
Causality and dependency

<table>
<thead>
<tr>
<th>user</th>
<th>op ID</th>
<th>operation</th>
</tr>
</thead>
<tbody>
<tr>
<td>Alice</td>
<td>w₁</td>
<td>write(Alice; town, NYC)</td>
</tr>
<tr>
<td>Bob</td>
<td>r₂</td>
<td>read(Alice; town)</td>
</tr>
<tr>
<td>Bob</td>
<td>w₃</td>
<td>write(Bob; town, LA)</td>
</tr>
<tr>
<td>Alice</td>
<td>r₄</td>
<td>read(Bob; town)</td>
</tr>
<tr>
<td>Carol</td>
<td>w₅</td>
<td>write(Carol; likes, ACM, 8/31/12)</td>
</tr>
<tr>
<td>Alice</td>
<td>w₆</td>
<td>write(Alice; likes, ACM, 9/1/12)</td>
</tr>
<tr>
<td>Alice</td>
<td>r₇</td>
<td>read(Carol; likes, ACM)</td>
</tr>
<tr>
<td>Alice</td>
<td>w₈</td>
<td>write(Alice; friends, Carol, 9/2/12)</td>
</tr>
</tbody>
</table>

![Diagram showing causality and dependency](image)

<table>
<thead>
<tr>
<th>op ID</th>
<th>Dependencies</th>
</tr>
</thead>
<tbody>
<tr>
<td>w₁</td>
<td>-</td>
</tr>
<tr>
<td>w₃</td>
<td>w₁</td>
</tr>
<tr>
<td>w₅</td>
<td>-</td>
</tr>
<tr>
<td>w₆</td>
<td>w₃, w₁</td>
</tr>
<tr>
<td>w₈</td>
<td>w₆, w₅, w₃, w₁</td>
</tr>
</tbody>
</table>
Problem 1 fixed

Alice: “I’ve lost my wedding ring”
Alice: “Whew found it upstairs!”
Bob: “I’m glad to hear that”
How to handle concurrent writes?

• Causally-unrelated writes require additional support
  – Hard to maintain global invariants (e.g., balance > 0)

• These are rare, and can be handled with
  – Later reconciliation
  – Last-writer-wins