Recap

- Learned the importance naming in distributed systems
- Studied name resolution
Roadmap (so far)

Logical order of events

- Global clock
- Internal clock in components

Diagram:

- DNS
- Printer1
- Amazon.com
- Ut.ee
- P_k
- P_n
Agenda

Logical order of events

Global clock

Internal clock in components
Agenda

• **Goal:** To learn the role of replication and consistency
• **Content:**
  - Reasons of replication
  - Replication as scaling technique
  - Data-centric models
  - Client-centric consistency models

After this lecture, you should be able to:

• Understand the importance of replication to improve transparency and reliability in distributed systems
Replication
Reasons for replication (Cloning)
Reasons for replication (Cloning)
# Reasons for replication

## Essence

- **Enhancing reliability**
  - Protection against hardware crashes and corruption of data

- **Improving performance**
  - Scaling in terms of numbers and geographical area
  - Caching is a special form of replication
  - Trade-off between performance and scalability

## Challenges (Keep replicas consistent)

- When one copy is updated, other copies need to be updated, as well
- A number of consistency models
- Protocols for distribution of updates
System performance and replication

Essence

Replication improves server performance, particularly reliability and availability.

However, if data is not replicated properly, this can lead to a system that cannot be utilized in practice.
### Adaptation triggering

- **When and how adaptation are triggered?**
  - Often based on periodic evaluation of performance metrics

- **Periodic evaluation may miss sudden changes such as** flash crowds
  - Sudden burst of requests for a specific web document that can bring an entire web site down

- **Flash crowds are difficult to deal with**
  - Overprovisioning is expensive
  - **Flash-crowd predictor** for giving enough time for dynamic replication before the flash crowd
    - Flash crowds can have very different access patterns
Scaling based on workload

Increased workload

Scaling based on workload

Workload quantification

2 days
(a)

2 days
(b)

6 days
(c)

2.5 days
(d)
# Replication as scaling technique

## Further use of replication

- Replication and caching widely applied as scaling technique
  - Placing copies of objects near processes using them improves performance through reduction of access time

- Possible trade-off: keeping copies up-to-date may require more network bandwidth, if access-to-update ratio is low

- Keeping multiple copies consistent may itself be subject to serious scalability problems
  - Ideally update of all copies is performed as single atomic operation

- Required global synchronization can be very costly
  - Solution: loosen consistency constraints
Replication as scaling technique
Replication issues

Main issue
To keep replicas consistent, we generally need to ensure that all conflicting operations are done in the same order everywhere.

Conflict operations
- **Read–write conflict**: a read operation and a write operation act concurrently
- **Write–write conflict**: two concurrent write operations

Issue
Guaranteeing global ordering on conflicting operations may be a costly operation, downgrading scalability. **Solution**: weaken consistency requirements so that hopefully global synchronization can be avoided.
Consistency Models
Data-centric consistency models

Data store (Shared data)

- Broader term for a (distributed) shared memory/database/file system
- May be physically distributed across multiple machines
- Replicated across multiple processes
  - Each process is assumed to have a local copy of the entire store
- Write operation changes the data, propagated to all copies
# Data-centric consistency models

## Consistency model

- **Contract between processes and data store**
  - Processes agree to obey certain rules
  - Store promises to work correctly
    - Normally, read operation on data item returns value corresponding to the result of the last write operation on that data item
- **No global clock to define precisely last write → consistency model**
  - Restricts the values that a read operation can return
  - Trade-off between flexibility and performance

## Note (Different from coherense model)

Describes what can be expected from single data items replicated at several places
No best solution to replicating data

- No general efficient solution for consistency problems
- Loosening consistency is highly dependent on applications

Continuous consistency ranges: three independent axes for defining inconsistencies between replicas (Yu & Vahdat)

- Deviation in numerical values
  - Absolute numerical deviations
  - Relative numerical deviations
- Deviation in staleness
  - Relate to the last time a replica was updated
- Deviation with respect to ordering of update operations
  - Ordering of updates are allowed to be different at various replicas, as long as differences remain bounded
## Continuous consistency

We can actually talk about a degree of consistency:

- Replicas may differ in their numerical value
- Replicas may differ in their relative staleness
- There may be differences with respect to (number and order) of performed update operations

### Consistency unit

Consistency unit → specifies the data unit over which consistency is to be measured.
Conit (contains the variables g, p, and d)

- Each replica has a vector clock: ([known] time @ A, [known] time @ B)
- B sends A operation [(5, B) : g ← d +45]; A has made this operation permanent (cannot be rolled back)
Example

Conit (contains the variables $g$, $p$, and $d$)

- A has three pending operations $\rightarrow$ order deviation = 3
- A missed two operations from B; max diff is $70 + 412$ units $\rightarrow$ (2,482)
Continuous consistency

Trade-off in conit size

- Coarse-grained (large conit) may bring replicas sooner in inconsistent state
  Example: two replicas may differ in no more than one outstanding update
    (a) Two updates lead to update propagation
    (b) No update propagation is needed (yet)
    (c) If data items contained in a conit are used completely independently, they are said to **falsely share** the conit
- Fine-grained (small conit) may lead to excess overhead in managing conits
Continuous consistency

Two important issues in practical use of conits

- Protocol are needed to enforce consistency
- Program developers must specify consistency requirements for their applications, which may be extremely difficult in practice
Consistent ordering of operations

**Consistency operations**

- Replicas need to agree on the global ordering of the updates to be committed
- Different models for consistent ordering operations on shared, replicated data
- Models differ in terms of their consistency level
  
  - Strict → sequential → causal
Strict consistency

**Principle**

"Any read on a data item x returns a value corresponding to the result of the most recent write on x”

<table>
<thead>
<tr>
<th></th>
<th>P1:</th>
<th>P2:</th>
</tr>
</thead>
<tbody>
<tr>
<td>Write</td>
<td>W(x)</td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Read</td>
<td>R(x)</td>
<td></td>
</tr>
</tbody>
</table>

- **Strictly consistent data store**
- **Data store not strictly consistent**

- \( W_1(x)a \) write by process P1 to data item x with the value a
- \( R_2(x)a \) read by process P2 from data item x returning the value a
- Each data item is initially assumed to be NIL

**Ideal model, impossible to implement in a distributed system**

Implicit assumption of absolute global time
Sequential consistency

**Principle**

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 and the operations of each individual process appear in this sequence in the order specified by its program.

(a) Sequentially consistent data store

W2(x)b appears to have taken place before W1(x)a

(b) Data store not sequentially consistent

Processes do not see same interleaving of write operations
Sequential consistency

Example on sequential consistency

- Three concurrently executing processes
  
<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>

- There are 720 (6!) potential execution sequences
  - Only 90 execution sequences are valid, though

- Four valid execution sequences (vertical axis is time)
Causal consistency

Principle

*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.*

- Weakening of sequentical consistency by making a distinction between causally related events and concurrent (causally not related) events

- Example #1: event sequence allowed with causally consistent data store but not with sequentially consistent data store

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

Writes $W_2(x)b$ and $W_1(x)c$ are concurrent, hence in causally consistent data store it is not required that all processes see them in same order which would be the case in sequentially consistent data store.
Causal consistency

Example #2: A violation of causally consistent data store

- Writes $W_1(x)a$ and $W_2(x)b$ are causally related so all processes must see them in same order

A correct sequence of events in a causally consistent data store

- Concurrent writes are not required to be globally ordered

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

(a)

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

(b)

Note

Implementing causal consistency requires keeping track of which processes have seen which writes (e.g. vector timestamps)
# Grouping operations

## Definition

- Accesses to locks are sequentially consistent.
- No access to a lock is allowed to be performed until all previous writes have completed everywhere.
- No data access is allowed to be performed until all previous accesses to locks have been performed.

## Basic idea

It is not important that reads and writes of a series of operations are immediately known to other processes. It is important to know the effect of the series itself to be known.
Grouping operations

Consistency models enforced with synchronization variables

- **Weak consistency**: shared data can be counted to be consistent only after a synchronization is done
- **Release consistency**: shared data are made consistent when a critical region is exited
- **Entry consistency**: shared data pertaining to a critical region are made consistent when a critical region is entered

An example of entry consistency

- Locks are associated with each data item
- Process P2 does not acquire lock for Y, hence may read NIL
Example: Consistency for mobile users

Consider a distributed database to which you have access through your notebook. Assume your notebook acts as a front end to the database.

• At location A you access the database doing reads and updates.
• At location B you continue your work, but unless you access the same server as the one at location A, you may detect inconsistencies:
  your updates at A may not have yet been propagated to B
  you may be reading newer entries than the ones available at A
  your updates at B may eventually conflict with those at A

Note

The only thing you really want is that the entries you updated and/or read at A, are in B the way you left them in A. In that case, the database will appear to be consistent to you
Client-centric consistency models

Basic architecture

The principle of a mobile user accessing different replicas of a distributed database

![Diagram showing client-centric consistency models with a mobile user accessing multiple replicas through a wide-area network.](image-url)
**Monotonic reads**

**Definition**

- 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 or a more recent value.
- 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.
  - b) A data store that does not provide monotonic reads

<table>
<thead>
<tr>
<th>L1</th>
<th>L2</th>
</tr>
</thead>
<tbody>
<tr>
<td>$W_1(x_i)$</td>
<td>$W_2(x_i;x_2)$</td>
</tr>
<tr>
<td>$R_i(x_i)$</td>
<td>$R_i(x_2)$</td>
</tr>
</tbody>
</table>

(a)

- $W_1(x_2)$ is the write operation by process $P_1$ that leads to version $x_2$ of $x$.
- $W_1(x_i;x_j)$ indicates $P_1$ produces version $x_j$ based on a previous version $x_i$.
- $W_1(x_i|x_j)$ indicates $P_1$ produces version $x_j$ concurrently to version $x_i$. 

<table>
<thead>
<tr>
<th>L1</th>
<th>L2</th>
</tr>
</thead>
<tbody>
<tr>
<td>$W_1(x_i)$</td>
<td>$W_2(x_i</td>
</tr>
<tr>
<td>$R_i(x_i)$</td>
<td>$R_i(x_2)$</td>
</tr>
</tbody>
</table>

(b)
Monotonic writes

Definition

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

\[
\begin{align*}
\text{L1:} & \quad W_i(x) \\
\text{L2:} & \quad W_i(x_1,x_2) \quad W_i(x_2,x_3) \\
\text{(a)}
\end{align*}
\]

\[
\begin{align*}
\text{L1:} & \quad W_i(x) \\
\text{L2:} & \quad W_i(x_1|x_2) \quad W_i(x_1|x_3) \\
\text{(b)}
\end{align*}
\]

\[
\begin{align*}
\text{L1:} & \quad W_i(x) \\
\text{L2:} & \quad W_i(x_1|x_2) \quad W_i(x_2|x_3) \\
\text{(c)}
\end{align*}
\]

\[
\begin{align*}
\text{L1:} & \quad W_i(x) \\
\text{L2:} & \quad W_i(x_1|x_2) \quad W_i(x_1|x_3) \\
\text{(d)}
\end{align*}
\]

a) A monotonic-write consistent data store.
b) A data store that does not provide monotonic-write consistency.
c) Again, no consistency as \( WS(x_1|x_2) \) and thus also \( WS(x_1|x_3) \).
d) Consistent as \( WS(x_1;x_3) \) although \( x_1 \) has apparently overwritten \( x_2 \).
Reading your writes

Definition

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.

\[
\begin{array}{c}
\text{L1: } W_i(x_i) \\
\text{L2: } W_2(x_1, x_2) \quad R_i(x_2)
\end{array}
\]

\[
\begin{array}{c}
\text{L1: } W_i(x_i) \\
\text{L2: } W_2(x_1|x_2) \quad R_i(x_2)
\end{array}
\]

(a) A data store that provides read-your-writes consistency.

(b) A data store that does not.
Write follows read

Essence

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.

$$\begin{align*}
\text{L1: } & W_i(x_i) & R_i(x_i) \\
\text{L2: } & W_3(x_i;x_2) & W_2(x_2;x_3)
\end{align*}$$

(a)

$$\begin{align*}
\text{L1: } & W_i(x_i) & R_i(x_i) \\
\text{L2: } & W_3(x_i;x_2) & W_2(x_i;x_3)
\end{align*}$$

(b)

a) A writes-follow-reads consistent data store.
b) A data store that does not provide writes-follow-reads consistency
Summary

What we learned?

• The concept of replication
• The relation between replication and consistency
• Multiple consistency models
Next lecture

Consistency and Replication II
Questions?

E-mail: huber.flores@ut.ee