Replication

Make copies of services on multiple machines.

Why?:

- Reliability
  - Redundancy
- Performance
  - Increase processing capacity
  - Reduce communication
- Scalability (prevent centralisation)
  - Prevent overloading of single server (size scalability)
  - Avoid communication latencies (geographic scalability)

Data vs Control Replication

Data Replication (Server Replication/Mirroring):

- FTP Server
  - FTP
  - Data

Data Replication (Caching):

- Cache
  - Pop Website
  - Web Server
  - HTTP

What’s the difference between mirroring and caching?
Control Replication:

What are the challenges of doing this?

Data and Control Replication:

We will be looking primarily at data replication (including combined data and control replication).

Replication Issues

Updates
- Consistency (how to deal with updated data)
- Update propagation

Replica placement
- How many replicas?
- Where to put them?

Redirection/Routing
- Which replica should clients use?

Distributed Data Store

- data-store stores data items

Client’s Point of View:
Distributed Data-Store’s Point of View:

- Client A
- Client B
- Client C
- Client D
- Replica 1
- Replica 2
- Replica 3
- Replica 4

Data Model:
- data item: simple variable
- data item values: explicit (0, 1), abstract (a, b)
- data store: collection of data items

Operations on a Data Store:
- Read. $R_i(x)b$: Client $i$ performs a read for data item $x$ and it returns $b$
- Write. $W_i(x)a$: Client $i$ performs write on data item $x$ setting it to $a$
- Operations not instantaneous
  - Time of issue (when request is sent by client)
  - Time of execution (when request is executed at a replica)
  - Time of completion (when reply is received by client)
- Coordination among replicas

Replica Managers:

Timeline:
- Client A/Replica 1: $W(x)1, W(x)0$
- Client B/Replica 2: $R(x) -, R(x)1, R(x)1, R(x)0$
**CONSISTENCY**

Conflicting Data:
- Do replicas have exactly the same data?
- What differences are permitted?

Consistency Dimensions:
- Time and Order
  - Time:
    - How old is the data (staleness)?
    - How old is the data allowed to be?
  - Time, Versions

Operation order:
- Were operations performed in the right order?
- What orderings are allowed?

Real world examples of inconsistency?

---

**ORDERING**

Updates and concurrency result in conflicting operations

Conflicting Operations:
- Read-write conflict (only 1 write)
- Write-write conflict (multiple concurrent writes)
- The order in which conflicting operations are performed affects consistency

Partial vs Total Ordering:
- partial order: order of a single client’s operations
- total order: interleaving of all conflicting operations

---

Example:

Client A: $x = 1; x = 0;$
Client B: $\text{print}(x); \text{print}(x);$  
Possible results: $-- , 11, 10, 00$  
How about $01$?

What are the conflicting ops? What are the partial orders? What are the total orders?

---

Example:

Client A: $x = 1; x = 0;$
Client B: $\text{print}(x); \text{print}(x);$  
Possible results: $-- , 11, 10, 00$  
How about $01$?

What are the conflicting ops? What are the partial orders? What are the total orders?

---

Can you sanely use a system like this?
**Consistency Model**

Defines which interleavings of operations are valid (admissible)

Consistency Model:
- Concerned with consistency of a data store.
- Specifies characteristics of valid total orderings

A data store that implements a particular model of consistency will provide a total ordering of operations that is valid according to the model.

**Data Coherence vs Data Consistency:**

- **Data Coherence** ordering of operations for single data item
  - e.g. a read of x will return the most recently written value of x

- **Data Consistency** ordering of operations for whole data store
  - implies data coherence
  - includes ordering of operations on other data items too

**Non-distributed data store:**
- Data coherence is respected
- Program order is maintained

---

**Data-Centric Consistency Model**

A contract, between a distributed data store and clients, in which the data store specifies precisely what the results of read and write operations are in the presence of concurrency.

- Multiple clients accessing the same data store
- Described consistency is experienced by all clients
  - Client A, Client B, Client C see same kinds of orderings
- Non-mobile clients (replica used doesn’t change)

---

**Strong Ordering vs Weak Ordering**

**Strong Ordering (tight):**
- All writes must be performed in the order that they are invoked
- Example: all replicas must see: \( W(x)a \ W(x)b \ W(x)c \)
- Strict (Linearisable), Sequential, Causal, FIFO (PRAM)

**Weak Ordering (loose):**
- Ordering of groups of writes, rather than individual writes
- Series of writes are grouped on a single replica
- Only results of grouped writes propagated.
- Example: \( \{W(x)a \ W(x)b \ W(x)c\} \rightarrow \{W(x)a \ W(x)c\} \rightarrow \{W(x)c\} \)
- Weak, Release, Entry
**Strict Consistency**

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

- Absolute time ordering of all shared accesses.

What is most recent in a distributed system?
- Assumes an absolute global time.
- Assumes instant communication (atomic operation).
- Normal on a uniprocessor.
- Impossible in a distributed system.

---

**Sequential Consistency**

All operations are performed in some sequential order.
- More than one correct sequential order possible.
- All clients see the same order.
- Program order of each client maintained.
- Not ordered according to time.

Why is this good?

Performance:
- Read time + write time $\geq$ minimal packet transfer time.

---

**Linearisable Consistency**

All operations are performed in a single sequential order.
- Operations ordered according to a global (finite) timestamp.
- Program order of each client maintained.

---

**Causal Consistency**

Potentially causally related writes are executed in the same order everywhere.

Causally Related Operations:
- Read followed by a write (in same client).
- $W(x)$ followed by $R(x)$ (in same or different clients).

How could we make this valid?

---

**FIFO (PRAM) Consistency**
**FIFO (PRAM) Consistency**

*Only partial orderings of writes maintained*

- Client A
  - \( W(x) \)
  - \( R(x) \)
- Client B
  - \( W(x) \)
  - \( R(x) \)
- Client C
  - \( W(x) \) \( R(x) \)
- Client D
  - \( W(x) \) \( R(x) \)

FIFO consistent

How could we make this valid?

**Weak Consistency**

*Shared data can be counted on to be consistent only after a synchronisation is done*

- Enforces consistency on a *group of operations*, rather than single operations
  - Synchronisation variable (S)
  - Synchronise operation (synchronise(S))
  - Define ‘critical section’ with synchronise operations

**Properties:**
- Order of synchronise operations sequentially consistent
- Synchronise operation cannot be performed until all previous writes have completed everywhere
- Read or Write operations cannot be performed until all previous synchronise operations have completed

**Release Consistency**

*Explicit separation of synchronisation tasks*
- acquire(S) - bring local state up to date
- release(S) - propagate all local updates
- acquire-release pair defines ‘critical region’

**Properties:**
- Order of synchronisation operations are FIFO consistent
- Release cannot be performed until all previous reads and writes done by the client have completed
- Read or Write operations cannot be performed until all previous acquires done by the client have completed
Slide 29

What is an example of an invalid ordering?

Slide 30

Lazy Release Consistency:
- Don’t send updates on release
- Acquire causes client to get newest state
- Added efficiency if acquire-release performed by same client (e.g., in a loop)

Slide 31

**Entry Consistency**

Synchronisation variable associated with specific shared data item (guarded data item)
- Each shared data item has own synchronisation variable
- 
  - Provides ownership of synchronisation variable
  - Exclusive and nonexclusive access modes
  - Synchronises data
  - Requires communication with current owner
- release()
  - Relinquishes exclusive access (but not ownership)

Slide 32

Properties:
- Acquire does not complete until all guarded data is brought up to date locally
- If a client has exclusive access to a synchronisation variable, no other client can have any kind of access to it
- When acquiring nonexclusive access, a client must first get the updated values from the synchronisation variable’s current owner

Entry consistent
You can only choose two of C A or P
CAP Impossibility Proof:

**Slide 37**

Replica A → Replica B

Client

CAP Impossibility Proof:

**Goal:** Consistency and Availability

**No Partition:** It works!

**Slide 38**

Replica A → Replica B

Read → Write

Client

CAP Impossibility Proof:

**Partition:** no messages between A and B

**Slide 39**

Replica A ◼️ Replica B

Write

Client

CAP Impossibility Proof:

**Assume:** Availability and Partition Tolerance

**Slide 40**

Replica A ◼️ Replica B

Read

Write

No Consistency

Client
CAP Impossibility Proof:
Assume: Consistency and Partition Tolerance

Slide 41

CAP Impossibility Proof:
Assume: Consistency and Partition Tolerance

Slide 42

CAP Impossibility Proof:
Assume: Consistency and Partition Tolerance

Slide 43

CAP Consequences
For wide-area systems:
- Must choose: Consistency or Availability
- Choosing Availability
  - Give up on consistency?
  - Eventual consistency
- Choosing Consistency
  - No availability
  - Delayed (and potentially failing) operations

Why can’t we choose C and A and forget about P?

Slide 44

Eventual Consistency
If no updates take place for a long time, all replicas will gradually become consistent

Requirements:
- Few read-write conflicts (R > W)
- Few write-write conflicts
- Clients accept time inconsistency (i.e., old data)
- What about ordering?
Examples:

- **DNS:**
  - no write-write conflicts
  - updates slowly (1-2 days) propagate to all caches

- **WWW:**
  - few write-write conflicts
  - mirrors eventually updated
  - cached copies (browser or proxy) eventually replaced
  - manual merging for write-write conflicts

**CLIENT-CENTRIC CONSISTENCY MODELS**

*Provides guarantees about ordering of operations for a single client*

- Single client accessing data store
- Client accesses different replicas (modified data store model)
- Data isn’t shared by clients
- Client A, Client B, Client C may see different kinds of orderings

In other words:

- The effect of an operation depends on the client performing it
- Effect also depends on the history of operations that client has performed.

**DATA-STORE MODEL FOR CLIENT-CENTRIC CONSISTENCY:**

- Data-items have an owner
- No write-write conflicts

**NOTATION AND TIMELINE FOR CLIENT-CENTRIC CONSISTENCY:**

- $x_{i[t]}$: version of $x$ at replica $i$ at time $t$
- Write Set: $WS(x_{i[t]})$: set of writes at replica $i$ that led to $x(t)$
- $WS(x_{i[t1]};x_{j[t2]})$: $WS(x_{j[t2]})$ contains same operations as $WS(x_{i[t1]})$
- $WS(x_{i[t1]};x_{j[t2]})$: $WS(x_{j[t2]})$ does not contain the same operations as $WS(x_{i[t1]})$
- $R(x_{i[t]})$: a read of $x$ returns $x(t)$

```
<table>
<thead>
<tr>
<th>Replica 1</th>
<th>W(x1)</th>
<th>WS(x1)</th>
<th>R(x1)</th>
</tr>
</thead>
<tbody>
<tr>
<td>Replica 2</td>
<td>W(x1)</td>
<td>WS(x1)</td>
<td>W(x2)</td>
</tr>
</tbody>
</table>
```
**MONOTONIC READS**

If a client has seen a value of $x$ at a time $t$, it will never see an older version of $x$ at a later time.

<table>
<thead>
<tr>
<th>Replica 1</th>
<th>Replica 2</th>
</tr>
</thead>
<tbody>
<tr>
<td>$W(x_1)$</td>
<td>$R(x_1)$</td>
</tr>
<tr>
<td>$W(x_1;x_2)$</td>
<td>$R(x_2)$</td>
</tr>
</tbody>
</table>

When is Monotonic Reads sufficient?

**MONOTONIC WRITES**

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

All writes by a single client are sequentially ordered.

<table>
<thead>
<tr>
<th>Replica 1</th>
<th>Replica 2</th>
</tr>
</thead>
<tbody>
<tr>
<td>$W(x_1)$</td>
<td>$W(x_1;x_2)$</td>
</tr>
<tr>
<td>$W(x_2)$</td>
<td>$R(x_1)$</td>
</tr>
</tbody>
</table>

How is this different from FIFO consistency?

- Only applies to write operations of single client.
- Writes from clients not requiring monotonic writes may appear in different orders.

**READ YOUR WRITES**

The effect of a write on $x$ will always be seen by a successive read of $x$ by the same client.

<table>
<thead>
<tr>
<th>Replica 1</th>
<th>Replica 2</th>
</tr>
</thead>
<tbody>
<tr>
<td>$W(x_1)$</td>
<td>$W(x_1;x_2)$</td>
</tr>
<tr>
<td>$R(x_2)$</td>
<td>$R(x_1)$</td>
</tr>
</tbody>
</table>

When is Read Your Writes sufficient?

**WRITE Follows READS**

A write operation on $x$ will be performed on a copy of $x$ that is up to date with the value most recently read by the same client.

<table>
<thead>
<tr>
<th>Replica 1</th>
<th>Replica 2</th>
</tr>
</thead>
<tbody>
<tr>
<td>$W(x_1)$</td>
<td>$R(x_1)$</td>
</tr>
<tr>
<td>$W(x_1;x_2)$</td>
<td>$W(x_2)$</td>
</tr>
</tbody>
</table>

When is Write Follows Reads sufficient?
Choosing the Right Model

Trade-offs
Consistency and Redundancy:
- All copies must be strongly consistent
- All copies must contain full state
- Reduced consistency → reduced reliability

Consistency and Performance:
- Consistency requires extra work and communication
- Can result in loss of overall performance
- Weaker consistency possible

Consistency and Scalability:
- Implementation of consistency must be scalable
  - don’t take a centralised approach
  - avoid too much extra communication

Consistency Protocols

Consistency Protocol: implementation of a consistency model (e.g., sequential consistency)

Primary-Based Protocols:
- Remote-write protocols
- Local-write protocols

Replicated-Write Protocols:
- Active Replication
- Quorum-Based Protocols

Remote-Write Protocols

Single Server (single reader/single writer):
- All writes and reads executed at single server
- No replication of data

Primary-Backup (multiple reader/single writer):
- All writes executed at single server, Reads are local
- Updates block until executed on all backups
- Performance

Local-Write Protocols
**Local-Write Protocols**

Migration (single reader/single writer):
- Data item migrated to local server on access
- Performance (when not sharing data)

1. Read or write request
2. Forward request to current server for \( x \)
3. Move item \( x \) to client's server
4. Return result of operation on client’s server

**Active Replication**

- Updates (write operation) sent to all replicas
- Need totally-ordered multicast (for sequential consistency)
- e.g. sequencer/coordinator to add sequence numbers

**Quorum-Based Protocols**

- Voting
- Versioned data
- Read Quorum: \( N_r \)
- Write Quorum: \( N_w \)
- \( N_r + N_w > N \) Why?
- \( N_w > N/2 \) Why?
### Push vs Pull

<table>
<thead>
<tr>
<th>Pull</th>
<th>Push</th>
</tr>
</thead>
<tbody>
<tr>
<td>Pull: Updates propagated only on request</td>
<td>Push: Push updates to replicas</td>
</tr>
<tr>
<td>Also called client-based</td>
<td>Also called server-based</td>
</tr>
<tr>
<td>R/W low (W &gt; R)</td>
<td>When low staleness required</td>
</tr>
<tr>
<td>× Polling delay</td>
<td>R/W high (R &gt; W)</td>
</tr>
<tr>
<td></td>
<td>× Have to keep track of all replicas</td>
</tr>
</tbody>
</table>

### Push Update Propagation:
- **What to propagate?**
  - Data
  - R/W high
  - Update operation
    - low bandwidth costs
  - Notification/Invalidation
    - R/W low

### Compromise: Leases:
Server promises to push updates until lease expires
Lease length depends on:
- **age**: Last time item was modified
- **renewal-frequency**: How often replica needs to be updated
- **state-space overhead**: lower expiration time to reduce bookkeeping when many clients

### Replica Placement

- Server-initiated replication
- Client-initiated replication
Permanent Replicas:
- Initial set of replicas
- Created and maintained by data-store owner(s)
- Allow writes

Server-Initiated Replicas:
- Enhance performance
- Not maintained by owner
- May or may not allow writes
- Placed close to groups of clients
  - Manually vs Dynamically

Client-Initiated Replicas:
- Client caches
- Temporary
- Owner not aware of replica
- Placed close to client
- Maintained by host (often client)

Dynamic Replication
Situation changes over time
- Number of users, Amount of data
- Flash crowds
- R/W ratio

Dynamic Replica Placement:
- Network of replica servers
- Keep track of data item requests at each replica
- Thresholds:
  - Deletion threshold
  - Replication threshold
  - Migration threshold
- Clients always send requests to nearest server

Miscellaneous Implementation and Design Issues
End-to-End argument:
- Where to implement replication mechanisms?
- Application? Middleware? OS?

Policy vs Mechanism:
- Consistency models built into middleware?
- One-size-fits-all?

Determining Policy:
- Who determines the consistency model used?
  - Application, Middleware
  - Client, Server

Keep It Simple, Stupid:
- Will the programmer understand the consistency model?

Reading List

Eventual Consistency An overview of eventual consistency and client-centric consistency models.
Homework

Consistency Models:
- Research consistency models used in existing Distributed Systems
- Why are those models being used?
- In the systems you looked at, could other models have been used? Would that have made the system better?

Hacker’s Edition:
- Find a system that provides Eventual Consistency
  - (alternatively, implement (possibly in Erlang) a system that provides Eventual Consistency)
- Replicate some data and perform queries. How often do you get inconsistent results?
- If you can tweak replication parameters, how do they affect the consistency of results?