Slide 1

**Distributed Systems (COMP9243)**

Lecture 3a: Replication & Consistency

- Replication
- Consistency
  - Models vs Protocols
- Update propagation

---

**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 Replication (Server Replication/Mirroring):**

FTP Server

---

**Data Replication (Caching):**

Cache

---

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

- process request
- build web page

Apache + Perl

SQL Database

HTTP

Data and Control Replication:

- process request
- build web page

Apache + Perl

HTTP

SQL Database

HTTP

What are the challenges of doing this?

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:

Client A
Client B
Client C
Client D

Data Store

We will be looking primarily at data replication (including combined data and control replication).
Distributed Data-Store’s Point of View:

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
**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?
  - 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

---

**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: \( \{ W(x)a \ W(x)b \ W(x)c \} == \{ W(x)a \ W(x)c \} == \{ W(x)c \} \)
- 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
**Slide 21**

**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

![Diagrams showing linearisable vs not linearisable operations]

**Slide 22**

**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

![Diagrams showing sequential vs not sequential operations]

**Performance:**

read time + write time >= minimal packet transfer time

**Slide 23**

**Causal Consistency**

Potentially causally related writes are executed in the same order everywhere

- Read followed by a write (in same client)
- W(x) followed by R(x) (in same or different clients)

![Diagrams showing causally consistent vs not causally consistent operations]

**How could we make this valid?**

**Slide 24**

**FIFO (PRAM) Consistency**

Only partial orderings of writes maintained

- Not ordered according to time

![Diagrams showing FIFO consistent vs not FIFO consistent operations]

**How could we make this valid?**

**Causal Consistency** 11

**Weak Consistency** 12
**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 \( (\text{synchronise}(S)) \)
  - 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

**Example:**

- \( \text{synchronise}(S) \) \( W(x)a \) \( W(y)b \) \( W(x)c \) \( \text{synchronise}(S) \)
- Writes performed locally
- Updates propagated only upon synchronisation
- Only \( W(y)b \) and \( W(x)c \) have to be propagated

---

**Release Consistency**

*Explicit separation of synchronisation tasks*

- \( \text{acquire}(S) \) - bring local state up to date
- \( \text{release}(S) \) - propagate all local updates
- \( \text{acquire-release} \) 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

---

*How could we make this valid?*
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)

**Entry Consistency**

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

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

**CAP Theory**

You can only choose two of C A or P
CAP Theory

C: Consistency: Linearisability
A: Availability: Timely response
P: Partition-Tolerance: Functions in the face of a partition

You can only choose two of C, A, or P.

CAP Impossibility Proof:

Slide 36
CAP Impossibility Proof:

**Re**

No Partition Tolerance

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?

Eventual Consistency

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

- 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

Slide 45

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:

Slide 46

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

Slide 47

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 \( x_i[t] \)
- \( WS(x_i[t_1]; x_j[t_2]) \): \( WS(x_j[t_2]) \) contains same operations as \( WS(x_i[t_1]) \)
- \( WS(x_i[t_1]; x_j[t_2]) \): \( WS(x_j[t_2]) \) does not contain the same operations as \( WS(x_i[t_1]) \)
- \( R(x_i[t]) \): a read of \( x \) returns \( x_i[t] \)

Slide 48

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

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.

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.

**WRITE/FOLLOW 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.

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

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

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

REMOTE-WRITE PROTOCOLS

Single Server:
- All writes and reads executed at single server
- No replication of data

LOCAL-WRITE PROTOCOLS

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

- ✓ Performance for concurrent reads
- ❌ Performance for concurrent writes

**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: Nr
- Write Quorum: Nw
- Nr + Nw > N Why?
- Nw > N/2 Why?

**Push vs Pull**

- Pull:
  - Updates propagated only on request
  - Also called client-based
  - R/W low
  - Polling delay

- Push:
  - Push updates to replicas
  - Also called server-based
  - When low staleness required
  - R = W
  - Have to keep track of all replicas
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

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
- Placed close to groups of clients
  - Manually
  - 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?