CS 5523 Operating Systems: Consistency & Replication

Instructor: Dr. Tongping Liu

Thank Dr. Dakai Zhu and Dr. Palden Lama for providing their slides.

Objectives

- To understand replication and related issues in distributed systems
- To learn about how to keep multiple replicas consistent with each other

Outline

- Motivations for replications
  - Performance and/or fault-tolerance
- Data-Centric Consistency Models
  - Continuous Consistency, Consistent Ordering of Operations
- Client-Centric Consistency Models
  - Eventual Consistency
  - Monotonic Reads, Monotonic Writes
  - Read Your Writes, Writes Follow Reads
- Replica Management
  - Replica-Server Placement, Content Replication&Placement
  - Content Distribution
- Consistency Protocols
  - Implementation of the consistency models

Why Replications are Needed?

- Data are replicated
  - To increase the reliability of a system:
    - If one crash, we can switch to another one
    - Provide better protection on the data
  - To improve performance ➔ Scalability
    - Scaling in numbers and in geographical area (e.g., place copies of data close to the processes using them. So clients can quickly access the content.)
- Problems
  - How to keep replicas consistent
    - Distribute replicas
    - Propagate modifications
  - Cost >> benefit if access-to-update is very low
Replication as Scaling Technique

What if there is an update?
- **Update all in an atomic way** (sync replication)
- To keep replicas consistent → conflicting operations are done in the same order everywhere
  - Read-write conflict: read and write operations act concurrently
  - Write-write conflict: two concurrent write operations

Solution
- Loosen the consistency constraint so that hopefully global synchronization can be avoided

Outline

- **Motivations for replications**
  - Performance and/or fault-tolerance
- **Data-Centric Consistency Models**
  - Continuous Consistency, Consistent Ordering of Operations
- **Client-Centric Consistency Models**
  - Eventual Consistency
  - Monotonic Reads, Monotonic Writes
  - Read YourWrites, Writes Follow Reads
- **Replica Management**
  - Replica-Server Placement, Content Replication&Placement
  - Content Distribution
- **Consistency Protocols**
  - Implementation of the consistency models

Consistency Models

- Data-Centric
- Client-Centric

Data-Centric Consistency Model

Consistency Model: A contract between a (distributed) data store and processes: if processes agree to obey certain rules, the store promises to work correctly.

A data store is a distributed collection of storages:

Without a global clock, it is hard to define precisely which write is the last one! So we need other definitions [degree/range of consistency]
Continuous Consistency

- We can actually talk about a degree of consistency:
  - replicas may differ in their numerical value (e.g., price)
  - replicas may differ in their staleness (e.g., not too old)
  - differences with respect to the order of updation

- Examples
  - Replication of stock market prices (e.g., no more than $.02 or 0.5% difference between any two copies)
  - Duration of updates (e.g., weather reports stay accurate over some time, web pages)
  - Order of operations could be different (e.g., see next slide)

Consistent Ordering of Operations

- How to reach a global order of operations applied to replicated data so we can provide a system-wide consistent view on data store?

- Comes from concurrent programming
  - Sequential consistency
  - Causal consistency

W(x): a write by process P_i to data item x with the value a

Sequential Consistency (1)

- The result of any execution is the same as if the (R/W) operations of all processes were executed in some sequential order, and the operations of each individual process appear in this sequence in the order specified by its program – by Lamport

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

<table>
<thead>
<tr>
<th>Process</th>
<th>Operation</th>
<th>Time</th>
</tr>
</thead>
<tbody>
<tr>
<td>P1</td>
<td>W(x)a</td>
<td>0</td>
</tr>
<tr>
<td>P2</td>
<td>R(x)NIL</td>
<td>1</td>
</tr>
<tr>
<td></td>
<td>R(x)a</td>
<td>2</td>
</tr>
</tbody>
</table>

- It took sometime to propagate new value of x

Sequential Consistency (2)

- Any valid interleaving of R and W is acceptable as long as all processes see the same interleaving of operations.

- Everyone sees all W in the same order
Causal Consistency (1)

- Weakening sequential consistency
  - NOT all, only causally related W → seen in same order
- It implies:
  - 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.
- If event b is caused by an earlier event a, a → b
  - P1: Wx P2: Rx then Wy, then Wx → Wy (potentially causally related)

Causal Consistency (2)

- This sequence is allowed with a causally-consistent store, but not with a sequentially consistent store.

(a) A violation of a causally-consistent store
(b) Causally but not sequentially consistent event

Implementing causal consistency requires keeping track of which processes have seen which write → Construct a dependency graph using vector timestamps...

Outline

- Motivations for replications
  - Performance and/or fault-tolerance
- Data-Centric Consistency Models
  - Continuous Consistency, Consistent Ordering of Operations
- Client-Centric Consistency Models
  - Eventual Consistency
  - Monotonic Reads, Monotonic Writes
  - Read Your Writes, Writes Follow Reads
- Replica Management
  - Replica-Server Placement, Content Replication&Placement
  - Content Distribution
- Consistency Protocols
  - Implementation of the consistency models

Client-Centric Consistency Models

- Data-centric: aiming at providing a system-wide consistent view on a data store.
  - Assumption: processes can update simultaneously the data store, thus it is necessary to provide consistency
  - Sequential is good but costive, only guarantee when using transactions or locks.
- Client-centric: In some special data stores without simultaneous updates, client-centric consistency models can deal with inconsistencies in a less costly way: we only care about when updates happen
  - From a specific client point of view
Eventual Consistency (1)

- Most processes never perform updates while a few do updates.
- How fast updates should be made available to only reading processes (e.g., DNS)
  - Consider WWW pages, not write-write conflict
    - To improve performance clients cache web pages. Caches might be inconsistent with original page for some time...
    - Eventually all will be brought up to date
  - MongoDB, CouchDB, Amazon DynamoDB and SimpleDB
- Eventual consistency:
  - If no updates take place for a long time, all replicas will become consistent.

Eventual Consistency (2)

- As long as a client access the same replica, then there is no problem...
- But when the client (mobile one) accesses different replica, then we have a problem...

Example: 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 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

Eventual Consistency (3)

- In the previous example, the user may notice inconsistent behavior:
  - He expect to see what is made at A when he is at B.
- The problem can be alleviated by client-centric consistency.
  - It provides guarantees for a single client about the consistency of accesses to a data store.
- Four models:
  - All R & W are performed locally and eventually propagated to all
  - Data items have an associated owner which is permitted to modify data to avoid W-W conflicts.

Monotonic Reads

- If a process reads the value of a data item x, any successive read operation on x by that process will always return the same or a more recent value.

(a) A monotonic-read consistent data store.
(b) A data store that does not provide monotonic reads.

Example 1: Automatically reading your personal calendar updates from different servers. Monotonic Reads guarantees that the user sees all updates, no matter from which server the automatic reading takes place.

Example 2: Reading (not modifying) incoming mail while you are on the move. Each time you connect to a different e-mail server, that server fetches (at least) all the updates from the server you previously visited.
Monotonic Writes

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

(a) A monotonic-write consistent data store.

(b) A data store that does not provide monotonic-write consistency.

Example 1: Updating a program at server S2, and ensuring that all components on which compilation and linking depends, are also placed at S2.

Example 2: Maintaining versions of replicated files in the correct order everywhere (propagate the previous version to the server where the newest version is installed).

---

Read Your Writes

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.

Example: Updating your Web page and guaranteeing that your Web browser shows the newest version instead of its cached copy.

---

Writes Follow Reads

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.

Example: See reactions to posted articles only if you have the original posting (a read "pulls in" the corresponding write operation).

---

Outline

- Motivations for replications
  - Performance and/or fault-tolerance
- Data-Centric Consistency Models
  - Continuous Consistency, Consistent Ordering of Operations
- Client-Centric Consistency Models
  - Eventual Consistency
  - Monotonic Reads, Monotonic Writes
  - Read Your Writes, Writes Follow Reads
- Replica Management
  - Replica-Server Placement, Content Replication&Placement
  - Content Distribution
- Consistency Protocols
  - Implementation of the consistency models
Replica Management

Where, when, and by whom replicas should be placed and, which mechanisms to use for keeping them consistent?

Placement of replicas:
- Replica-Server Placement (find the best location)
- Content Replication and Placement (find the best servers)

Content distribution:
- Propagation of content to replica servers

Replica Server Placement

K out of N locations need to be selected:
- take distances between clients and possible locations and min distance.
- The K most dense cells are chosen for placing a server

Identify K largest clusters and assign a node from each cluster to host replicated content
- How to choose a proper cell size for server placement?

Content Replication and Placement

- Permanent replicas: Initial set of processes and machines always having a replica (web site mirrors)
- Server-initiated replica: for performance, replicas are created by the data store: installing replicas temporarily in regions that requests are coming from
- Client-initiated replica: a local storage that is used by a client locally (client cache)

Server-Initiated Replicas

Q counts access requests from different clients.
- If the number of requests is below a threshold, a file can be removed
- Otherwise, it is higher, we can move F to P.
Client-Initiated Replicas

- Client caches for improving the performance
  - Local storage, management is left to client
  - Keep it for a limited time
  - Caches can be shared between clients, assuming a data request of C1 is useful for nearby client C2

- Server-initiated is becoming more common than client-initiated…. Why?
  Shared cache is useless if some are rarely shared.
  Improvement of network and server make it good to share in common places.

Content Distribution

- How to **propagate the updated content** to the relevant replicas?

  - What is to be propagated:
    1. only a notification of an update (similar to cache coherence)
      - Invalidation protocols use notifications to inform others
      - + little network overhead
      - + good when \( W \gg R \) (r/w is small)
    2. Transfer data from one copy to another.
      - + good when \( W < R \) (r/w is high)
    3. Sending the update operations.
      - + little network overhead
      - - requires same computation power at each replica

  No single approach is the best, highly depends on available bandwidth and r/w ratio at replicas

Pull versus Push Protocols

- Pushing updates:
  - server-initiated, in which update is propagated regardless whether target asked for it. + good if r/w is high: read more

- Pulling updates:
  - client-initiated: + good if r/w is low: write more, read less

  We can dynamically switch between pulling and pushing using **leases** (a hybrid form):
  - Lease is a contract in which the server promises to push updates to clients until the lease expires.

Unicast vs. Multicast

- Unicast: N separate send
- Multicast: one send to N servers

- Pull-based--- unicast
- Push-based-- multicast
Outline

- Motivations for replications
  - Performance and/or fault-tolerance
- Data-Centric Consistency Models
  - Continuous Consistency, Consistent Ordering of Operations
- Client-Centric Consistency Models
  - Eventual Consistency
  - Monotonic Reads, Monotonic Writes
  - Read Your Writes, Writes Follow Reads
- Replica Management
  - Replica-Server Placement, Content Replication & Placement
  - Content Distribution
- Consistency Protocols
  - Implementation of the consistency models

Implementation of Consistency Protocols

- Describes the implementation of a specific consistency model

  Data-centric
  - Continuous consistency
  - Primary-based protocols: sequential consistent
  - Replicated-write protocols
  - Cache-coherence protocols

  Client-centric Consistency

Primary-based protocols

- Mostly for sequential consistency
- Idea: each data item x in the data store has an associated primary, which is responsible for coordinating write operations on x.
- Subtypes: whether the primary is fixed at a remote server or if writes can be performed locally after moving the primary to the process where the write operation is initiated

Primary-Based Protocol: Remote-Write

- Remote-Write Protocol: Primary-Backup
  - All Ws are forwarded to a fixed single server, but reads can perform locally

  Problem: the initial process wait a while to proceed
  Non-blocking: proceed after updating primary, but fault tolerant
Primary-Based Protocol: Local-Write

- Local-Write Protocol: the primary copy migrates
  - + can be used in non-blocking protocol and updates will be propagated
  - + successive writes can perform locally

Summary

- Motivations for replications
- Data-Centric Consistency Models
  - Continuous Consistency, Consistent Ordering of Operations
- Client-Centric Consistency Models
  - Eventual Consistency
  - Monotonic Reads, Monotonic Writes
  - Read Your Writes, Writes Follow Reads
- Replica Management
  - Replica-Server Placement, Content Replication&Placement
  - Content Distribution
- Consistency Protocols
  - implementation of the consistency models