Consistency & Replication

527950-1
Fall 2019
12/5/2019
Kyoung Shin Park
Applied Computer Engineering
Dankook University

Consistency & Replication

- Consistency & Replication:
  - What are the **reasons for replicating**, data, objects, or code?
  - How can we make sure that this **replicated data is consistent**?
  - What do we do we mean by **the term consistency**?
    And is there more than one **type of consistency**?
  - What do we mean by **client-centric consistency models**?
  - How can we **implement the replication** of data?
    And what are the **advantages and disadvantages** of different methods?
  - How can we **replicate** data and still have it be **consistent**?
  - **Examples of consistent systems with replicated data**

Replication

- Replication
  - A notable latency avoidance technique is ***replication***.
  - Normally, when several processes need access to the same data, the data are transferred back and forth, which is inefficient.
  - With replication, multiple copies of the data are held in different locations, and different processes work with different copies.
  - The idea of replication can be extended to computations; in this case the same computation is run on multiple nodes, which saves the communication of results.

Replication

- Replication
  - A well-known example of replication is ***caching***, the insertion of an additional fast memory level that holds **frequently used data**.
  - Caching is not only used in the design of basic computer architectures, **it is also a technique for speeding up the web**.
  - The **drawback** of any form of replication is the necessity to **maintain consistency of the replicated data**.

Replication

- Replication
  - A common requirement when data are replicated is for ***replication transparency***.
    - **That is, clients should *not* normally have to be aware that multiple *physical* copies of data exist.**
  - As far as clients are concerned, **data are organized as individual *logical* objects** (or objects) and they identify only one item in each case when they request an operation to be performed.
  - Furthermore, clients expect operations to return only one set of values.
    - This is despite the fact that operations may be performed upon more than one physical copy in concert.

Replication

- Replication
  - The other general requirement for **replicated data** -- one that can vary in strength between applications -- is that of ***consistency***.
  - This concerns whether or not the operations performed upon a collection of replicated objects produce results that meet the specification of correctness for those objects.

Replication

- Replication
  - The fundamental problem in managing replicated data is to maintain the consistency of the data.
  - In a local sense, a query on the data should return the data value that was 'most recently written'.
  - In a global sense, the interaction of a program with the collection of all global data should obey a global consistency constraint.
  - A primary motivation for replicating data is ***fault tolerance***.

Reasons for Replication

- The **advantages** of replication include the following:
  - **Increased availability**
  - **Increased reliability**
  - Improved response time
  - Reduced network traffic
  - Improved system throughput
  - Better scalability
  - Autonomous operation

Problems with Replication

- The **problems** with replication are:
  - Having multiple copies may lead to **consistency** problems.
    - Whenever a copy is modified, that copy becomes different from the rest. Consequently, modifications have to be carried out on all copies to ensure consistency. Exactly when and how those modifications need to be carried out determines the price of replication.
  - **Cost of increased bandwidth** for maintaining replication.

Replication versus Caching

- **Replication** is often confused with **caching**, probably because they both deal with multiple copies of a data.
- However, the two concepts have the following basic differences:
  1. A **replica** is associated with a **server**, whereas a **cached copy** is normally associated with a **client**.
  2. The existence of a cached copy is primarily dependent on the locality in file access patterns, whereas the existence of a **replica** normally depends on **availability and performance requirements**.
  3. As compared to a cached copy, **a replica is more persistent, widely known, secure, available, complete, and accurate.**
  4. **A cached copy is contingent upon a replica**. Only by periodic revalidation with respect to a replica can a cached copy be useful.

Replication as a Scaling Technique

- Replication and caching are widely used in scaling technique, but:
  - Keeping replicas up to date needs **networks use**.
  - Update needs to be **atomic (transaction)**
  - Replicas need to be **synchronized (time consuming)**

Loose Consistency

In this case copies are not always the same everywhere.

Consistency

- Consistency
  - ***Consistency*** is more **difficult** to achieve in a **distributed system**.
  - The lack of global information, potential replication and partitioning of data, the possibility of component failures, and the complexity of interaction among modules all contribute to the problem of inconsistency in the system.
  - **A system is consistent from the user's perspective if there is uniformity in using the system and the system behavior is predictable.**
  - In addition, the system must be capable of maintaining its integrity with proper concurrency control mechanisms and failure handling and recovery procedures.
  - **Consistency control** in data and files (or database in a transaction-oriented system) is a crucial issue in distributed file systems.

Consistency

- Consistency
  - ***Consistency*** is a term used to describe the function of **ensuring all data copies are the same and correct**.
  - There are three popular methods for realizing cache consistency.
    - The first employs software to enforce critical regions, protected regions of code where a given process is changing shared data.
    - The second utilizes software to prevent a processor from ever caching shared memory.
    - The third method for maintaining cache consistency is referred to as ***snoopy cache*** ... every processor constantly snoops or monitors the shared bus, relying on the fact that all processors are connected via a common bus.

Consistency

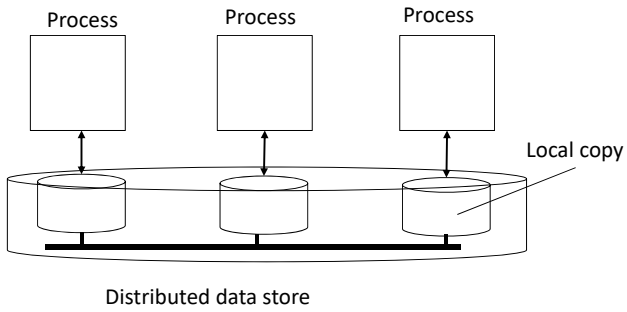
- Consistency
  - A ***consistency model*** basically refers to the degree of consistency that has to be maintained for the shared-memory data for the memory to work correctly for a certain set of applications.
  - It is defined as a set of rules that applications must obey if they want the DSM system to provide the degree of consistency guaranteed by the consistency model.

Consistency Models

- Data-Centric Consistency Models
  - Continuous Consistency
  - Consistent Ordering of Operations
    - **Strict consistency**, **sequential consistency**, **causal consistency, FIFO consistency**
    - **Grouping ordering of Operation: Weak, Release, and Entry**.
- Client-Centric Consistency Models
  - Eventual Consistency
  - Monotonic Reads
  - Monotonic Writes
  - Read Your Writes
  - Writes Follow Reads

Data-Centric Consistency Model

- Data-Centric Consistency Model
  - A contract between a distributed data store and processes, in which the data store specifies precisely what the results of read and write operations are in the presence of concurrency.



Type of Consistency Model

- ***Strict*** Consistency Model
- ***Sequential*** Consistency Model
- ***Casual*** Consistency Model
- ***FIFO*** Consistency Model/***Pipelined Random-Access Memory (PRAM)*** Consistency Model
- ***Weak*** Consistency Model
- ***Release*** Consistency Model
- ***Entry*** Consistency Model
- ***Processor*** Consistency Model
- ***General*** Consistency Model

Strict Consistency Model

- The ***strict consistency model*** is the strongest form of memory coherence, having the **most stringent consistency** requirements.
- A shared-memory system is said to support the *strict consistency model* if the value returned by a **read** operation on a memory address is always the same as the value written by the **most recent write** operation to that address, irrespective of the locations of the processes performing the read and write operations.
- That is, **all writes instantaneously become visible to all processes.**

Strict Consistency Model

- Any **read** on a data item returns a value corresponding to the result of the **most recent write**.
- Two operations in the same time interval are said to be ***in conflict*** if they operate on the **same data** and one of them is a **write** operation.
- Strict consistency is the ideal model but it is ***impossible*** to implement in a distributed system. It is based on *absolute global time* or a *global agreement on commitment of change.*

Strict Consistency Model

Wi(x)a – Write by process Pi to data item x with the value a.
Ri(x)b – Read from that item by Pi to returning b.
Assume that each data item is initially NIL.

- Behavior of two processes, operating on the same data item. The horizontal axis is time.
  a. A **strictly consistent** data store.
  b. A data store that **is not strictly consistent**.

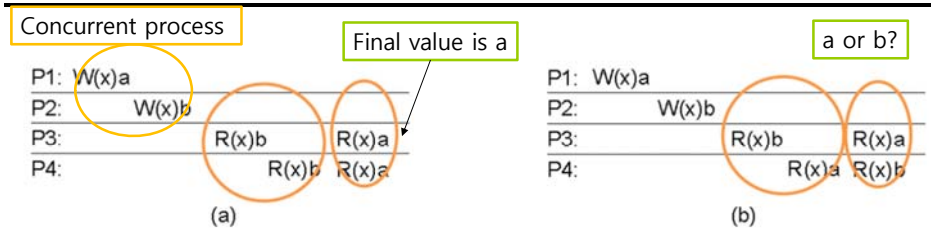
Sequential Consistency Model

- The ***sequential consistency model*** was proposed by Lamport (1979)
- A shared-memory system is said to support the *sequential consistency model* if **all processes see the same order** of all memory access operations on the shared memory.
- The exact order in which the memory access operations are interleaved does not matter.
- If **one** process sees **one** of the orderings of ... three operations and **another** process sees a **different** one, the memory is **not a sequentially consistent** memory.

Sequential Consistency Model

- Sequential Consistency it is a weaker consistency model than strict consistency.
- 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.
- When **processes run concurrently** on (possibly) different machines, **any valid interleaving of read and write operations is acceptable** behavior.
- **But, all processes see the same interleaving of operations.**

Sequential Consistency Model



- No reference to the timing of the operations
  a. A **sequentially consistent** data store.
  b. A store that is **not sequentially consistent**.

Casual Consistency Model

- The *causal consistency model* (by Hutto and Ahamad, 1990) ... relaxes the requirement of the sequential model for better concurrency.
- Unlike the sequential consistency model, in the causal consistency model, all processes see only those memory reference operations in the same (correct) order that are **potentially causally related**.
- Memory reference operations that are not potentially causally related may be seen by different processes in different orders.

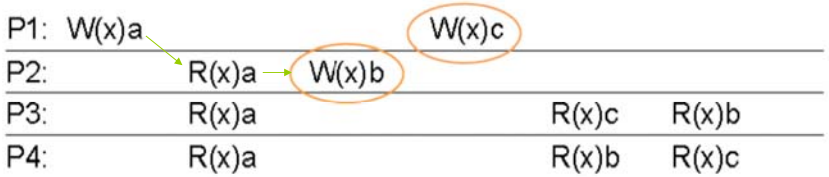
Casual Consistency Model

- When there is a **read followed by** a **write**, the two events are ***potentially causally related***.
- Operation **not causally related** are said **concurrent**.
- The same idea in multi-processors:
  - **Writes that are potentially causally related** must be seen by **all processors** in the **same order.**
  - **Concurrent writes** may be seen in a **different order** on **different machines:**
  - **Causally related writes**: the write comes after a read that returned the value of the other write

Casual Consistency Model

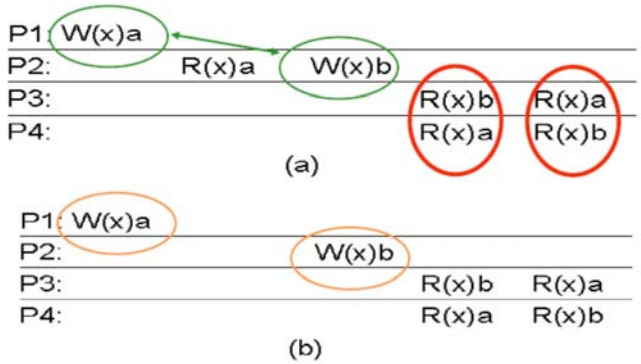
- Necessary condition:
  - Writes that are **potentially causally related** must be seen by **all processes in the same order**.
  - **Concurrent** writes may be seen in a **different order** on **different machines**.
- We already came across causality when discussing vector timestamps.
  - **a → b**, If event b is caused or influenced by an earlier event a, causality requires that everyone else first see a, then see b.

Casual Consistency Model



- This sequence is allowed with a **causally-consistent** data store, but **not with sequentially or strictly** consistent data store.
- Note that the writes $W_2(x)b$ and $W_1(x)c$ are **concurrent**.
- Causal consistency requires **keeping tracks of which processes have seen which writes.**

Casual Consistency Model



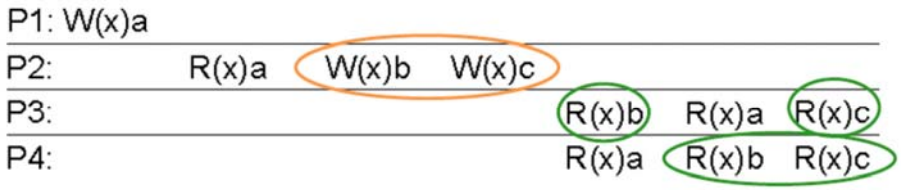
a. A **violation of a casually-consistent** data store. $W_2(x)b$ may be **related to** $W_1(x)a$ because the b may be a result of a computation involving the value read by $R_2(x)a$. The two writes are causally related, so all processes must see them in the same order.

b. A **correct sequence of events** in a casually-consistent data store. $W_1(x)a$ and $W_2(x)b$ are **concurrent** and no need to be globally ordered. This situation would **not be acceptable by sequentially consistent** store.

FIFO Consistency Model

- **Relaxing** consistency requirements we **drop causality**
- Necessary condition:
  - Writes done by a ***single* process** are seen by all other processes in the order in which they were issued,
  - but writes from different processes may be seen in a different order by different processes.
- **All writes** generated by **different processes** are considered **concurrent**. It is easy to implement.
- ***FIFO* consistency** is called ***PRAM consistency*** in the case of distributed shared memory systems.

FIFO Consistency Model



- A **valid** sequence of events of **FIFO consistency**. It is **not valid for causal consistency**

PRAM Consistency Model

- ***Pipelined Random-Access Memory (PRAM)*** Consistency Model provides a weaker consistency semantics than the *(first three)* consistency models described so far.
- It only ensures that **all write** operations performed by a **single process** are seen by **all other processes** in **the order** in which they were performed as if all the write operations performed by a single process are in a pipeline.
- **Write** operations performed by **different processes** may be seen by **different processes** in **different orders**.

Weak Consistency Model

- **Synchronization accesses** (accesses required to perform synchronization operations) are **sequentially consistent**.
- Before a synchronization access can be performed, all previous regular data accesses must be completed.
- Before a regular data access can be performed, all previous synchronization accesses must be completed.
- This essentially leaves the problem of consistency up to the programmer.
- The memory will only be **consistent** immediately **after a synchronization** operation.

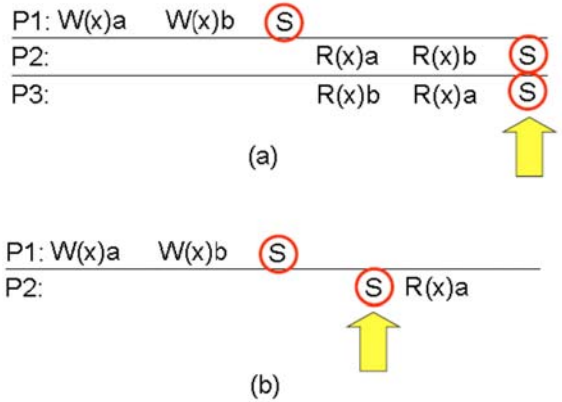
Weak Consistency Model

- We can release the requirements of **writes within the same process** seen in order everywhere introducing a **synchronization variable**.
- A synchronization operation synchronize all local copies of the data store.
- **Weak consistency** uses **synchronization variables** to propagate **writes** to and from a machine at appropriate points.
- It forces **consistency on a group of operations**, not on individual write and read. It limits the time when consistency holds, not the form of consistency.

Weak Consistency Model

- Properties of weak consistency:
  - **Accesses to synchronization variables associated with a data store are sequentially consistent.**
  - **No operation on a synchronization variable is allowed to be performed until all previous writes have been completed everywhere.**
  - **No read or write operation on data items are allowed to be performed until all previous operations to synchronization variables have been performed**.
- The same idea is used in multi processors:
  - Accessing a synchronization variable "flushes the pipeline".
  - At a **synchronization** point, all processors have **consistent** versions of data.

Weak Consistency Model



Convention: **S** means access to synchronization variable

a. A **valid** sequence of events for **weak consistency**.
b. Tanenbaum says the second is **not weakly consistent**. Do you agree? (What is the order of synchronizations? How do the rules prevent this execution?).

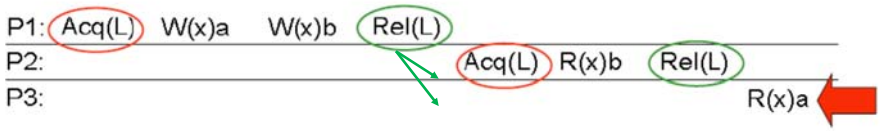
Release Consistency Model

- ***Release consistency*** is essentially the **same as weak consistency**, but synchronization accesses must only be **processor consistent** with respect to each other.
- Synchronization operations are broken down into ***acquire*** and ***release*** operations.
- All pending acquires (e.g., a lock operation) must be done before a release (e.g., an unlock operation) is done.
- Local dependencies within the same processor must still be respected.
- **Release consistency** is a further **relaxation** of **weak consistency** without a significant loss of coherence.

Release Consistency Model

- If it is possible to know the difference between **entering** a critical region or **leaving** it, a more efficient implementation might be possible. To do that, two kinds of synchronization variables are needed.
- Release consistency : **acquire** operation to tell that a critical region is being entered; **release** operation when a critical region is to be exited.

Release Consistency Model



- A **valid** event sequence for release consistency.
- Shared data kept consistent are called **protected**.

Release Consistency Model

- The same idea in multi-processors
  - **Release consistency** is **like weak consistency**, but there are two operations "**lock**" and "**unlock**" for **synchronization**:
  - "**Acquire/release**" are the conventional names
  - Doing a "**lock**" means that writes on other processors to protected variables will be known
  - Doing an "**unlock**" means that writes to protected variables are exported
  - And will be seen by other machines when they do a "**lock**" (lazy release consistency) or immediately (eager release consistency)

Release Consistency Model

- Rules:
  - **Before a read or write operation on shared data is performed, all previous acquires done by the process must have completed successfully.**
  - **Before a release is allowed to be performed, all previous reads and writes by the process must have completed.**
  - **Accesses to synchronization variables are FIFO consistent (sequential consistency is not required).**
- Explicit acquire and release calls are required.

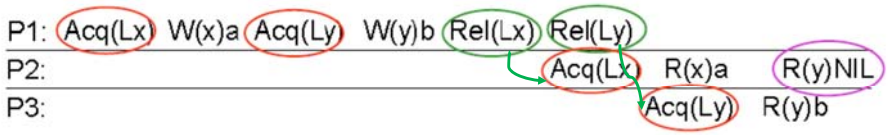
Entry Consistency Model

- Like ... variants of release consistency, it requires the programmer (or compiler) to use **acquire** and **release** at **the start and end of each critical section**, respectively.
- However, unlike release consistency, ***entry consistency*** requires each ordinary shared data item to be associated with some synchronization variable, such as a lock or barrier.
- If it is desired that elements of an array be accessed independently in parallel, then different array elements must be associated with different locks.
- When an **acquire** is done on a synchronization variable, **only those data guarded by that synchronization** variable are made **consistent**.

Entry Consistency Model

- Conditions:
  - **An acquire access of a synchronization variable is not allowed to perform with respect to a process until all updates to the guarded shared data have been performed with respect to that process.**
  - **Before an exclusive mode access to a synchronization variable by a process is allowed to perform with respect to that process, no other process may hold the synchronization variable, not even in nonexclusive mode.**
  - **After an exclusive mode access to a synchronization variable has been performed, any other process's next non-exclusive mode access to that synchronization variable may not be performed until it has performed with respect to that variable's owner.**

Entry Consistency Model



- A **valid** event sequence for **entry consistency**.
- **Lock are associated with each data item**

Processor Consistency Model

- **Writes** issued by a processor are observed in the **same order** in which they were issued.
- However, the order in which writes from two processors occur, as observed by themselves or a third processor, need not be identical.
- That is, **two simultaneous reads** of the same location from different processors may yield different results.

General Consistency Model

- A system supports ***general consistency*** if all the copies of a memory location eventually contain the same data when all the writes issued by every processor have completed.

Summary of Consistency Models

Consistency	Description
Strict	**Absolute time ordering** of all shared accesses matters.
Sequential	All processes see all shared accesses **in the same order**. Accesses are not ordered in time.
Casual	All processes see **causally-related** shared accesses in the same order.
FIFO	All processes see **writes** from each other **in the order they were used**. Writes from different processes may not always be seen in that order.

Consistency	Description
Weak	Shared data can be counted on to be consistent **only after a synchronization is done**.
Release	Shared data are made consistent **when a critical region is exited.**
Entry	Shared data pertaining to a critical region are made consistent **when a critical region is entered**.

ACID (Atomicity, Consistency, Isolation, Durability)

- Related to the consistency of a distributed system (or database system) are the ***ACID*** properties.
- The ***ACID (Atomicity, Consistency, Isolation, Durability) properties*** are primarily concerned with achieving the ***concurrency transparency*** goal of a distributed system
  - A property that allows sharing of objects without interference.
  - In a sense, the execution of a transaction appears to take place in a critical section.
  - However, operations from different transactions are interleaved (in some 'safe' way) to gain more concurrency.

ACID (Atomicity, Consistency, Isolation, Durability)

- ***Atomicity***
  - Either **all of the operations** in a transaction are performed or **none of them** are, in spite of failures.
- ***Consistency***
  - The execution of interleaved transactions is equivalent to a **serial execution** of the transactions in some order.
- ***Isolation***
  - **Partial results** of an incomplete transaction are **not visible** to others before the transaction is successfully committed.
- ***Durability***
  - The system guarantees that the **results of a committed transaction** will be made **permanent** even if a failure occurs after the commitment.

ACID (Atomicity, Consistency, Isolation, Durability)

- ***ACID properties***
  - Since all four properties are related to *consistency*, sometimes it is preferable to call the second property ***serializability*** to differentiate it from the others.
  - **Atomicity** refers to the consistency of replicated or partitioned objects.
  - **Violation of isolation** is seeing something that has never occurred, and
  - **Violation of durability** is not seeing something that has actually occurred.
  - Both are **inconsistent** perceptions of the system state.

References

- http://www.cs.colostate.edu/~cs551/CourseNotes/Consistency/ReplIndex.html