Chapter 7 Consistency and Replication

Why Replication?

• Replication involves creating and maintaining copies of data at multiple locations
• Replication improves availability
  – If one replica is unavailable or crashes, use another
• Replication improves performance
  – Replicating data reduces load on individual servers → improves response time and size scalability
  – Placing a copy of data close to the clients improves response time and geographical scalability
• Price to be paid: maintaining **consistency** of replicated data
  – When one copy is updated, the other copies must be updated too → this consumes network bandwidth
Consistency Issues

• When to replicate?
  – Suppose a process $P$ accesses a local replica $R$ times per second, and the replica is updated $W$ times per second
  – Should not install a replica close to $P$ if $R << W$ – high consistency overhead and wasted update messages
• “Tight” consistency requirement: all the replicas are always the same
  – Update must be performed at all copies as a single atomic operation
  – Requires global synchronization - difficult to achieve
• The solution is to loosen the consistency requirements
  – Data-centric consistency models
  – Client-centric consistency models

Distributed Data Store

• A data store is a repository of shared data (e.g., databases, file systems)
• A data store consists of multiple servers each maintaining a copy of all the data items stored in the data store
• Multiple processes can simultaneously access shared data; a process always access a single local (or nearby) copy of the store
  – Reads are performed locally
  – Writes are performed locally first and then propagated to the other copies
Data-Centric Consistency Models

• Strict consistency: a read returns the result of the last write; this is Impossible to implement due to
  – absence of a global clock
  – propagation delay of update messages
• In a distributed data store, write operations can be performed in different orders at different replicas
• A data-centric consistency model is a contract between a distributed data store and processes, in which the data store specifies the order in which the write operations are seen by processes

Data-Centric Consistency Models

• Not using synchronization operations
  – Sequential consistency
  – Causal consistency
  – FIFO consistency
• Using synchronization operations
  – Release consistency
  – Entry consistency
Sequential Consistency

• A data store is **sequentially consistent** when it satisfies the following conditions
  – The result of any execution is the same as if the (read and write) operations by all the processes on the data store were executed in some sequential order
  – The operations of each individual process appear in this sequence in the program order
• All processes see all write operations in the same order!

(a) A sequentially consistent data store. (b) A data store that is not sequentially consistent.

Sequential Consistency: An Example

Four valid execution sequences for the three concurrently executing processes. (Note: Not all 64 signature patterns are legal. For example, 001001 is not legal.)

The processes must accept all valid results as proper answers and work correctly if any of them occurs.
Causal Consistency

- A data store is **causally consistent** when it satisfies the following conditions
  - Writes that are causally related must be seen by all processes in the same order
  - Concurrent writes may be seen in a different order on different machines
- Two operations are **causally related** if
  - A read is followed by a write in the same process
  - A write of a data item is followed by a read of that data item in any process
- Operations that are not causally related, even through other operations, are said to be **concurrent**
- Causal consistency is weaker than sequential consistency

Causal Consistency: Examples

| P1: W(x)a | | W(x)c |
| P2: R(x)a | W(x)b |
| P3: R(x)a | R(x)c | R(x)b |
| P4: R(x)a | R(x)b | R(x)c |

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

| P1: W(x)a | | W(x)b |
| P2: R(x)a | | |
| P3: R(x)b | R(x)a | |
| P4: R(x)a | R(x)b | |

(a) A violation of a causally-consistent store.

| P1: W(x)a | | W(x)b |
| P2: | | |
| P3: | R(x)b | R(x)a |
| P4: | R(x)a | R(x)b |

(b) A correct sequence of events in a causally-consistent store.
FIFO Consistency

• A data store is **FIFO consistent** when it satisfies the following conditions
  – Writes of a single process are seen by all other processes in the order in which they were issued
  – Writes of different processes may be seen in a different order by different processes (even if causally related)

• FIFO consistency is weaker than causal consistency

| P1 | W(x)a | W(x)c |
| P2 | R(x)a | W(x)b |
| P3 | R(x)a | R(x)b | R(x)c |
| P4 | R(x)b | R(x)a | R(x)c |

| P1 | W(x)a | W(x)c |
| P2 | R(x)a | W(x)b |
| P3 | R(x)a | R(x)b | R(x)c |
| P4 | R(x)c | R(x)a | R(x)b |

FIFO consistent

Not FIFO consistent

Release Consistency

• Two kinds of synchronization operations are used: **acquire** and **release**

• A process **acquires** a lock before entering a critical section and **releases** the lock when leaving the critical section

• A release followed by an acquire of the same lock guarantees that all writes previous to the release will be seen by all reads following the acquire
Release Consistency: An Example

A valid event sequence for release consistency. *acquire* and *release* are illustrated as *Acq(L)* and *Rel(L)*, respectively.

Entry Consistency

- Each synchronization variable has some associated data items called **guarded data items**
  - To write to a guarded data item a process must acquire that item’s synchronization variable in exclusive mode
  - To read a guarded data item a process must acquire that item’s synchronization variable in nonexclusive mode
- When performing an acquire on a synchronization variable S, the current values of the data items associated with S are fetched from the owner of S (i.e., the last process that performed an exclusive acquire on S)
- A process releases the synchronization variable when it is done
Entry Consistency: An Example

P1  Acq(Lx)  W(x)a  Acq(Ly)  W(y)b  Rel(Lx)  Rel(Ly)

P2  Acq(Lx)  R(x)a  R(y)  NIL

P3  Acq(Ly)  R(y)  b

A valid event sequence for entry consistency.