Systèmes et Applications Répartis

Fundamentals – Part Three

Professeur Olivier Gruber

Université Joseph Fourier

Projet SARDES (INRIA et IMAG-LSR)

Message Fundamentals

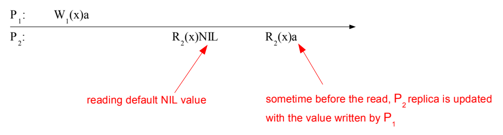
- Previous Lectures
  - Discussing messages
    - How do we name the destination?
    - How do we route the message?
  - Discussing time
    - What notion of time do we have?
    - How do we synchronize activities?
- Today's Lecture
  - Discussing memory models...
    - When accessing something means acquiring a copy through messages
    - When updating a local copy requires sending notification messages

Outline

- Memory consistency models
  - Distributed systems have no central consistent memory
    - Natural consistency (sequential consistency) is expensive to implement
  - Search for different compromises
    - Sequential, causal or eventual consistency
    - Integrating consistency and synchronization
  - Memory consistency protocols
    - A few protocols such primary-based and replicated-write protocols

Useful Notations

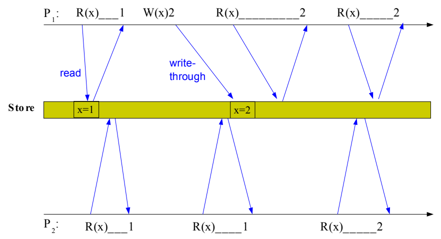
- Notations
  - Read operation at process ***P**$_i$* on data ***x*** returning the value ***a***
    - $R_i(x)a$
  - Write operation at process ***P**$_i$* on data ***x*** writing the value ***b***
    - $W_i(x)b$
  - Time flows from left to write
  - All data items are initialized to *NIL*



P_1:
$W_1(x)a$
P_2:
$R_2(x)NIL$
$R_2(x)a$
reading default NIL value
sometime before the read, P_2 replica is updated with the value written by P_1

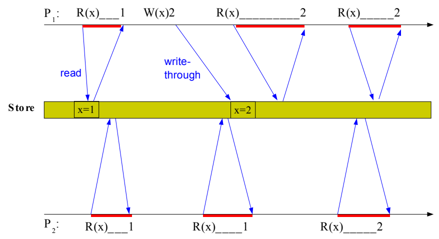
Shared Store

- Processes
  - We have N processes
  - We have a shared data store with data items


P_1: R(x)___1 W(x)2 R(x)_________2 R(x)_____2
read
write-through
Store
x=1
x=2
P_2: R(x)___1 R(x)____1 R(x)_____2

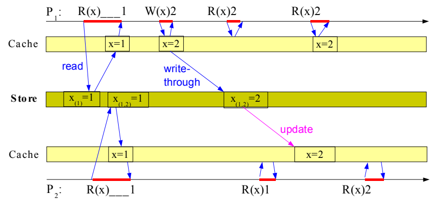
Shared Store

- Discussion
  - Like a normal shared memory, requires the use of synchronization
  - Poor performance (latencies)


P_1: R(x)___1 W(x)2 R(x)_________2 R(x)_____2
read
write-through
Store
x=1
x=2
P_2: R(x)___1 R(x)____1 R(x)_____2

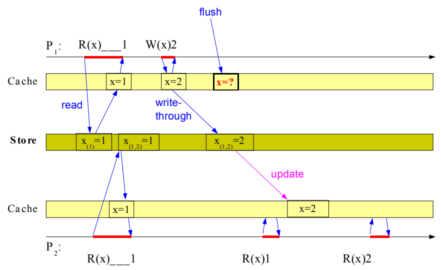
Shared Store

- Introducing local caches
  - One local cache per process
  - Shorter latencies, but we need a consistency protocol
    - Store must remember who has a cached copy of each data item
    - Must be able to callback caches to install updates



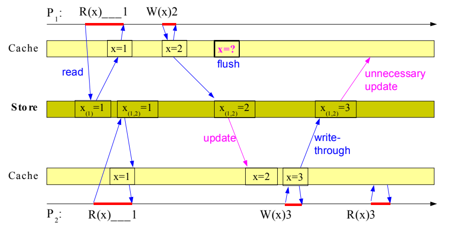
Shared Store

- Introducing cache flushes
  - Caches may overflow, we need to flush some cached data items



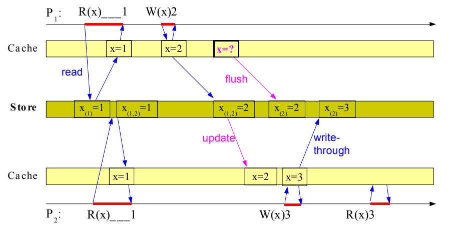
Shared Store

- Cache Design
  - How do we maintain the store copy lists?
  - So to avoid unnecessary update messages



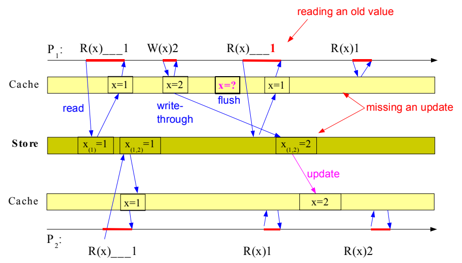
Shared Store

- Cache Design
  - Maintaining the store copy lists
    - Background messages
    - Time-To-Live caches (**watch for time skew**)



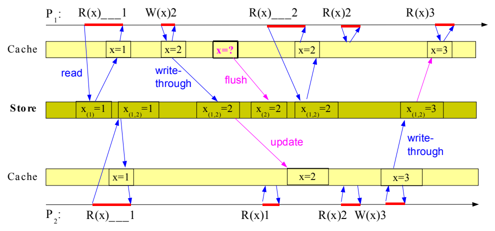
Shared Store

- Cache Design
  - Use FIFO communication channels!
    - With TCP/IP, requires to keep sockets open
    - With UDP, you need to implement FIFO/lossless



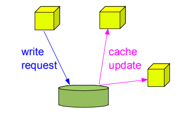
Shared Store

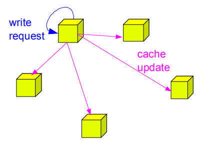
- Overview
  - Shared data store
  - Cache with flush and consistency protocol over FIFO channels



Distributed Store

- Shared Data Store
  - Once shared store
  - Multiple processes with local caches
- Distributed Data Store
  - Multiple processes with local caches
  - No materialized shared store at any one site
  - Implemented through distributed protocols
- Consistency Challenge
  - Maintaing consistency incurs a certain overhead
- Dilema
  - The stronger the consistency, the easier to use the distributed system, but the more expensive the consistency protocol





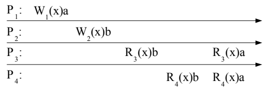
Consistency Models

- Consistency Definition
  - Essentially a contract between processes and a data store
  - Each model defines correctness rules
  - Each model defines operations and a data unit of consistency
- Different Models
  - Trying to maximize both performance and usability
    - Using domain-specific knowledge
  - Three dimensional space:
    - Numerical deviation
      - Allowing different replicas to differ numerically
      - E.g. precise temperature sensors but replicas have degree-rounded temperatures
    - **Staleness deviation**
      - Allowing different replicas to have less recent values
      - E.g. DNS, in-network web caches or out-of-date seat availability (booking systems)
    - **Ordering deviation**
      - Allowing different replicas to execute read/write operations in different orders

Sequential Consistency

- Defined by Lamport (1979)
  - Memory works as expected with multiple processes

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.



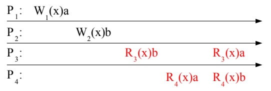
sequentially consistent

possible equivalent sequential order:

$\mathbf{W}_2(x)b$ $\mathbf{R}_3(x)b$ $\mathbf{R}_4(x)b$ $\mathbf{W}_1(x)a$ $\mathbf{R}_3(x)a$ $\mathbf{R}_4(x)a$

Sequential 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.



not sequentially consistent

no possible equivalent sequential order:

$\mathbf{W}_2(x)b$ $\mathbf{R}_3(x)b$ $\mathbf{W}_1(x)a$ $\mathbf{R}_3(x)a$ $\mathbf{R}_4(x)a$ $\mathbf{R}_4(x)b$

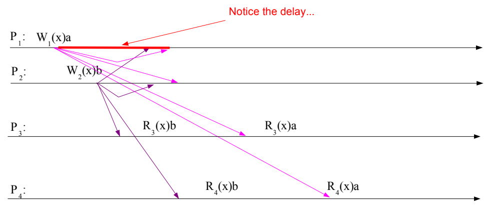
can't read the value b from x

$\mathbf{W}_1(x)a$ $\mathbf{R}_3(x)a$ $\mathbf{R}_4(x)a$ $\mathbf{W}_2(x)b$ $\mathbf{R}_3\mathbf{(x)b}$ $\mathbf{R}_4(x)b$

violate P_3 local sequential order

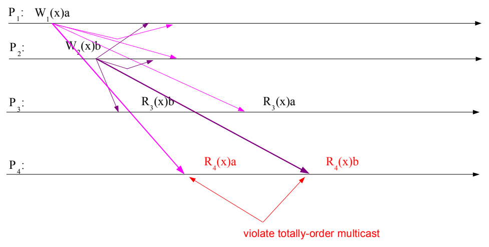
Sequential Consistency

- Possible Design
  - Use totally-ordered multicast on writes
    - All processes see the same order of writes
    - It produces only sequentially consistent executions



Sequential Consistency

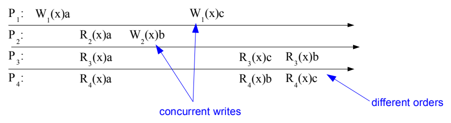
- Non-Sequential Executions
  - Impossible, would violate the totally-ordered multicast property


P_1: $W_1(x)a$
P_2: $W_2(x)b$
P_3: $R_3(x)b$ $R_3(x)a$
P_4: $R_4(x)a$ $R_4(x)b$
violate totally-order multicast

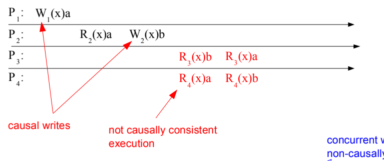
Causal Consistency

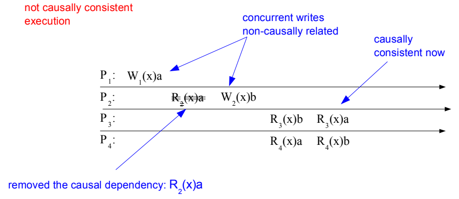
- Weaker Consistency
  - Harder to use, potentially more parallelism
  - Causality
    - If an event E_2 is caused or may be influenced by an event E_1
    - Causality requires that everyone sees the event E_1 before the event E_2

*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.*


P_1: $W_1(x)a$ $W_1(x)c$
P_2: $R_2(x)a$ $W_2(x)b$
P_3: $R_3(x)a$ $R_3(x)c$ $R_3(x)b$
P_4: $R_4(x)a$ $R_4(x)b$ $R_4(x)c$
concurrent writes
different orders

Causal Consistency

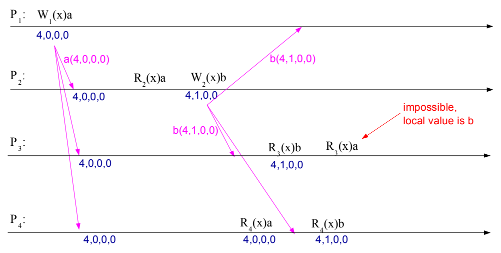

P_1: $W_1(x)a$
P_2: $R_2(x)a$ $W_2(x)b$
P_3: $R_3(x)b$ $R_3(x)a$
P_4: $R_4(x)a$ $R_4(x)b$
causal writes
not causally consistent execution


concurrent writes non-causally related
causally consistent now
P_1: $W_1(x)a$
P_2: ~~$R_2(x)a$~~ $W_2(x)b$
P_3: $R_3(x)b$ $R_3(x)a$
P_4: $R_4(x)a$ $R_4(x)b$
removed the causal dependency: $R_2(x)a$

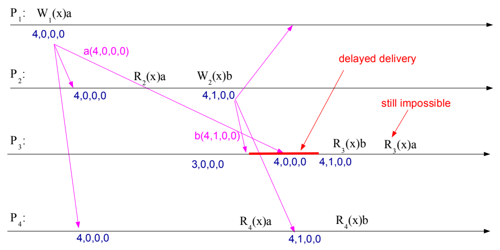
Causal Consistency

- Design
  - Requires that each process keeps tracks of which write operations it has seen
    - One may use vector clocks for this
  - Replica coherence
    - One **vector clock per data item**
      - Clock ticks on writes (as in causally-ordered multicast)
    - On local writes, multicast update messages
      - Causally-ordered multicast of the value
      - Timestamped with the vector clock

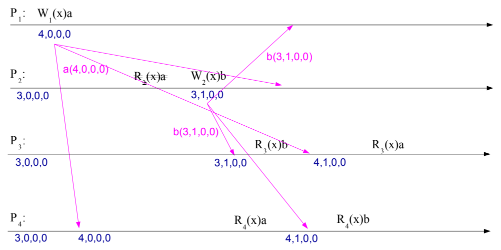
Causal Consistency



Causal Consistency

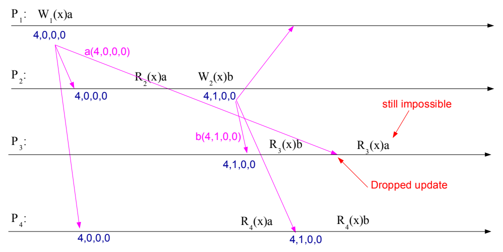


Causal Consistency



Causal Consistency

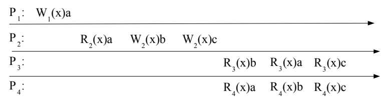
Advanced protocol... when does it work?



FIFO Consistency

- Even Weaker Consistency
  - Even harder to use, but potentially more parallelism

Writes from a single process must be seen by all processes in the same order.
Writes from different processes may be seen in different orders.


P_1: $W_1(x)a$
P_2: $R_2(x)a$ $W_2(x)b$ $W_2(x)c$
P_3: $R_3(x)b$ $R_3(x)a$ $R_3(x)c$
P_4: $R_4(x)a$ $R_4(x)b$ $R_4(x)c$

Implementation: a simple counter per item and per process is enough

Eventual Consistency

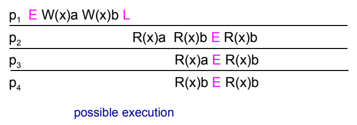
- Eventual Consistency
  - Weaker consistency, but rather easy to use
    - Corresponds to a class of systems with simpler requirements
    - Before, we tried system-wide consistency
      - Assumed concurrent updates from different processes
      - Assume that processes use mutual exclusion or transactions
    - Looking at a special class of data stores
      - Most processes are simply reading data
      - When concurrent updates happen, they can be easily resolved
  - DNS example:
    - Everybody reads, only the domain owner updates DNS records
      - It is ok to read out of date records for a while
      - Use lazy background update messages
    - *Eventually*, copies will get consistent
  - Web example:
    - Same reality about the Web and web page updates
    - Even including in-network caching à la Akamai

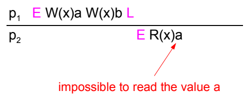
Synchronization and Consistency

- Rationales
  - Consistency on simple read-write operations are costly
  - If synchronization is used, consistency protocols may be delayed at synchronization points
- Basic Idea
  - Associate a monitor with one or more data items
    - Called *protected data items*
  - Coordinate consistency and synchronization protocols
    - Monitor operations must respect sequential consistency
      - Enter and leave critical sections are seen in the same order by all processes
    - Data consistency
      - All writes on protected items must be visible locally before one enters the critical section
      - Accesses (reads or writes) on protected items outside the critical section are undefined

Synchronization & Consistency

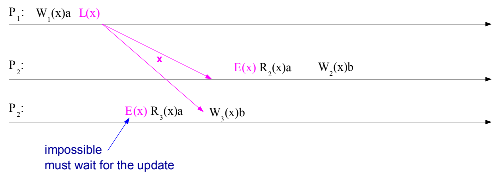
Examples





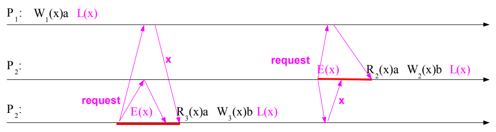
Release Consistency

- Eager Approach
  - Only enters the critical section once all local copies are up to date
    - Acquiring the critical section and receiving pending updates must be coordinated
  - When leaving the critical section
    - *Eagerly* send local updates towards other replicas



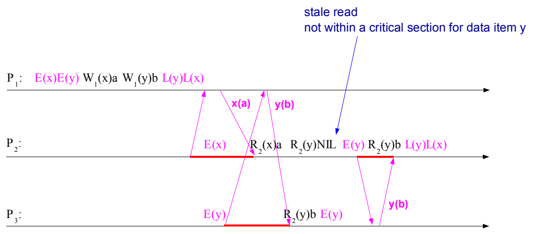
Release Consistency

- Lazy Approach
  - Upon entering the critical section
    - Pull missing updates on protected items
    - Optimization: pigging back updates on granting access
  - When leaving the critical section
    - Nothing needs to be done



Discussing Consistency

- Undefined Semantics
  - Accessing protected data items outside critical sections


stale read
not within a critical section for data item y
P1: E(x)E(y) W1(x)a W1(y)b L(y)L(x)
x(a)
y(b)
P2: E(x) R2(x)a R2(y)NIL E(y) R2(y)b L(y)L(x)
y(b)
P3: E(y) R3(y)b E(y)