CIS 505: Software Systems Lecture Note on Consistency and Replication (3)

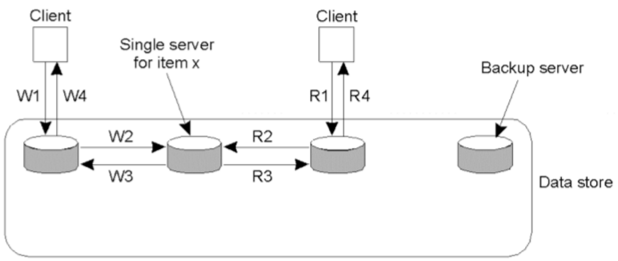
Instructor: Insup Lee
Department of Computer and Information Science
University of Pennsylvania

CIS 505, Spring 2007

Coherence Protocols

- Replicated data need to be ensured to be coherent (i.e., nodes do not access stale data)
- Primary-based protocols
  - Remote-write protocols: all R/W are done by a single server (e.g., partitioned data among servers)
    - Primary-backup protocols: reads can be using a local copy.
    - Easy to support sequential consistency
  - Local-write protocols
    - A single copy of data is migrated to a local server: problem is how to keep track of data
    - Primary can be updated, whereas backups are read only
- Replicated write protocol
- Cache coherence protocol

Remote-Write Protocols (1)

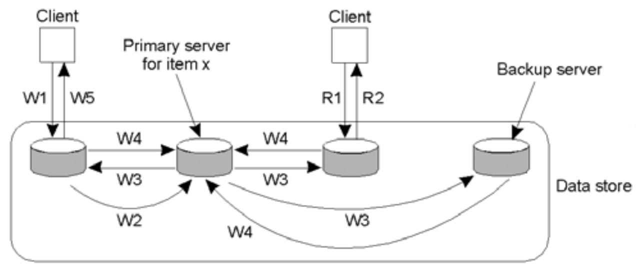


W1. Write request
W2. Forward request to server for x
W3. Acknowledge write completed
W4. Acknowledge write completed

R1. Read request
R2. Forward request to server for x
R3. Return response
R4. Return response

- Primary-based remote-write protocol with a fixed server to which all read and write operations are forwarded.

Remote-Write Protocols (2)



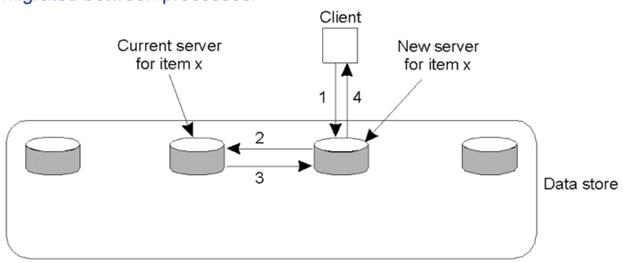
W1. Write request
W2. Forward request to primary
W3. Tell backups to update
W4. Acknowledge update
W5. Acknowledge write completed

R1. Read request
R2. Response to read

- The principle of primary-backup protocol.

Local-Write Protocols (1)

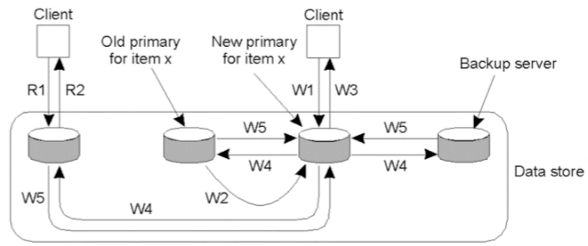
- Primary-based local-write protocol in which a single copy is migrated between processes.



1. Read or write request
2. Forward request to current server for x
3. Move item x to client's server
4. Return result of operation on client's server

Local-Write Protocols (2)

- Primary-backup protocol in which the primary migrates to the process wanting to perform an update.



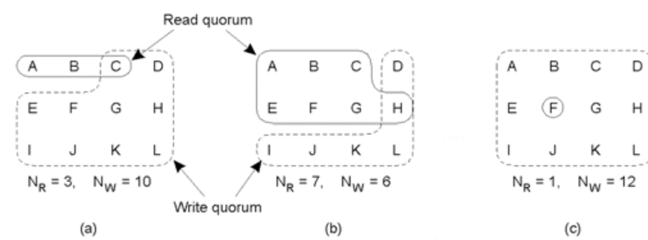
W1. Write request
W2. Move item x to new primary
W3. Acknowledge write completed
W4. Tell backups to update
W5. Acknowledge update

R1. Read request
R2. Response to read

Replicated-Write Protocols

- Writes can be done at multiple replicas
- Active replication
  - o Writes are flushed to all replicas
  - o Need a total ordering
    - Centralized sequencer
    - Totally-ordered multicasting
  - o Potential problem with replicated invocations
- Quorum-based protocols
  - o For N replicas, use voting to get a read quorum (≥ N_r) and a write quorum (≥ N_w), where
    - $N_r + N_w > N$
    - $N_w > N/2$

Quorum-Based Protocols

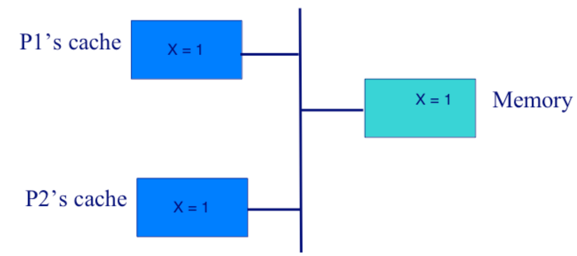


- Three examples of the voting algorithm:
  a) A correct choice of read and write set
  b) A choice that may lead to write-write conflicts
  c) A correct choice, known as ROWA (read one, write all)

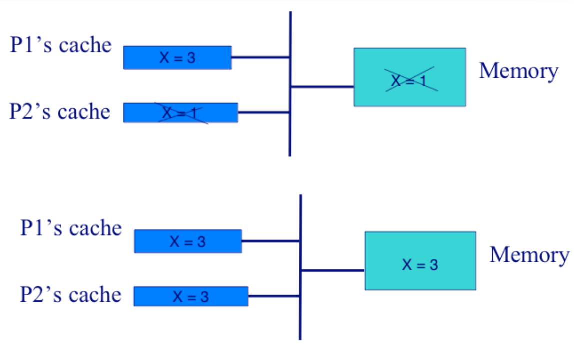
Cache Coherence

- Caches are a special case of replication as they are controlled by clients instead of servers.
- Usually in the context of shared-memory multiprocessor systems
- Many processors can have locally cached copies of the same object
  - Level of granularity can be an object or a block of 64 bytes
- We want to maximize concurrency
  - If many processors just want to read, then each one can have a local copy, and reads won't generate any bus traffic
- We want to ensure coherence
  - If a processor writes a value, then all subsequent reads by other processors should return the latest value
- Coherence refers to a logically consistent global ordering of reads and writes of multiple processors
- Modern multiprocessors support intricate schemes
  - Coherence detection strategy: when inconsistencies are detected
  - Coherence enforcement strategy: how cashes are kept consistent
    - Write-invalidate protocol
      - Invalidate all copies and then write
      - Need to send invalidate-msg to all nodes, even if they no longer use
      - Better if several updates between reads
    - Write-update
      - Update all copies
      - More network traffic overhead

Example



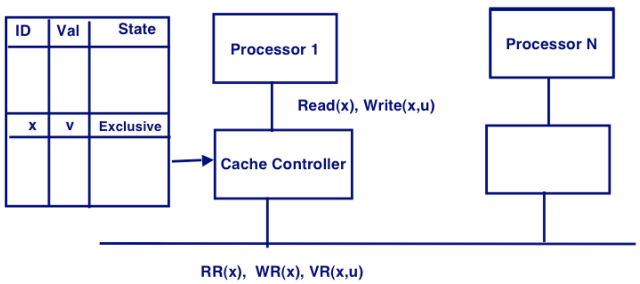
Invalidate vs. update protocols



Snoopy Protocol

- Each processor, for every cached object, keeps a state that can be **Invalid**, **Exclusive** or **Read-only**
- Goal: If one has **Exclusive** copy then all others must be **Invalid**
- Each processor issues three types of messages on bus
  - Read-request (**RR**), Write-request (**WR**), and Value-response (**VR**)
  - Each message identifies object, and VR has a tagged value
- Assumption:
  - If there is contention for bus, then only one succeeds
  - No split transactions (**RR** will have a response by **VR**)
- Protocol is called Snoopy, because everyone is listening to the bus all the time, and updates state in response to messages **RR** and **WR**
- Each cache controller responds to 4 types of events
  - Read or write operation issued by its processor
  - Messages (**RR**, **WR**, or **VR**) observed on the bus
- Caution: This is a simplified version

Snoopy Cache Coherence



Snoopy Protocol

- If state is **Read-only**
  - Read operation: return local value
  - Write operation: Broadcast **WR** message on bus, update state to **Exclusive**, and update local value
  - **WR** message on bus: update state to **Invalid**
  - **RR** message on bus: broadcast **VR(v)** on bus
- If state is **Exclusive**
  - Read operation: return local value
  - Write operation: update local value
  - **RR** message on bus: Broadcast **VR(v),** and change state to **Read-only**
  - **WR** message on bus: update state to Invalid
- If state is **Invalid**
  - Read operation: Broadcast **RR**, Receive **VR(v),** update state to **Read-only**, and local value to v
  - Write operation: As in first case
  - **VR(v)** message on bus: Update state to **Read-only**, and local copy to v
  - **WR** message on the bus: do nothing

Sample Scenario for Snoopy

- Assume 3 processors P1, P2, P3. One object *x : int*
- Initially, P1's entry for x is invalid, P2's entry is Exclusive with value 3, and P3's entry is invalid
- A process running on P3 issues Read(x)
- P3 sends the message RR(x) on the bus
- P2 updates its entry to Read-only, and sends the message VR(x,3) on the bus
- P3 updates its entry to Read-only, records the value 3 in the cache, and returns the value 3 to Read(x)
- P1 also updates the x-entry to (Read-Only, 3)
- Now, if Read(x) is issued on any of the processors, no messages will be exchanged, and the corresponding processor will just return value 3 by a local look-up

- P1: x=(inv,-) … x=(ro,3)
- P2: x=(exc,3) … X=(ro,3); VR(x,3);
- P3: x=(inv,-) … Read(x); RR(x); … x=(ro,3),return(x,3)

Snoopy Scenario (Continued)

- Suppose a process running on P1 issues Write(x,0)
- At the same time, a process running on P2 issues Write(x,2)
- P1 will try to send WR on the bus, as well as P2 will try to send WR on the bus
- Only one of them succeeds, say, P1 succeeds
- P1 will update cache-entry to (Exclusive,0)
- P3 will update cache-entry to Invalid
- P2 will update cache-entry to Invalid
- Now, Read / Write operations by processes on P1 will use local copy, and won't generate any messages

- P1: Write(x,0); WR(x); x=(ex,0)
- P2: Write(x,2); WR(x); x=(inv,-)
- P3: … x=(inv,-)