CompSci131

Parallel and Distributed Systems

Prof. A. Veidenbaum

Today's topics

- Causal Consistency
- Grouping operations

- Reading assignment:
  - Today: the rest of 7.2
  - Next time: Cache coherence (lecture notes only)
  - After that: 7.3

Last Lecture Covered

- **Sequential consistency**

Causal Consistency

- Relaxes some of the sequential consistency rules
  - To improve performance
- Already saw causality in logical clock events
  - Data in a message used in computation, result sent out
- Can something like this be used to relax rules?
  - P1: W(a)x
  - P2: W(b)y
- The two writes are completely independent
  - Are not causally related, may be seen in different order…

Causal Consistency

- On the other hand, if
  - P1 performs W(x),
  - P2 performs Rd(x), Wr(y)
  - The two writes are causally related, may be dependent

- A necessary condition for causal consistency:
  - 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 by different processes.

Causal Consistency Example

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

- **Is this valid for a sequentially consistent store?**
- **Is this a valid causally consistent store?**

More Causal Consistency

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

(a)

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

(b)

a) **Is this a casually-consistent store?**

b) **What about this one?**

More Causal Consistency

- **Previous examples looked at one variable – x**
- **True causality is better illustrated by the next example**
  - **Assume prior values of x and y are NIL**

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

- **What values can R_3(x) and R_4(y) return?**

- **How can this be implemented?**
  - **Maintain a dependency graph between accesses**

Grouping Operations

- **Even less strict models are possible and useful**
- **So far dealt with a conit R/W order**
  - **May not match the granularity of applications**
  - **Not all writes are important**
- **An example: a Critical Section (CS)**
  - **Only one process in the CS**
  - **operates on N variables in the CS**
    - » **only these N variables are important, not the rest**
  - **But the system may not know a process is in a CS!**
- **Solution: *a system <u>aware</u> of synchronization***

- **Define "Enter_CS" and "Leave_CS" operations**
  - Enter_CS guarantees the local store is up-to-date
    - » Stalls until this is true
  - Initiate update upon/prior to Leave_CS
- **All reads and writes are grouped into an indivisible unit accessed and updated by one process**
  - *The rest are not guaranteed to be consistent*
- **Need to define the semantics of Enter and Leave**
  - Done using shared synchronization variables, aka locks
    - » Operations on locks – Acquire and Release
  - Shared data items are associated with each lock
-
    - » Each shared data item is associated with at most one lock

- **Define a Synchronization Variable (SV)**
  - **Each has a current "owner"**
    - » **the process that last acquired it**
  - **This process may enter and exit the CS many times**
- **A process not currently owning the SV has to**
  - **Send message to the current owner**
    - » **Requesting SV ownership *AND* <u>all associated data</u>**
- **Several processes may simultaneously own an SV, *but in a non-exclusive mode***
  - **They can only Read the SV, *not write it.***
- ***Note that we are basically requiring sequential consistency for locks***

- **The following criteria need to be met:**
  - A lock acquire is not allowed to perform until all the updates to its guarded shared data have been performed
    - With respect to the acquiring process
  - An exclusive mode access to an SV is allowed to perform only when no other process holds the SV
    - Exclusively or non-exclusively
  - No read or write operation on data items are allowed to be performed until lock acquire has been performed.
  - A non-exclusive access to the SV may be performed only after any previous exclusive access has completed
    - E.g. the SV owner updated the data and released the lock

Entry Consistency

- **A way to implement a form of grouping**
- **Associate a lock with each data item**
  - **L(x) is a lock op for data item x**
  - **U(x) is an unlock op for data item x**

P1:	L(x)	W(x)a	L(y)	W(y)b	U(x)	U(y)				
P2:							L(x)	R(x)a		R(y) NIL
P3:									L(y) R(y)b	

- **Each process has a copy of a variable, BUT**
  - **This copy is not instantly or atomically updated**
    - » **Updated on locking**
- ***Not easy to program due to multiple locks!***