

Consistency Models

Based on Tanenbaum/van Steen's
"Distributed Systems", Ch. 6, section 6.2

Consistency Models

- Both for DSM, but also for regular multiprocessors with real shared memory
 - i.e., either hardware or software could be enforcing these protocols
- (Tanenbaum 6.2)

Consistency Models

- A *Consistency Model* is a contract between the software and the memory
 - it states that the memory will work correctly but only if the software obeys certain rules
- The issue is how we can state rules that are not too restrictive but allow fast execution in most common cases
- These models represent a more general view of sharing data than what we have seen so far!

Conventions we will use:

- $W(x)a$ means “a *write* to x with value a ”
- $R(y)b$ means “a *read* from y that returned value b ”
- “processor” used generically

Strict Consistency

- Strict consistency is the strictest model
 - a read returns the most recently written value (changes are instantaneous)
 - not well-defined unless the execution of commands is serialized centrally
 - otherwise the effects of a slow write may have not propagated to the site of the read
 - this is what uniprocessors support:
 - `a = 1; a = 2; print(a);` always produces “2”
 - to exercise our notation:
 - P1 : $W(x) 1$
 - P2 : $R(x) 0 \quad R(x) 1$
 - is this strictly consistent?

Sequential Consistency

- Sequential consistency (*serializability*): the results are the same as if operations from different processors are interleaved, but operations of a single processor appear in the order specified by the program
- Example of sequentially consistent execution:
P1: W(x)1
P2: R(x)0 R(x)1
- Sequential consistency is inefficient: we want to weaken the model further

Causal Consistency – by Tech people

- Causal consistency: 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

- Examples (which one is causally consistent, if any?)

| | | |
|-----|-------------|-------------|
| P1: | W(x)1 | W(x)3 |
| P2: | R(x)1 W(x)2 | |
| P3: | R(x)1 | R(x)3 R(x)2 |
| P4: | R(x)1 | R(x)2 R(x)3 |

| | | |
|-----|-------------|-------------|
| P1: | W(x)1 | |
| P2: | R(x)1 W(x)2 | |
| P3: | | R(x)2 R(x)1 |
| P4: | | R(x)1 R(x)2 |

- Implementation needs to keep dependencies

Pipelined RAM (PRAM) or FIFO Consistency

- PRAM consistency is even more relaxed than causal consistency: writes from the same processor are received in order, but writes from distinct processors may be received in different orders by different processors

P1: W(x)1

P2: R(x)1 W(x)2

P3: R(x)2 R(x)1

P4: R(x)1 R(x)2

- Slight refinement:
 - **Processor consistency:** PRAM consistency plus writes to the same memory location are viewed everywhere in the same order

Weak Consistency

- Weak consistency uses synchronization variables to propagate writes to and from a machine at appropriate points:
 - accesses to synchronization variables are sequentially consistent
 - no access to a synchronization variable is allowed until all previous writes have completed in all processors
 - no data access is allowed until all previous accesses to synchronization variables (by the same processor) have been performed
- That is:
 - accessing a synchronization variable “flushes the pipeline”
 - at a synchronization point, all processors have consistent versions of data

Weak Consistency Examples

- Which one is valid under weak consistency?
 - convention: **S** means access to synchronization variable

P1: W(x)1 W(x)2 S

P2: R(x)1 R(x)2 S

P3: R(x)2 R(x)1 S

P1: W(x)1 W(x)2 S

P2: S R(x)1

- Tanenbaum says the second is not weakly consistent. Do you agree? (What is the order of synchronizations? How do the rules prevent this execution?)
- Weak consistency means that the programmer has to manage synchronization explicitly

Release Consistency

- 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)

- example (valid or not?):

P1: L W(x)1 W(x)2 U

P2: L R(x)2 U

P3: R(x)1

- Variant: entry consistency: like lazy release consistency but all data variables are explicitly associated with synchronization variables

Summary of Consistency Models

Consistency models not using synchronization operations.

| Consistency | Description |
|-----------------|--|
| Strict | Absolute time ordering of all shared accesses matters. |
| Linearizability | All processes must see all shared accesses in the same order. Accesses are furthermore ordered according to a (nonunique) global timestamp |
| Sequential | All processes see all shared accesses in the same order. Accesses are not ordered in time |
| Causal | 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 |

Models with synchronization operations.

| 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. |

a) Consider the following sequence of operations:

P1: $W(x)_1$ $W(x)_3$

P2: $W(x)_2$

P3: $R(x)_3$ $R(x)_2$

P4: $R(x)_2$ $R(x)_3$

Is this execution causally consistent? Add or modify an event to change the answer.

d) Consider the following sequence of operations:

P1: L W(x)1 W(x)3 U

P2: L R(x)3 U R(x)1

Is this execution release consistent? Add or modify an event to change the answer.