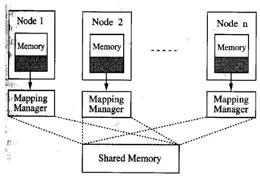
Distributed shared memory

- motivation and the main idea
- consistency models
  - strict and sequential
  - causal
  - PRAM and processor
  - weak and release
- implementation of sequential consistency
- implementation issues
  - granularity
  - thrashing
  - page replacement

DSM idea



- all computers share a single paged, virtual address space
- pages can be physically located on any computer
- when process accesses data in shared address space a *mapping manager* maps the request to the physical page
- mapping manager – kernel or runtime library
- if page is remote – block the process and fetch it

Advantages of DSM

- Simpler abstraction - programmer does not have to worry about data movement, may be easier to implement than RPC since the address space is the same
- easier portability - sequential programs can in principle be run directly on DSM systems
- possibly better performance
  - locality of data - data moved in large blocks which helps programs with good locality of reference
  - on-demand data movement
  - larger memory space - no need to do paging on disk
- flexible communication - no need for sender and receiver to exist, can join and leave DSM system without affecting the others
- process migration simplified - one process can easily be moved to a different machine since they all share the address space

Maintaining memory coherency

- DSM systems allow concurrent access to shared data
- concurrency may lead to unexpected results - what if the read does not return the value stored by the most recent write (write did not propagate)?
- Memory is *coherent* if the value returned by the read operation is always the value the programmer expected
- To maintain coherency of shared data a mechanism that controls (and synchronizes) memory accesses is used.
- This mechanism only allows a restricted set of memory access orderings
- *memory consistency model* - the set of allowable memory access orderings

Strict and sequential consistency

- strict consistency (strongest model)
  - value returned by a read operation is always the same as the value written by the most recent write operation
  - hard to implement
- sequential consistency (Lamport 1979)
  - the result of any execution of the operations of all processors is the same as if there were executed in <u>some</u> sequential order and one process' operations are in the order of the program
    - Interleaving of operations doesn't matter, if all processes see the same ordering
  - read operation may not return result of most recent write operation!
    - running a program twice may give different results
  - little concurrency

Causal consistency

- proposed (Hutto and Ahmad 1990)
- there is no single (even logical) ordering of operations – two processes may see the same operations ordered differently
- the operations are sequenced in the same order if they are potentially causally related
- read/write (or two write) operations on the same item are causally related
- all operations of the same process are causally related
- causality is transitive - if a process carries out an operation B that causally depends on the preceding op A - all consequent ops by this process are causally related to A (even if they are on different items)

PRAM and processor consistency

- PRAM (Lipton & Sandberg 1988)
  - All processes see only memory writes done by a single process in the same (correct) order
  - PRAM = pipelined RAM
    - Writes done by a single process can be pipelined; it doesn't have to wait for one to finish before starting another
    - writes by different processes may be seen in different order on a third process
  - Easy to implement — order writes on each processor independent of all others
- Processor consistency (Goodman 1989)
  - PRAM +
  - coherency on the same data item - all processes agree on the order of write operations to the same data item

Weak and release consistency

- Weak consistency (Dubois 1988)
  - Consistency need only apply to a group of memory accesses rather than individual memory accesses
  - Use synchronization variables to make all memory changes visible to all other processes (e.g., exiting critical section)
    - all access to synchronization variables must be sequentially consistent
    - write operations are completed before access to synchvar
    - access to non-synchvar is allowed only after sycnhvar access is completed
- Release consistency (Gharachorloo 1990)
  - two synchronization vars
    - acquire - all changes to synchronized vars are propagated to the process
    - release - all changes to synchronized vars are propagated to other processes
    - programmer has to write accesses to these variables

Comparison of consistency models

- Models differ by difficulty of implementation implement, ease of use, and performance
- Strict consistency — most restrictive, but hard to implement
- Sequential consistency — widely used, intuitive semantics, not much extra burden on programmer
  - But does not allow much concurrency
- Causal & PRAM consistency — allow more concurrency, but have non-intuitive semantics, and put more of a burden on the programmer to avoid doing things that require more consistency
- Weak and Release consistency — intuitive semantics, but put extra burden on the programmer

Implementation issues

- how to keep track of the location of remote data
- how to overcome the communication delays and high overhead associated with execution of communication protocols
- how to make shared data concurrently accessible at several nodes to improve system performance

Implementing sequential consistency on page-based DSM

- Can a page move? …be replicated?
- Nonreplicated, nonmigrating pages
  - All requests for the page have to be sent to the owner of the page
  - Easy to enforce sequential consistency — owner orders all access request
  - No concurrency
- Nonreplicated, migrating pages
  - All requests for the page have to be sent to the owner of the page
  - Each time a remote page is accessed, it migrates to the processor that accessed it
  - Easy to enforce sequential consistency — only processes on that processor can access the page
  - No concurrency

Implementing sequential consistency on page-based DSM (cont.)

- Replicated, migrating pages
  - All requests for the page have to be sent to the owner of the page
  - Each time a remote page is accessed, it's copied to the processor that accessed it
  - Multiple read operations can be done concurrently
  - Hard to enforce sequential consistency — must invalidate (most common approach) or update other copies of the page during a write operation
- Replicated, nonmigrating pages
  - Replicated at fixed locations
  - All requests to the page have to be sent to one of the owners of the page
  - Hard to enforce sequential consistency — must update other copies of the page during a write operation

Granularity

- Granularity - size of shared memory unit
- Page-based DSM
  - Single page — simple to implement
  - Multiple pages — take advantage of locality of reference, amortize network overhead over multiple pages
    - Disadvantage — false sharing
- Shared-variable DSM
  - Share only those variables that are need by multiple processes
  - Updating is easier, can avoid false sharing, but puts more burden on the programmer
- Object-based DSM
  - Retrieve not only data, but entire object — data, methods, etc.
  - Have to heavily modify old programs

Thrashing

- Occurs when system spends a large amount of time transferring shared data blocks from one node to another (compared to time spent on useful computation)
  - interleaved data access by two nodes causes data block to move back and forth
  - read-only blocks invalidated as soon as they are replicated
- handling thrashing
  - application specifies when to prevent other nodes from moving block - has to modify application
  - “nailing” block after transfer for a minimum amount of time t - hard to select t, wrong selection makes inefficient use of DSM
    - adaptive nailing?
  - tailoring coherence semantics (Minin) to use – object based sharing

Page replacement

- What to do when local memory is full?
  - swap on disk?
  - swap over network?
  - what if page is replicated?
  - what if it’s read-only?
  - what if it’s read/write but clean(dirty)?
  - are shared pages given priority over private (non-shared)?