

Lecture 16/17: Distributed Shared Memory

CSC 469H1F
Fall 2006
Angela Demke Brown



Outline

- Review distributed system basics
- What is distributed shared memory?
- Design issues and tradeoffs

CSC469



Distributed System Features

- **Multiple** computers
 - May be heterogenous, or homogeneous
 - May be controlled by a single organization or by distinct organizations or individuals
 - **No physical shared memory**, no shared clock
- Connected by a **communication network**
 - Typically a general-purpose network, not dedicated to supporting the distributed system
 - Messages are sent over network for communication
- **Co-operating** to share resources and services
 - Application processing occurs on more than one machine

CSC469



Distributed IPC

- Option 1: Use message passing primitives
 - E.g. Unix sockets
 - ✓ Good match for underlying structure
 - ✗ programmer has to deal with sending data
- Option 2: Use remote procedure call (RPC)
 - ✓ Familiar programming model
 - ✓ RPC system handles communication details
 - ✗ passing complex data types is hard
 - ✗ model is synchronous, not a good fit for parallel programming

CSC469



(Local) Shared Memory

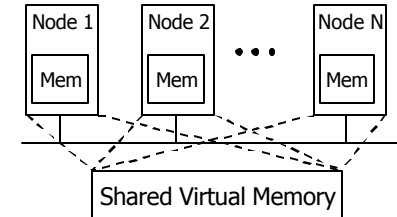
- Uniprocessor or SMP systems
- Processes can share part of their address space
 - Threads in a process share entire address space
- IPC provided through access to shared data
 - ✓ Easy to express concurrency, share complex data structures
 - ✗ Synchronization needed to prevent data races
- How is this implemented on single computer?
- Can we achieve same effect on dist. system?

CSC469



Distributed Shared Memory (DSM)

- Goal: allow processes on networked computers to share physical memory through a single shared virtual address space



CSC469



DSM Basics

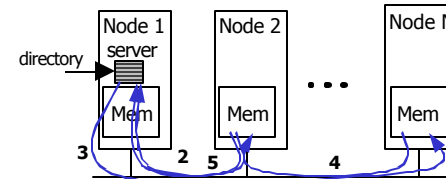
- Physical memory on each node holds pages of shared virtual address space
 - Local pages are present in current node's memory
 - Remote pages are in some other node's memory
- Exploit MMU hardware to locate pages
 - Page table entry for a page is valid if the page is local
 - Access to non-local page causes a page fault
 - DSM protocol handles page fault, retrieves remote data
 - Operations are transparent to programmer

CSC469



Locating Remote Data

- Simplest Design: central server maintains directory recording which machine currently holds each page



1. Node 2 pg faults
2. Consult central server to locate
3. Page requested from current owner, Node N
4. Owner invalidates, sends to new location, Node 2
5. Node 2 informs directory of new ownership

CSC469



Problem 1

- Directory at central server becomes bottleneck
 - All page query requests go to this node
- Solution: Distributed directory
 - Each node is responsible for portion of address space
 - Responsible node = (page #) mod (num nodes)

N1		N2		...	N4	
Page	Locn	Page	Locn		Page	Locn
0000	N3	0001	N1		0003	N4
0004	N1	0005	N3		0007	N2
0008	N2	0009	N4		000B	N3
000C	N2	000D	N1		000F	N3
...

CSC469



Problem 2

- Each virtual page exists on only one machine at time
 - No caching
- Actively shared pages may lead to thrashing
- Solution: allow replication (caching)
 - Read operations become cheaper
 - Simultaneous reads can be executed locally on multiple nodes
 - Write operations become more expensive
 - Cached copies need to be invalidated or updated

CSC469



Simple Replication

- Multiple Readers, Single Writer (MRSW)
 - One node can be granted a read-write copy
 - **OR** multiple nodes can be granted read-only copies
- On read operation:
 - Acquire read-only copy of the page
 - Set access rights to read-only on any writeable copy on other nodes
- On write operation:
 - Revoke write permission from other writable copy (if any)
 - Get read-write copy of page
 - Invalidate all copies of page at other nodes

CSC469



Full Replication

- Multiple readers, multiple writers
 - More than one node can have writable copy of page
 - Access to shared data must be controlled to maintain consistency
 - More on this in a minute...

CSC469



Dealing with replication

- Must keep track of copies of the page
 - Extend directory with copyset
 - The set of all nodes that requested copies
- On request for page copy
 - Add requestor to copyset
 - Send page contents
- On request to invalidate page
 - Send invalidation requests to all nodes in copyset and wait for acknowledgements

CSC469



Consistency Model

- Defines when modifications to data may be seen at a given processor
- Defines how memory will appear to a programmer
 - Restricts what values can be returned by a read of a memory location
- Must be well-understood
 - Determines how programmer reasons about correctness of program
 - Determines what optimizations are allowed

CSC469



Recall Sequential Consistency

- All memory operations must execute one at a time
- All operations of a single processor appear to execute in program order
- Interleaving among processors is ok

CSC469



Achieving Sequential Consistency

- Node must ensure that previous memory operation is complete before proceeding with the next one
 - Must get acknowledgement that write has completed
 - With caching, must send invalidate or update messages to all copies
 - **ALL** these messages must be acknowledged
- To improve performance we relax the rules

CSC469



Relaxed (weak) consistency

- Allow reads/writes to different memory locations to be reordered
- Consider operation in critical section:
 - Should be used for all shared data operations
 - One process actively reading/writing
 - Nobody else will access until process leaves c.s.
 - → No need to propagate writes sequentially, or at all, until process leaves critical section!

CSC469



Synchronization Variables

- Operation for synchronizing memory
 - Analog of fences in shared memory multiprocessors
- All local writes get propagated
- All remote writes are brought in to the local processor
- Block until memory synchronized
- Access to synchronization variables are sequentially consistent

CSC469



Problems with Weak Consistency

- Inefficiency
 - Synchronization happens at begin and end of a critical section
 - Is process finished memory access? Or is it about to start?
- System must make sure
 - All locally-initiated writes have completed
 - All remote writes have been acquired

CSC469



Can we do better?

- Separate synchronization into two stages:
 1. **acquire access**
 - Obtain valid copies of all pages
 2. **release access**
 - Send invalidations for shared pages that were modified locally to nodes that have copies
- Eager Release Consistency

CSC469



Can do better still

- Release requires sending invalidations to all nodes with copy
 - And waiting for all to acknowledge
- Delay this process
 - On release, send invalidation to directory
 - On acquire, check with directory to see if new copy is needed
- Reduces message traffic on release

- Lazy Release Consistency

CSC469



How do you propagate changes?

- Send entire page
 - Easy, but may be a lot of data
- Send only what changed
 - Local system must save original and compute differences

CSC469

