Cluster Computing

Distributed Shared Memory

- Introduction
- Data Consistency
- Network Performance Issues
- Other Design Issues
- Conclusions

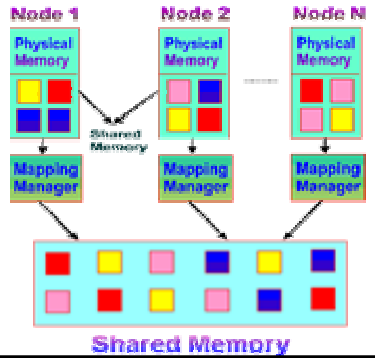
2

Introduction

- Loosely-coupled distributed systems
  - message passing
  - RPC
- Tightly-coupled architectures (multi-processor)
  - shared memory
  - simple programming model
- Shared memory gives transparent process-to-process communication, and easy programming
- Distributed shared memory
  - extended for use in more loosely-coupled architecture
  - processes shares data transparently across node boundaries
  - allow parallel programs to execute without modifications
  - programs are shorter and easier than equivalent message-passing
  - applications are slower than hand-coded message passing
  - consistency maintenance problem

3

Distributed Shared Memory



4

What DSM can Contribute?

- Automatic data distribution
  - Data gets pulled where its need on demand
  - Location transparency
- Transparent coherence management
  - Shared data automatically kept consistent
- Can build fault tolerance and security

5

Introduction

- DSM vs. Message-Passing
  - A well built message-passing implementation outperform shared-memory
  - Relative costs of communication and computation are important factors
    - In some cases, DSM can execute faster than message-passing
- Main Issues of DSM
  - Data Consistency
    - Data Location
    - Write Synchronization
    - Double Faulting
    - Relaxing Consistency
    - Application/Type-specify Consistency
  - Network Performance Issues
  - Other Design Issues
    - Synchronization
    - Granularity
    - Address-Space Structure
    - Replacement Police
    - Heterogeneity Support
    - Fault Tolerance
    - Memory Allocation
    - Data Persistence

6

Data Consistency

- Network latency
  - caching and consistency algorithms
- Three consistency model
  - strict consistency - "sequential consistency" (Lemport)
  - loosely consistency - synchronization points (compiler or user)
  - no consistency - user-level synchronization and consistency
- Many older DSM systems - strict consistency
  - single writer/multiple reader model
    - need one data locating node
    - data is replicated on two or more nodes, none has write access
    - two problem
      - must locate a copy of the data when it is non-resident
      - must synchronize accesses to the data, when writing at the same time
- But the performance benefits of relaxing the consistency model

7

Strict Consistency

- All writes are *instantaneously visible* to all processes and *absolute global time order* is maintained throughout the DS.
- Not at all easy!!!

8

Data Consistency - Data Location

- Assign an "owner" node to each item of data
- The owner is allowed to write to the data
- Ownership algorithm
  - Fixed ownership
    - each piece of data is assigned a fixed owner
    - from the address of the data (ex. by hashing)
  - Dynamic ownership
    - Manager: not own the data, but track the current owner
    - Centralized
      - A node is selected as the "manager" for all the shared data
    - Distributed
      - split the ownership information among the nodes
      - Broadcast
        - faulting processors send a broadcast message to find the current owner of each data item
        - O(2K-1)
      - Dynamic
        - search "probable" owner of each data item
        - in worst case, O(p+Klog p)

9

Data Consistency – Write Synchronization

- In maintaining a strictly-consistent data set
- Write Broadcast (write-update)
  - can broadcast writes
  - too expensive to be used
  - if there is not a low level broadcast primitive, it's a problem
- Invalidation (write invalidate)
  - can invalidate all other copies before doing an update
  - broadcasting is expensive (in large DSM system)
  - use unicast or multicast rather than broadcast
    - problem of location
  - the manager site maintains a "copy-set"
  - distributing the copy-set data
    - invalidation are passed down the tree

10

Data Consistency – Double Faulting

- Read & write fault: two network transactions
  - one to obtain a read-only copy of the page
  - a second to obtain write access to the page
    - the page is transmitted twice over the network
- Algorithms to solve double faulting problem
  - Hot Potato
    - all faults are treated as write faults
    - the ownership and data are transferred on very fault
  - Li
    - transfer ownership on a read fault
    - subsequent write fault may be handled locally
  - Shrewd
    - a sequence number per copy of a page
    - use to track where a page has been updated
    - don't eliminate the extra network transactions
- The crucial factor appears to be the read-to-write ratio of the application

11

Sequential Consistency

- A weaker consistency model
- *The result of any execution is the same as if the (read and write) operations by all proceses on the data-store were executed in some sequential order and the operations of each individual process appear in this sequence in the order specified by its program.*

12

Data Consistency – Relaxing Consistency

- Permitting temporary inconsistencies = increasing performance
- Memory is "loosely consistent"
  - If the value returned by a read operation is the value written by an update operation to the same object that "could" have immediately preceded the read operation in some legal schedule of the threads in execution
  - More than one thread may have write access to the same object, provided that the programmer knows that the writes will not conflict
- Delayed updates
  - virtual synchrony - message operations appear to be synchronous
  - semantic knowledge - when it is needed to access or update?
  - compromise the shared memory concept & increase complexity

13

Data Consistency – Relaxing Consistency

- Release Consistency
  - ordinary & synchronization-related accesses (acquire, release)
  - the "acquire" operation signals that shared data is needed
  - a processor's updates after an "acquire" are not guaranteed to be performed at other nodes until a "release" is performed
  - *When a process does an "acquire", the data-store will ensure that all the local copies of the protected data are brought up to date to be consistent with the remote ones if needs be.*
  - *When a "release" is done, protected data that have been changed are propogated out to the local copies of the data-store.*

14

Data Consistency – Relaxing Consistency

- Entry Consistency
  - in a DSM system called Midway
  - weaker than other models
  - it requires explicit annotations to associate synchronization object with data
  - on an "acquire", only the data associated with the synchronization object is guaranteed to be consistent
  - Each shared variable is associated with a synchronization variable.
  - When acquiring the synchronization variable, the most recent values of its associated shared variables are fetched.

15

Data Consistency – Relaxing Consistency

- Causal Consistency
  - This model distinguishes between events that are "causally related" and those that are not.
  - *If event B is caused or influenced by an earlier event A, then causal consistency requires that every other process see event A, then event B.*

  or

  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 on different machines (i.e., by different processes).
  - Operations that are not causally related are said to be *concurrent*.

16

Data Consistency – Application/Type-specific Consistency

- Problem-oriented shared memory
  - implement fetch and store operations specialized to the particular problem or application
  - provide a specialized form of consistency and consistency maintenance that exploits application-specific semantics
  - types of objects
    - synchronization: lock objects, managed using distributed lock system
    - private: not managed by the runtime, but brought back under the control of DSM if referenced remotely
    - write-once: initialized once and then only read, so may be efficiently replicated
    - result: not read until fully updated, use the delayed-updated mechanism to maximum benefit
    - producer consumer: written by one thread, read by a fixed set of threads
      - perform "eager object movement", update are transmitted to the reader before they request them

17

Data Consistency – Application/Type-specific Consistency

- Problem-oriented shared memory
  - types of objects
    - migratory: accessed by a single thread at a time, integrating the movement of the object with the movement of the lock associated with the object
    - write-many: written by many different nodes at the same time, sometimes without being read in between, handled using delayed updates
    - read-mostly: updated rarely using broadcasting
    - general read-write: multiple threads are reading and writing, or no hint of object type is given
  - 30% performance improvement is obtained
- Programmer's control
  - open implementation (meta-protocol)

18

Network Performance Issues

- Communication latency and bandwidth are so bad that affect design decisions of DSM system
  - Much larger than a multi-processor
  - Consistency and cache management (ex, data invalidation)
- Recent advance in network
  - Switching technology
  - Bandwidth are available over 1 Gbps even over wide-area network
  - Processor speeds is increasing
  - Latency is still higher than tightly-coupled architecture
  - Key factor is the computation-to-communication ratio
    - high ratios are suitable to message-passing mechanism
    - analogues to the "memory wall"

19

Other Design Issues - Synchronization

- It orders and controls accesses to shared data before actually accessing the data
- Clouds merges locking with the cache consistency protocol
- IVY - synchronization primitive (eventcounts)
- Mermaid - P and V operations and events
- Munin provides a distributed lock mechanism using "proxy objects" to reduce network load
- Anderson - locking using an Ethernet-style back-off algorithm (instead of spin and wait locks)
- Goodman - propose a set of efficient primitive to DSM case

20

Other Design Issues - Granularity

- When caching objects in local memory
  - Use fixed block size
- The choice of the block size
  - Cost of communication
  - The locality of reference in the application
  - False sharing or Ping-ping effect
- Virtual-memory management unit or multiple thereof
- The lowest common multiple of hardware-supported page size in heterogeneous machines
- Memnet - fixed small granularity (chunk = 32byte)
  - similar to shared memory multiprocessor system
- Mether - a software implementation of Memnet on top of SunOS
  - 8K page size

21

Other Design Issues – Address-Space Structure

- The arrival of 64-bit processors ⇒ single shared address-space systems (single level store)
  - The system appears as a set of threads executing in a single shared distributed address space
  - Data items always appear at the same addresses on all nodes
  - A private and a shared region
  - Security and protection are a major problem
- Separate Shared Address Spaces
  - Divide each process's address space into different fixed regions
  - Cloud - O (shared), P(local/user), K(local/kernel)
  - Objects always appear at the same address, but may not be visible from every address space
- Shared-Region Organization
  - How the shared region itself is organized?
  - Some DSM (IVY, Mether) use a single flat region
  - Most of DSM use paged segmentation
    - The shared region consists of disjoint pieces that usually managed separately

22

Other Design Issues - Replacement Policy and Secondary Storage

- A policy to deal with cache overflow
- LRU does not work well without modification
  - invalidated page
  - read-only page
  - replicated page
  - writable page/pages don't have replicas
- (Markatos, Dramitinos) use of main memory in other nodes to store pages
- (Memnet) reserve part of main memory of each node as storage for chunk
- (IVY) piggybacking memory-state information on other network message
- (Feeley) use of global memory for file buffer caching, VM for applications, and transactions

23

Other Design Issues - Heterogeneity Support

- Attempts to provide support for heterogeneous systems
  - Mermaid, CMU's Mach system, DiSOM
- Page-based Heterogeneity
  - support for different page sizes without considering the data of pages
    - Mach: use scheduler page size unit
    - It's not as simple as it sounds
  - real heterogeneous-data support (byte-order, word-size)
    - Endian problem, word size
    - CMU DSM divide this problem to hardware data-type and software-type
- Language-assisted Heterogeneity
  - DiSOM, object-oriented and operates at the language level
    - the application programmer should provide customized packing and unpacking routines for heterogeneity (marshalling)
  - Gokhale and Minnich, packing at compile time
    - load/store is 2-6 times slower

24

Other Design Issues – Fault Tolerance

- Most of DSM ignore the fault tolerance issue
- DSM system would strongly effect the fault tolerance of a system.
  - N systems are sharing access to a set of data, the failure of any one of them could lead to the failure of all the connected sites
- How do you handle a failed page-fault?
  - Clouds: a transactional system called Invocation-Based Concurrency Control (IBCC) using commit
  - Wu and Fuchs: incremental checkpointing system based on a twin-page disk storage management system
  - Ouyang and Heiser: two-phase commit protocol
  - DiSOM: checkpoint-based failure recovery
  - Casper: shadow paging

25

Other Design Issues – Memory Allocation

- The same piece of shared memory must not be allocated more than once
- Amber allocates large chunks of the shared address space to each site at startup, and has an address-space server that hands out more memory on demand

26

Other Design Issues – Data Persistence

- DSM support some form of secondary storage persistence for data held in shared memory
- It's not easy to obtain a globally-consistent snapshot of the data on secondary storage because of the distributed data
- Overlap between DSM and database system
- Garbage collection: global garbage collection
- Naming and location issues

27

Conclusions

- A useful extra service for a distributed system
- Provide a simple programming model for a wide range of parallel processing problems
- Advantages
  - no explicit communication is necessary
  - may be made of locality of reference
- Disadvantages
  - handling of security and protection issues are unclear
  - fault tolerance may be more difficult to achieve
- Choosing between DSM and message-passing is quite difficult
  - the amount of data exchanged between synchronization points is the main indicator
  - Forin: "... that the amount of data exchanged between synchronization points is the main indicator to consider when deciding between the use of distributed shared memory and message passing in a parallel application"

28