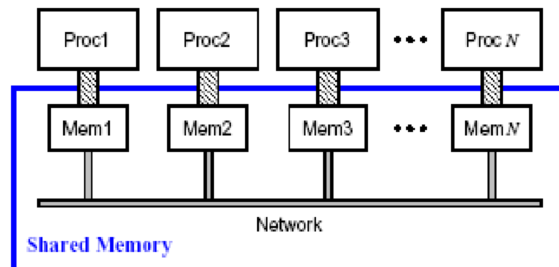
Implementing Shared Memory on Distributed Systems

Sandhya Dwarkadas
University of Rochester

SDSM Progression

- TreadMarks – shared memory for networks of workstations
- Cashmere-2L - 2-level shared memory system
- InterWeave - 3-level versioned shared state

Software Distributed Shared Memory



Why a Distributed Shared Memory (DSM) System?

- Motivation: parallelism utilizing commodity hardware; including relatively high-latency interconnect between nodes
- Comparable to pthreads library in functionality
- Implicit data communication (in contrast to an MPI-style approach to parallelism)
- Presents same/similar environment as shared memory multiprocessor machines

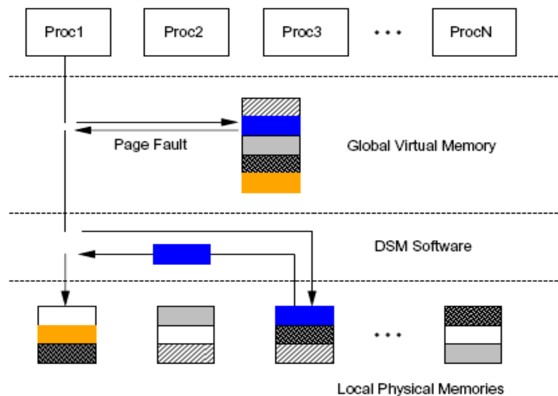
Applications (SDSM)

- Correspondence problem [CAHPP97]
- Object recognition
- Volumetric reconstruction
- Intelligent environments
- CFD in Astrophysics [CS00]
- Genetic linkage analysis [HH94,CBR95]
- Protein folding
- Laser fusion
- “Cone beam” tomography

Detecting Shared Accesses

- Virtual memory – page-based coherence unit
- Instrumentation – overhead on every read and write

Conventional SDSM System Using Virtual Memory [Li 86]



Problems

- Sequential consistency can cause large amounts of communication
- Communications is $$$ on a workstation network (Latency)
- Performance Problem: False Sharing

Goals

- Keep shared memory model
- Reduce communication using techniques such as
  - Lazy Release Consistency (to reduce frequency of metadata exchange)
  - Multiple Writer Protocols (to address false sharing overheads)

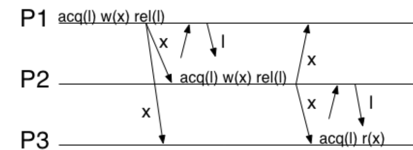
TreadMarks [USENIX'94,Computer'96]

- State-of-the-art software distributed shared memory system
- Page-based
- Lazy release consistency [Keleher et al. '92]
  - Using distributed vector timestamps
- Multiple writer protocol [Carter et al. '91]

API

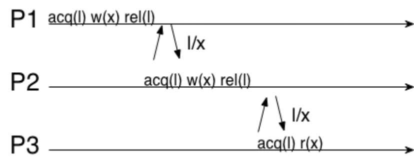
```
tmk_startup(int argc, char ∗ ∗ argv)
tmk_exit(int status)
tmk_malloc(unsigned size)
tmk_free(char ∗ ptr)
tmk_barrier(unsigned id)
tmk_lock acquire(unsigned id)
tmk_lock release(unsigned id)
```

Eager Release Consistency



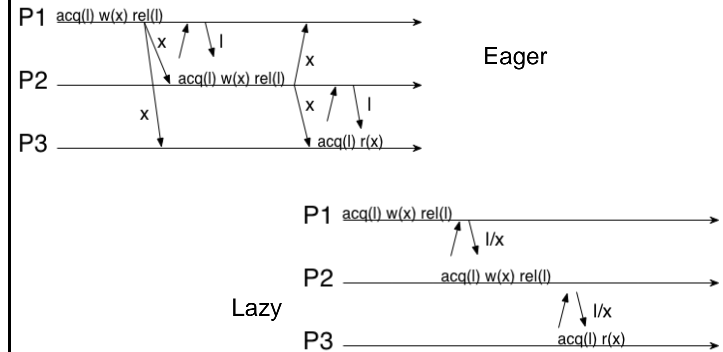
- Changes to memory pages ("x") propagated to all nodes at time of lock release
- Inefficient use of network
- Can we improve this?

Lazy Release Consistency



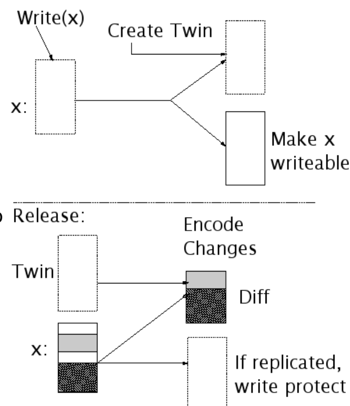
- Synchronization of memory occurs upon successful acquire of lock (“l”).
- More efficient; TreadMarks uses this.
- Changes to memory piggyback on lock acquire notifications

Release Consistency: Eager vs. Lazy

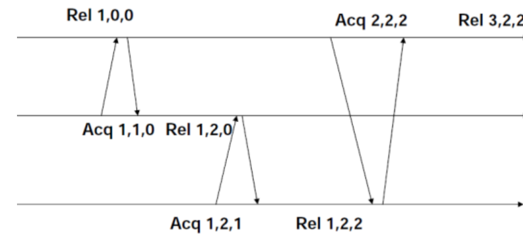


Multiple Writer Protocols

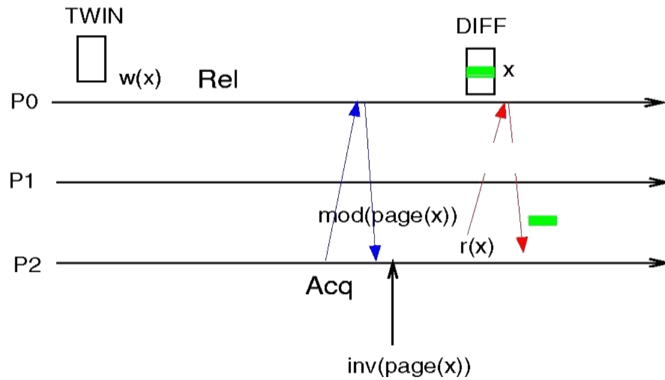
- TreadMarks traps write access to TM pages using VM system
- Copy of page -- a *twin* -- is created
- Memory pages are synced by generating a binary diff of the twin and the current copy of a page
- Remote node applies the diff to its current copy of the page



Vector TimeStamps



Protocol Actions for TreadMarks

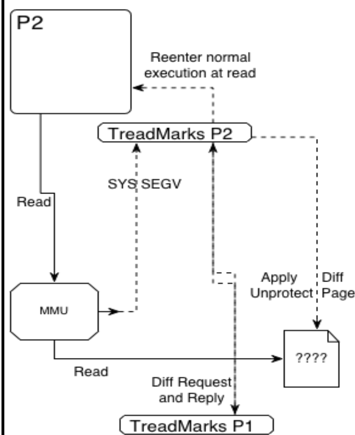


Uses vector timestamps to determine causally related modifications needed

TreadMarks Implementation Overview

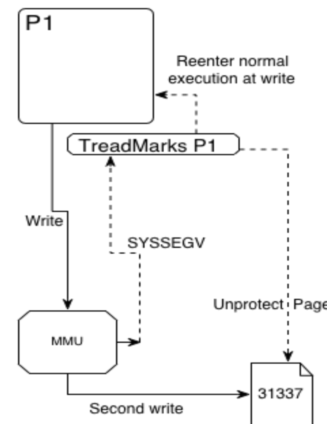
- Totally implemented in userspace
- Provides a TreadMarks heap [malloc() / free()] to programs; memory allocated from said heap is shared
- Several synchronization primitives: barrier, locks
- Memory page accesses (reading or writing) can be trapped by using mprotect()
  - Accessing a page that has been protected causes a SIGSEGV -- segmentation fault
  - TreadMarks installs a signal handler for SIGSEGV that differentiates faults on TreadMarks-owned pages.
- Messages from other nodes use SIGIO handler
- Writing to a page causes an *invalidation* notice rather than a data update

TreadMarks Read Fault Example



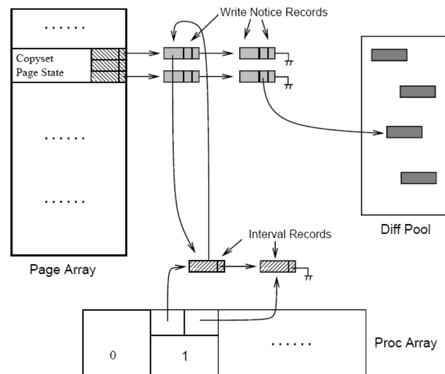
- Remember: a read fault means that the local copy needs to be updated.
- Pages are initially not loaded by diffs.

TreadMarks Write Fault

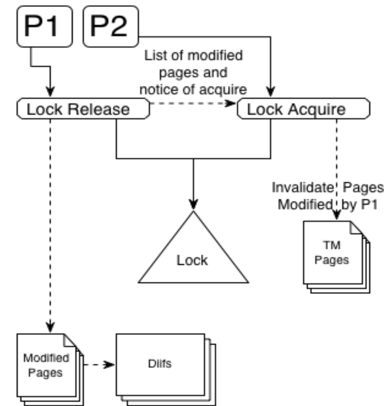


- The program on P1 attempts a write to a protected page
- The MMU intercepts this operation and throws a signal
- The TM signal handler intercepts this signal and determines whether it applies to a TM page
- Flags page as modified, unprotects it, and resumes execution at the write (creating a twin along the way)

Implementation



TreadMarks Synchronization Events



- Let us suppose that P1 has yielded a lock and P2 is acquiring it.
- P1 has modified pages
- Lazy release consistency tells us that an acquiring process needs the changes from the previous holder of the lock.
- P2 flags pages as invalid and uses mprotect() to trap reads and writes to said pages.
- P1 has diffs for its changes up to this synchronization event.

More on Synchronization Events

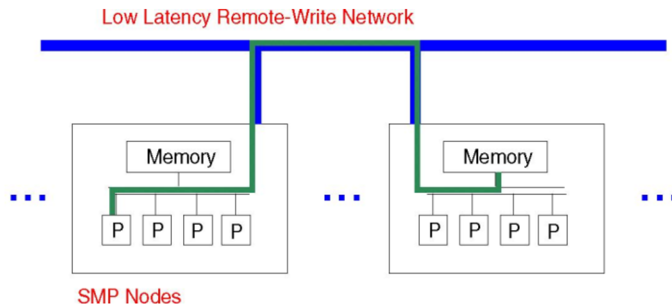
- TM may actually defer diff creation and simply flag that it needs to do a diff at some point. Many programs with high locality benefit from this.
- Set of updated pages (write notices) is constructed by using vector timestamps.
- Each process monitors its writes within each acquire-release block or *interval*.
- The set of write notices sent to an acquiring process consists of the union of all writes belonging to intervals at the releasing node that have not been performed at the acquiring node.

TreadMarks Summary

- Lazy release consistency minimizes the need to push updates to other processors and allows updates to piggyback on lock acquisitions
- Multiple writer protocols minimize false sharing and reduce update size
- Requires no kernel or compiler support
- Not good for applications with a high frequency of communication and synchronization

Cashmere-2L [SOSP'97,TOCS'05]

- Tightly-coupled cost-effective shared memory computing



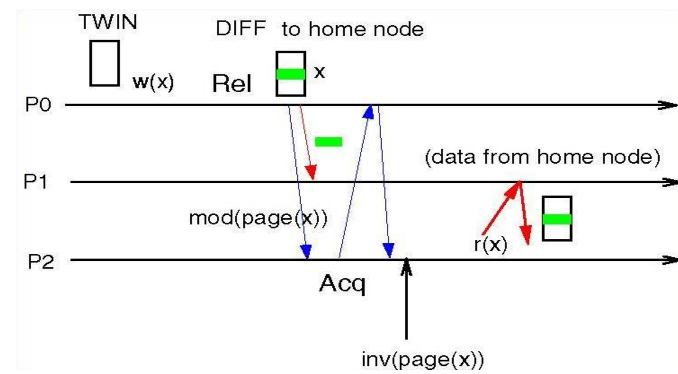
Motivation

- Take advantage of low-latency system-area networks
- Leverage available hardware coherence in SMP nodes

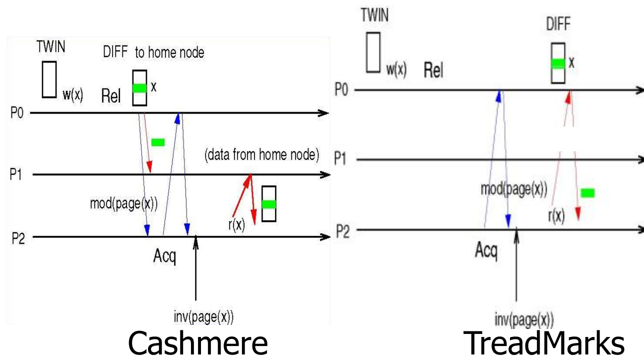
Core Cashmere Features

- Virtual memory-based coherence
  - Page-size coherence blocks
- Data-race-free programming model
  - "Moderately lazy" release consistency protocol
  - Multiple concurrent writers to a page
- Master copy of data at home node
- Distributed directory-based coherence

Protocol Actions for Cashmere-2L



Protocol Actions for Cashmere-2L & TreadMarks

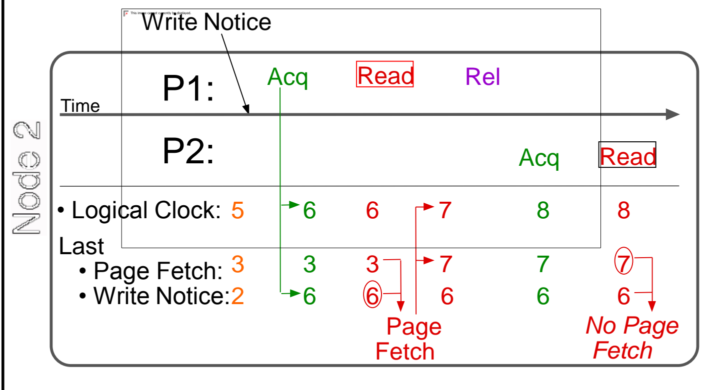


Cashmere TreadMarks

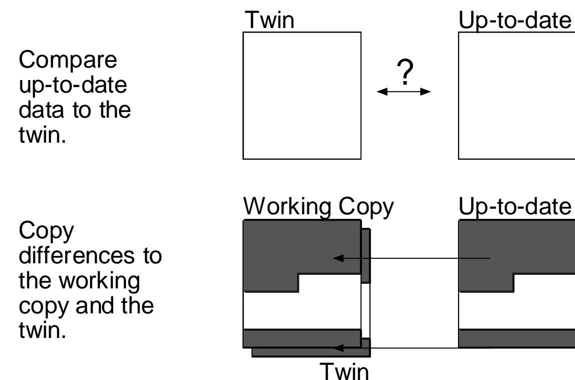
Hardware/Software Interaction

- Software coherence operations performed for entire node
  - Coalesce fetches of data to validate pages
  - Coalesce updates of modified data to home node
- Per-page timestamps within SMP to track remote-SMP events
  - Last write notice received
  - Last home node update
  - Last data fetch

Avoiding Redundant Fetches

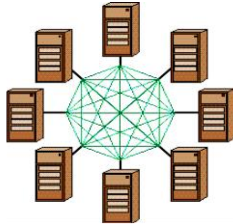


Incoming Diffs



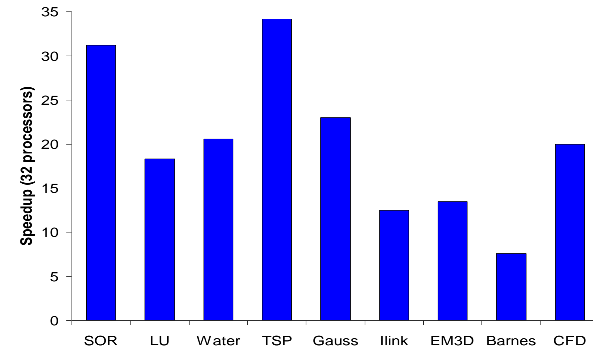
Experimental platform

- 8-node cluster of 4-way SMPs — 32 processors total

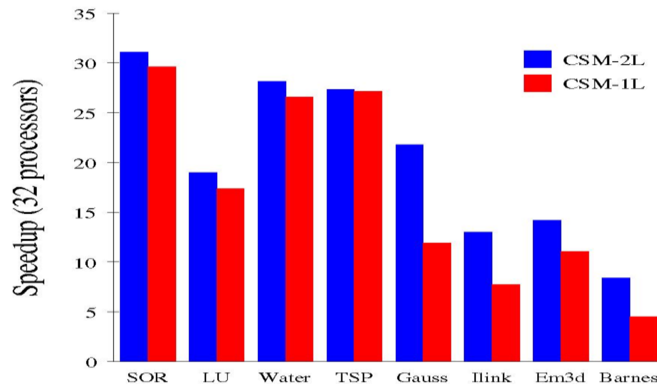


- Early work: 233 MHz EV45s, 2 GB total memory; 5us remote latency; 30 MB/s per-link bandwidth; 60 MB/s total bandwidth
- Nov. 1998: 600 MHz EV56s, 16 GB total memory; 3us remote latency; 70 MB/s per-link bandwidth; >500 MB/s total bandwidth

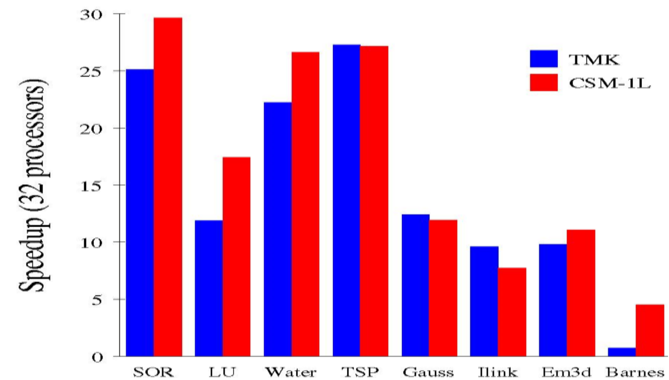
Cashmere-2L Speedups



Cashmere-2L vs. Cashmere-1L



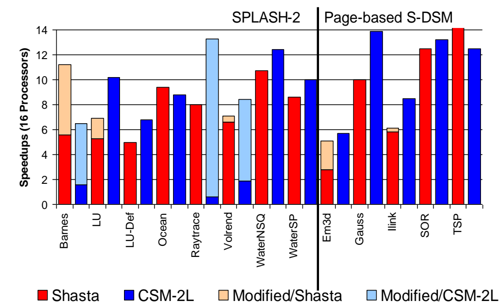
Comparison to TreadMarks



Shasta [ASPLOS'96,HPCA'98]

- 256-byte coherence granularity
- Inline checks to intercept shared accesses
  - Intelligent scheduling and batching of checks
- Directory-based protocol
- Release consistent (can provide sequential consistency)
- Can provide variable coherence granularity

Comparison to Shasta [HPCA'99]



Summary: Cashmere-2L

- Low-latency networks make directories a viable alternative for SDSM [ISCA'97]
- Remote-write capability mainly useful for fast messaging and polling [HPCA'00]
- Two-level design exploits hardware coherence within SMP [SOSP'97]
  - Sharing within SMP uses hardware coherence
  - Operations due to sharing across SMPs coalesced

InterWeave [LCR'00,ICPP'02,ICDCS'03,PPoPP'03, IPDPS'04]

- Shared state in a distributed and heterogeneous environment
- Builds on recent work to create a 3-level system of sharing using
  - hardware coherence within nodes
  - lazy release consistent software coherence across tightly-coupled nodes (Cashmere[ISCA'97,SOSP'97,HPCA'99,HPCA'00])
  - versioned coherence across loosely-coupled nodes (InterAct[LCR'98,ICDM'01])

Motivation

- Most applications cache state
  - e-Commerce
  - CSCW
  - even web pages!
  - multi-user games
  - peer-to-peer sharing

Problems:

- Major source of complexity (design & maintenance)
- High overhead if naively written

Solution: Automate the caching!

- much faster than simple messaging
- much simpler than fancy messaging

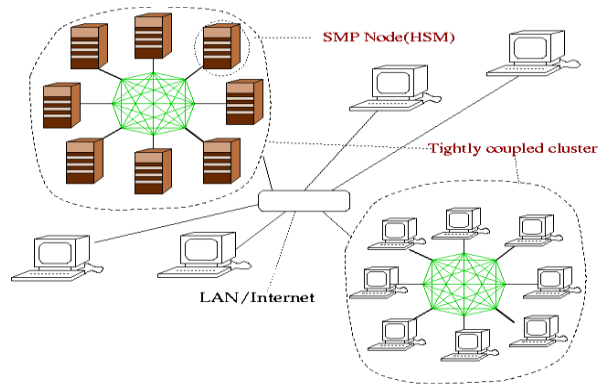
Target Applications (Wide-Area)

- Compute engines with remote satellites
  - Remote visualization and steering (Astrophysics)
- Client-server division of labor
  - Data mining (interactive, iterative)
- Distributed coordination and sharing
  - Intelligent environments

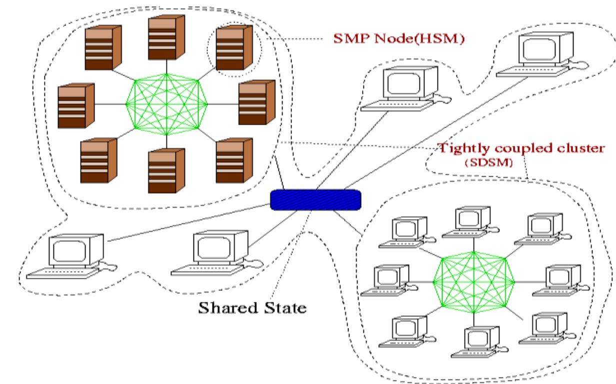
Goal:

- Ease of use
- Maximize utilization of available hardware

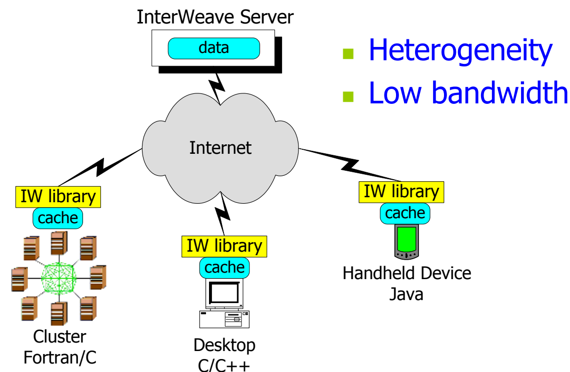
Distributed Environment



Desired Model



InterWeave



- Heterogeneity
- Low bandwidth

InterWeave API

- Data mapping and management as segments
  - `URL server = "iw.cs.rochester.edu/simulation/SegA";`
  - `IW_handle_t h = IW_open_segment(server);`
- Synchronization using reader writer locks
  - `IW_wl_acquire(h), IW_rl_acquire(h)`
- Data allocation
  - `p = (part*) IW_malloc(h, part_desc);`
- Coherence requirement specification
  - `IW_use_coherence(h)`
- Data access using ordinary reads and writes

InterWeave Design Highlights

- *Heterogeneity*
  - Transparently handle *optimized* communication across multiple machine types and operating systems
    - Leverage language reflection mechanisms and the use of an IDL
    - Two-way machine-independent wire format diffing
  - Handle multiple languages
- *Application-specific coherence information*
  - Relaxed coherence models and dynamic views
  - Hash-based consistency
- A *multi-level* shared memory structure
  - Leverage coherence and consistency management provided by the underlying hardware and software distributed memory systems