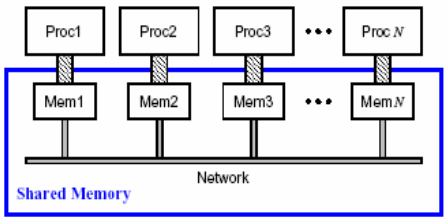
Implementing Shared Memory on Distributed Systems

Sandhya Dwarkadas
University of Rochester

Overview

- 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 synchronization of memory (in contrast to an MPI-style approach to parallelism)
- Presents same/similar environment as shared memory multiprocessor machine

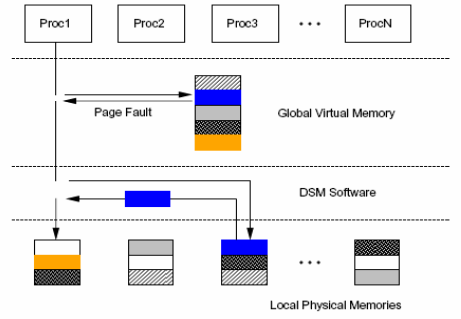
Applications (SDSM)

- 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)

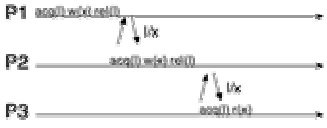
TreadMarks [USENIX'94,Computer'96]

- 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]

Eager Release Consistency

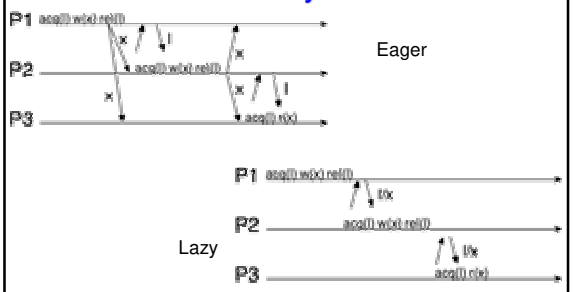
- Changes to memory pages ("x") propagated to all nodes at time of lock release
- Inefficient use of network
- Can we improve this?

Lazy Release Consistency



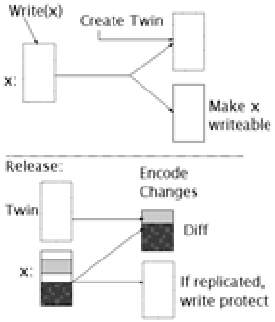
- Synchronization of memory occurs upon successful acquire of lock ("l").
- More efficient; TreadMarks uses this.
- Changes to memory piggyback on lock acquire notifications

Release Consistency: Eager vs. Lazy

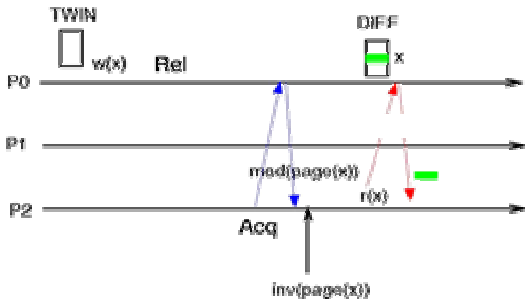


Multiple Writer Protocols

- 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



Protocol Actions for TreadMarks



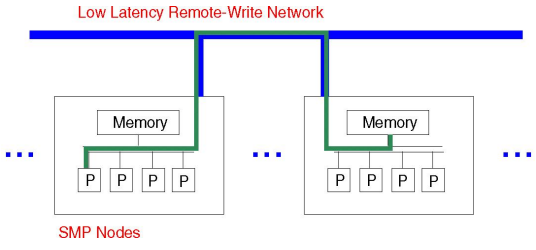
Uses vector timestamps to determine causally related modifications needed

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

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

- Tightly-coupled cost-effective shared memory computing



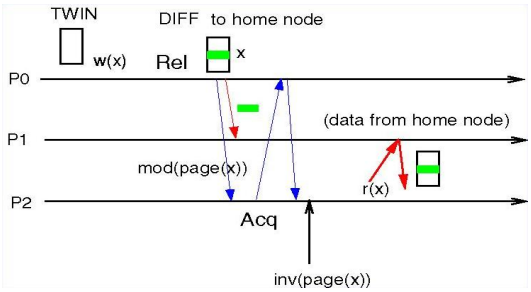
Motivation

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

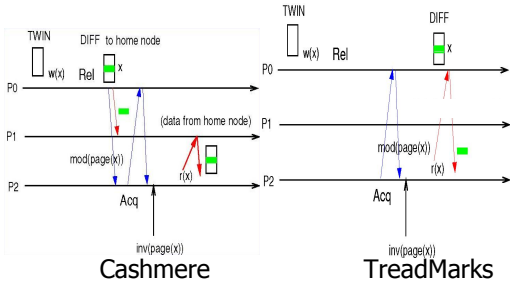
Core Cashmere Features

- 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



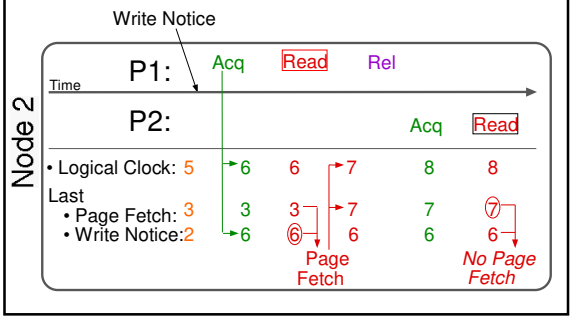
Protocol Actions for Cashmere-2L & TreadMarks



Hardware/Software Interaction

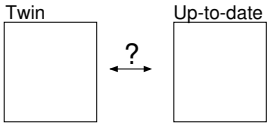
- Software coherence operations performed for entire node
  - Coalesce fetches of data to validate pages
  - Coalesce updates of modified data to home node
- Per-page timestamps within SMP to track remote-SMP events
  - Last write notice received
  - Last home node update
  - Last data fetch

Avoiding Redundant Fetches

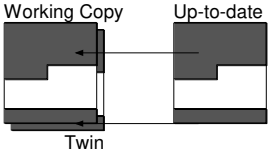


Incoming Diffs

Compare up-to-date data to the twin.

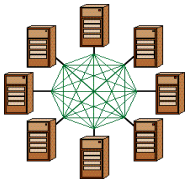


Copy differences to the working copy and the twin.



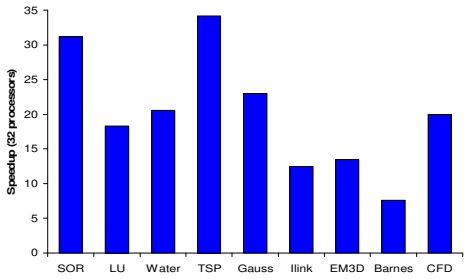
Experimental platform

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

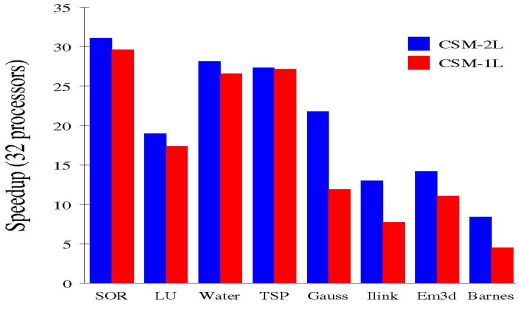


- Early work: 233 MHz EV45s, 2 GB total memory; 5us remote latency; 30 MB/s per-link bandwidth; 60 MB/s total bandwidth
- Nov. 1998: 600 MHz EV56s, 16 GB total memory; 3us remote latency; 70 MB/s per-link bandwidth; >500 MB/s total bandwidth

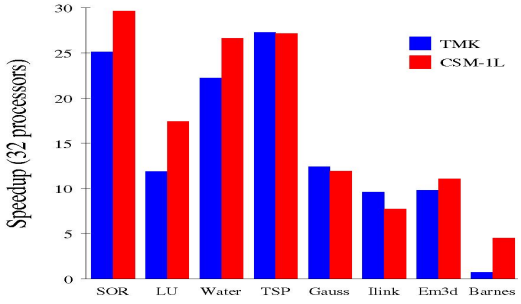
Cashmere-2L Speedups



Cashmere-2L vs. Cashmere-1L



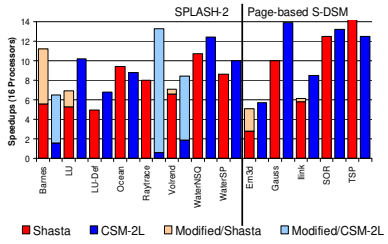
Comparison to TreadMarks



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

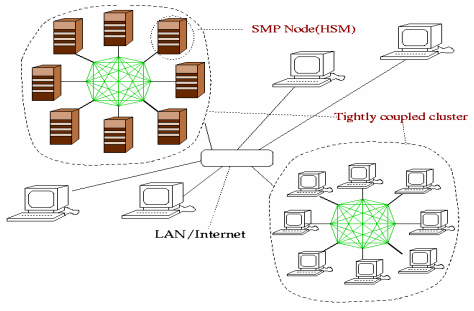
Target Applications (Wide-Area)

- 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

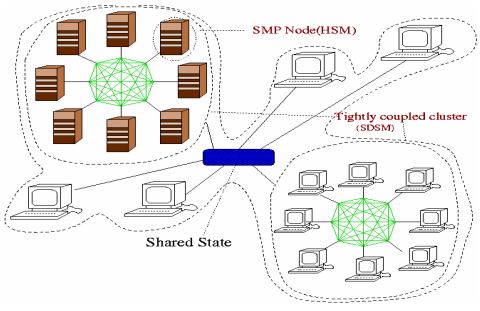
Goal:

- Ease of use
- Maximize utilization of available hardware

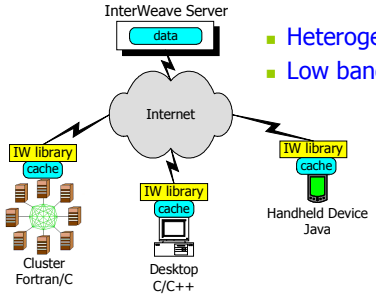
Distributed Environment



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

Language and Machine Heterogeneity

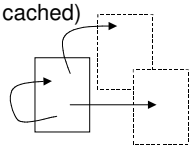
- Arbitrary data types (including pointers) supported across heterogeneous languages and machines
  - types declared in IDL; compiled to typedefs and metadata
  - automatic translation and swizzling for optimized data communucation; wire-format diffs

Heterogeneity

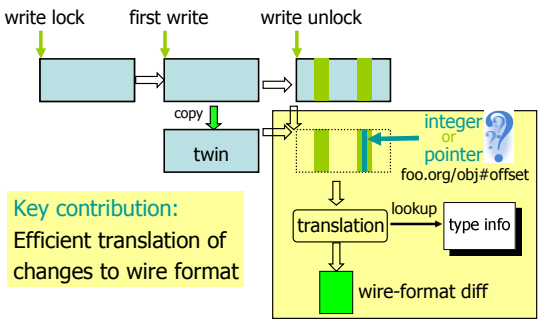
- Type descriptor in `malloc` call created by XDR compiler
- All communication with manager (server) converts to and from wire format
- Segments are self-descriptive, for debugging and perusal; may be laid out differently on different machines, for cache locality
- Machine-independent pointer (MIP) is also a URL:
  - http://iw.cs.rochester.edu/iw/demo#14#5
  - domain/path#block[#offset]

Pointer swizzling [Wilson '92]

- When data comes in, turn each MIP into a local ptr
- When data goes out, turn each ptr to a MIP
- Can dereference pointer so long as target is locked (ensures it's locally cached)
- "Expanding frontier", as in dynamic linking
- Can also convert explicitly between MIPs and ptrs, for bootstrapping



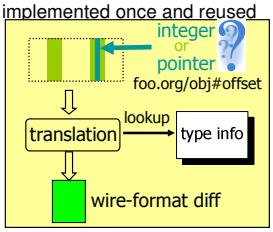
Change Detection and Translation



Key contribution:
Efficient translation of changes to wire format

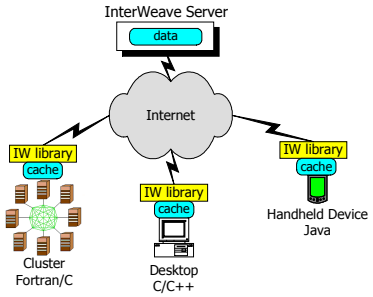
Change Detection and Translation

- Complexity comparable to sophisticated language reflection mechanism
- Efficiency comparable to RPC XDR [ICDCS'03]
  - Aggressive optimizations implemented once and reused by all applications



Key contribution:
Efficient translation of changes to wire format

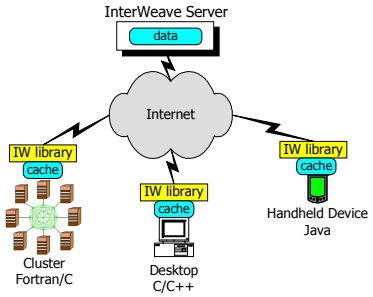
Sharing Among Multiple Languages



InterWeave Support for Java

- InterWeave allows Java application to share state with C and Fortran applications seamlessly
  - Important for bridging between legacy Fortran applications and newer Java GUI programs
- The Java implementation is based on the Kaffe JVM and the InterWeave client library for C
  - Interaction between the JVM and the InterWeave runtime system
  - Mapping between Java objects and C data structure
  - Fine-grained access control to shared objects

Maintaining Coherence



Coherence

- Without application-level information, traditional coherence models
  - Require the most recent update whenever there are modifications at synchronization points
  - Always update at the granularity of the coherence unit
    - Page-based systems always update at page unit
- Programmers usually hold high-level information that allows more aggressive optimizations

Coherence Optimization

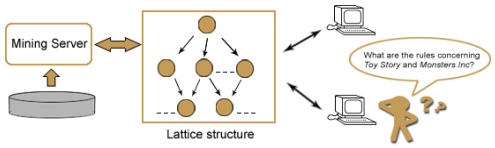
- Relaxed reader-writer locks
  - Writer grabs current version (does *not* exclude readers)
  - Reader checks to see if current cached copy (if any) is "recent enough"
- Cache whole segments; server need only keep most recent version
- Diff-based updates (both ways) based on page and block time stamps

High-level Coherence Information

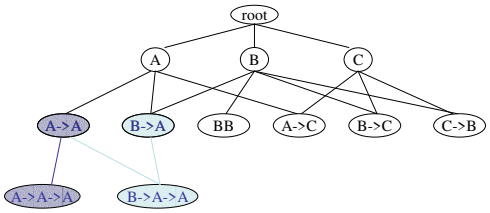
- Relaxed coherence models
  - Optimize communication bandwidth by postponing (skipping) unnecessary updates
  - Strict, Full, Temporal, *Diff*, Delta, and Null
- Dynamic views
  - Optimize communication bandwidth by limiting the region of updates
  - The scope of the region can be statically specified or dynamically expanded

Datamining

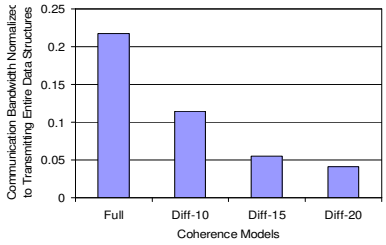
- Incremental and Interactive Datamining
  - Interactive service by sharing (caching) the summary structure at clients
  - Optimizing cache updates by allowing staleness and partial updates



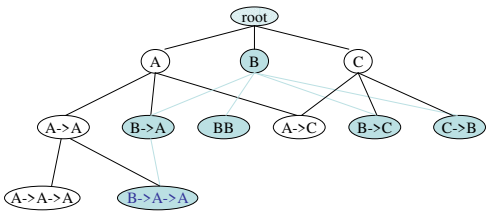
Diff Coherence



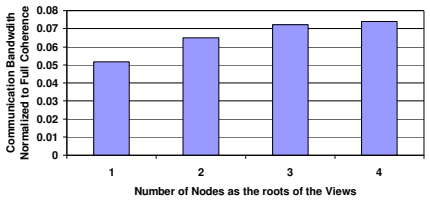
Bandwidth Reduction



Views with Dynamic Scope



Bandwidth Reduction



Consistency

- Need to respect happens-before
- Invalidate cached version of A that is older than version on which newly-obtained version of B depends
- Ideally want to know entire object history
- Can approximate using hashing
  - slot i of vector contains timestamp of most recent antecedent hashing to i
  - invalidate A if B.vec[hash(A)] is newer

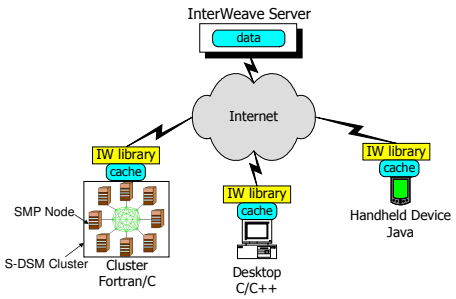
Transaction Support and RPC [IPDPS'04]

- Transaction metadata table (TMT) keeps track of
  - Locks acquired
  - Version numbers of locked segments
  - Segments modified
- The TMT allows processes within a transaction to share data invisible to other processes and to exchange data modifications
- It is passed between caller and callee in each RPC call and return

Another TM System: Distributed Multi-Versioning [PPoPP'06]

- Take advantage of natural replication in a distributed system
- Page-based conflict detection
- Isolate read-only transaction through selective application of outstanding modifications
- Cluster configurations
  - Update anywhere
  - Master update

Multi-Level Shared State Support



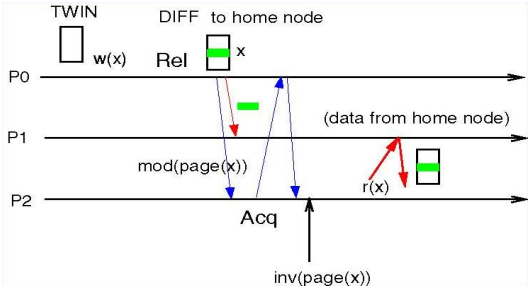
Multi-level Shared State

- Making shared state within a cluster accessible to remote nodes via InterWeave segments (level-3)
- Logical extension from 2-level distributed memory systems
  - Level-1: hardware shared memory support within multiprocessor machines
  - Level-2: software distributed shared memory systems within tightly coupled clusters
- Within the 3-level system,
  - Leverage level-2 software system functions for level-3 (IW) operations whenever possible

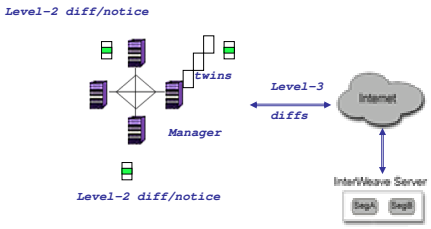
Two-Level Protocol

- HW coherence within nodes, SW coherence among nodes
- Avoid redundant fetches of data and home node updates
- Coalesce global meta-data operations
- Twins and *two-way* diffs to eliminate inter-processor synchronization
  - No TLB shootdown
- Largely asynchronous (lock-free meta-data)
- Results first reported at SOSP '97

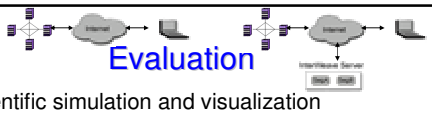
Protocol Actions for Cashmere-2L



Integrating InterWeave and Cashmere

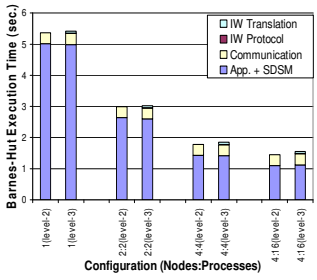


Evaluation



- Scientific simulation and visualization
  - A simulator running on a 4-node Cashmere cluster and visualization satellites running on a remote node
  - Connected with direct TCP connections and InterWeave shared state
- Quantify level-3 cost over the underlying 2-level system
  - Communication cost
  - Interweave cost for maintaining extra level-3 information

Efficiency of 3-Level Systems



Status of InterWeave

- Consists of 45,000 lines of C/C++ code
  - InterWeave server – management of persistent segment storage
  - Client-side library – coherence, heterogeneity, and multi-level support
- Supports
  - OS's: NT, Tr64 Unix, Solaris, Linux, IRIX, AIX
  - Architectures: x86, Alpha, Sparc, MIPS, Power4
  - Languages: C, C++, Java, Fortran 77, Fortran 90
- Compatible with using XDR RPC
  - Use RPC to move computation
  - Use InterWeave to share state

Applications

- Scientific visualization and steering
  - stellar dynamics (Astroflow)
  - inertial confinement fusion (DRACO)
- Games (MazeWars)
- Intelligent environments
  (object tracking in CFH smart medical home)
- Remote interactive datamining
- Internet Caching Protocol (ICP) server

See
http://www.cs.rochester.edu/research/interweave

Summary: InterWeave

InterWeave integrates:

- Function shipping
  - Remote invocation provides---Sun RPC, Java RMI, etc.
- Data shipping/caching
  - InterWeave provides a global, persistent store.
- Transactions
  - protect against system failures and race conditions.