TreadMarks: Shared Memory Computing on Networks of Workstations

Authors: Cristiana Amza, Alan L. Cox, Sandhya Dwarkadas, Pete Keleher, Honghui Lu, Ramakrishnan Rajamony, Weimin Yu, Willy Zwaenepoel

Presented by: Jonathan Williams

Why a Distributed Shared Memory (DSM) System?

- Motivation: parallelism utilizing commodity hardware; including relatively high-latency interconnect between nodes
- Metaphor: Memory appears a single contiguous block across nodes. Library provides synchronization API.
- Comparable to pthreads library in functionality
- Implicit synchronization of memory (when contrasted to MPI-ish approach to parallelism)
- Presents same environment as shared memory multiprocessor machine

Methods of Implementing DSM

- What general approaches can you think of?
- Kernel
  - Add system calls to support DSM.
  - Modify VM system.
- Compiler extensions for DSM.
  - Can you think about any problems?
  - What about code using pointers?
- Runtime Library
  - But how????

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.
- More details later…

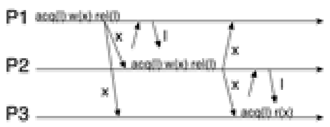
Consistency Models in a DSM System

- Obviously imposing a strict *sequential consistency* model on DSM is be prohibitively slow. Example: IVY DSM
- Each write requires synchronization with every node holding a copy of the page!
- System only allows one node to write to a page at a time.
- If two processes are contending for write access to a page (as in false sharing), page goes back and forth across network.
- Fortunately, we can utilize one of our relaxed models quite easily -- *release consistency.*

Consistency Models in a DSM System, Part 2

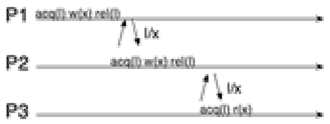
- Release consistency requires us to synchronize between conflicting accesses to a shared memory location. Normal operations do not have a strict order with respect to each other but are ordered in respect to to synchronization operations.
- The purpose of synchronization is to prevent access to certain memory locations until synchronization is complete.
- Consequentially, we do not need to propagate modifications to shared memory pages until a synchronization operation completes.

Eager Release Consistency



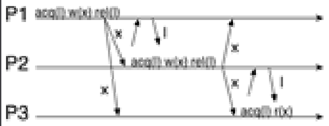
- Changes to memory pages (“x”) propagated to all nodes at time of lock release.
- Inefficient use of network
- Can we improve this?

Lazy Release Consistency

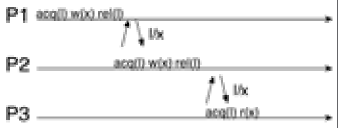


- Synchronization of memory occurs upon successful acquire of lock (“l”).
- More efficient; TreadMarks uses this.
- Changes to memory piggyback on lock acquire notifications

Release Consistency: Eager vs. Lazy



Eager



Lazy

Release Consistency Analysis

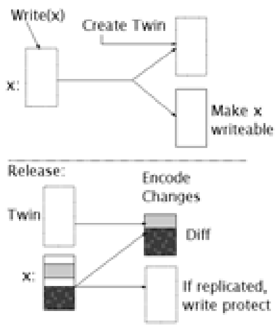
- Let’s think about the number of page update messages over the lifetime of a TreadMarks program.
- The “Eager” model sends a reply to every node for each release operation.
- The “Lazy” model sends a single reply for each acquire operation.

Multiple Writer Protocols

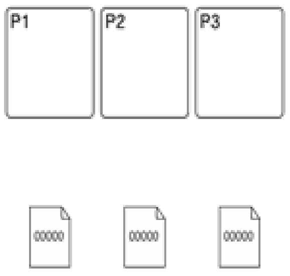
- Suppose P1 and P2 are writing to X1 and X2.
- What happens if X1 and X2 share the same coherency unit (in this case a memory page)?
- False sharing, oh no!
- Solution: *Multiple Writer Protocols*

Multiple Writer Protocols

- TreadMarks traps write access to TM pages using VM system. (Again, more on this later)
- 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.

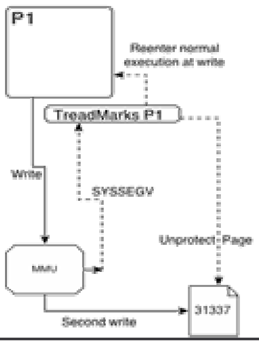


TreadMarks Examples



- Let us suppose that each of our TreadMarks hosts has an up to date copy of a page.
- Up to Date Pages are flagged ReadOnly until Write
- What if P1 wants to write to a page?

TreadMarks Write Fault

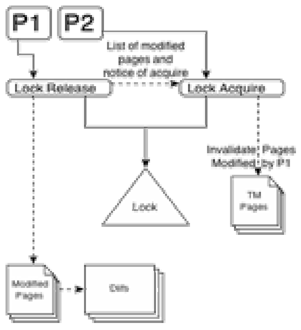


- The program on P1 attempts a write to a protected page
- The MMU intercepts this operation and throws a signal
- The TM signal handler intercepts this signal and determines whether it applies to a TM page
- Flags page as modified, unprotects it, and resumes execution at the write
- (Twin creation is omitted.)

More on the Write Fault Example

- TreadMarks does not unprotect the page itself; this is facilitated by the mprotect() kernel call and the MMU
- So how do changes propagate to the other processors?
- Synchronization events!
- Remember: TM programs need to enforce consistency between synchronization events.

TreadMarks Synchronization Events

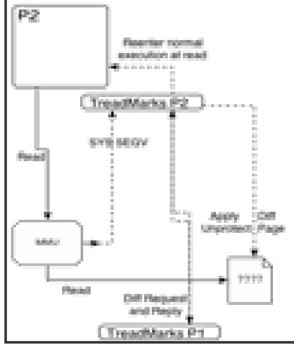

P1
P2
List of modified pages and notice of acquire
Lock Release
Lock Acquire
Invalidate Pages Modified by P1
Lock
TM Pages
Modified Pages
Diffs

- 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 Read Fault Example



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

Other Approaches to DSM to consider

- IVY - Strict sequential ordering. Every write to a page with other valid copies on network requires invalidate message. Yuck!
  - TreadMarks is hundreds of percent faster for real applications
- Munin - Eager release consistency, update protocol. Implementation simpler than TreadMarks, but additional message traffic not work it. (Keleher's thesis shows that even invalidate protocols using Eager RC are worse)
- Midway - Entry Consistency; explicit associated of shared memory structures and sync objects. Perhaps more efficient than TreadMarks?
- Linda - access methods for shared objects. More optimization possible?

Any Questions?