TreadMarks: Distributed Shared Memory on Standard Workstations and Operating Systems

- Proceedings of the Winter Usenix Conference, January 1994 -

Pete Keleher, Alan L. Cox, Sandhya Dwarkadas and Willy Zwaenepoel

Presented by
Younghyun Jo
Kristian Fjeld Hasselknippe

2013/10/15
Outlines

- Introduction
- Design : Techniques
  - Lazy release consistency
  - Multiple writer protocol
- Implementation
- Evaluation
Introduction
Distributed Shared Memory

- Distributed Shared Memory (DSM)
  - Enables processes on different machines to share memory, even though the machines physically do not share memory.

![Diagram of Distributed Shared Memory](image)

**Figure 1** Distributed Shared Memory
Distributed Shared Memory

- DSM is an attractive approach
  - Easier to use than a message passing
  - Programmers can focus on algorithmic development rather than on managing partitioned data sets and communicating values

- Previous DSM solutions are not widely available
  - In-house research platforms
  - Kernel modifications
  - Communication overhead followed by poor consistency protocol
  - False sharing
TreadMarks’ design focus

- Commercially available workstations and operating systems
  - Standard Unix system on DECstation
  - User-level DSM implementation

- To reduce the amount of communication
  - Lazy release consistency

- To avoid false sharing
  - Multiple-writer protocols
Design
: LRC, Multiple writer protocol
Release Consistency

- Release Consistency (RC)
  - Delay making its changes visible to other processors until certain synchronization accesses occurs
  - Acquire(), Release()
    - Roughly correspond to synchronization operations on a lock
  - Two types of RC
    - ERC (Eager Release Consistency)
    - LRC (Lazy Release Consistency)
Eager Release Consistency

- Eager Release Consistency (ERC)
  - Eager release consistency (ERC) postpones sending the modifications to the next release

![Diagram illustrating Eager Release Consistency](image)

A situation

<table>
<thead>
<tr>
<th>Proc 0</th>
<th>Proc 1</th>
</tr>
</thead>
<tbody>
<tr>
<td>x0 = 0</td>
<td>x0 = 0</td>
</tr>
<tr>
<td>x1 = 0</td>
<td>x1 = 0</td>
</tr>
</tbody>
</table>

Network

Proc 0

- Acquire(L)
- W(x1)1
- Release(L)

Proc 1

- R(x1)0
- Apply changes
- Acquire(L)
- R(x1)1
- Release(L)
Lazy Release Consistency

- Lazy Release Consistency (LRC)

Lazy release consistency (LRC) postpones sending of modifications until a remote processor actually needs them.

A situation

Proc 0

<table>
<thead>
<tr>
<th>x0</th>
<th>x1</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>0</td>
</tr>
</tbody>
</table>

Proc 1

<table>
<thead>
<tr>
<th>x0</th>
<th>x1</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>0</td>
</tr>
</tbody>
</table>

How do we know what were the previous modifications?

1. Write notification
2. Vector Clock

Write Notice (X=1 VC<2,1>)

Request: Sending VC<3,2>

Actual value update is depend on implementation.
Comparison of communication between LRC and ERC

- LRC is more effective than ERC when we have many processors in a distributed system

Communications of ERC

- Proc 0: Acquire(L) → W(x1) → Release(L)
- Proc 1: R(x1) → time → R(x1)
- Proc 2: R(x1) → time → Apply changes → R(x1) → time → Release(L)
- Proc 3: R(x1) → time → W(x1) → time → Acquire(L)
Comparison of communication between LRC and ERC

- LRC is more effective than ERC when we have many processors in a distributed system.
Multiple writer protocol

- False sharing is a situation in which two or more processes access different variables within a page.

- Single writer protocol leads to unnecessary communication.
  - False sharing

- Multiple-writer protocol
  - It is possible that several processes make modifications to different variables at the same page.
Multiple writer protocol

- Multiple-writer protocol: Munin’s solution
  - Modifications are merged at the next synchronization operation in accordance with the definition of RC

- In order to capture the modifications
  - Twin: DSM software makes a copy of the page
  - Diff: the twin and the current copy can later be compared to create a diff
Multiple writer protocol

- **Twin/Diff**

- **TreadMarks** allows diff creation to be postponed until the modifications are requested
  - LRC with multiple-writer protocol vs ERC with multiple-writer protocol
Implementation
Implementation

Overall

![Diagram of TreadMarks Data Structures]

Figure 2  Overview of TreadMarks Data Structures
Implementation

- Data Structures
  - PageArray
    - Current state
    - A approximate copyset
    - Pointer to a list of write notice records
  - ProcArray
    - Pointer to a list of interval records
  - Interval records
    - Vector timestamp
  - Write notice records
  - Diff pool

![Diagram of TreadMarks Data Structures](image)

**Figure 2** Overview of TreadMarks Data Structures
Implementation

- Access misses
  - If miss -> request a copy from a member of the pages approximate copy set.
  - If write notice are present -> obtain diffs and apply

- Garbage collection
  - To reclaim space used by PageArray, interval records, write notices and diffs
  - Each processor updates the copy set for every page

- Unix aspects
  - Communication through UDP/IP or ATM LAN / AAL 3/4 protocol on ATM LAN
  - Neither protocols guarantees reliable delivery
Evaluation
Experimental Environment

- Experimental environment
  - 8 DECstation-5000/240’s running Ultrix V4.3
  - 100-Mbps ATM LAN / 10-Mbps Ethernet

- Implementation
  - LRC with multiple-writer protocol
  - ERC with multiple-writer protocol
Speedups Obtained on TreadMarks

- Speedups obtained on TreadMarks
- Benchmark applications
  - Water – molecular dynamic simulation
  - Jacobi – successive over-relaxation
  - TSP – branch & bound algorithm to solve the traveling salesman problems
  - Quicksort
  - ILINK – genetic linkage analysis

<table>
<thead>
<tr>
<th></th>
<th>Water</th>
<th>Jacobi</th>
<th>TSP</th>
<th>Quicksort</th>
<th>ILINK</th>
</tr>
</thead>
<tbody>
<tr>
<td>Input</td>
<td>343 mols</td>
<td>2000x1000</td>
<td>19-city tour</td>
<td>256000 integers</td>
<td>CLP</td>
</tr>
<tr>
<td>Time (secs)</td>
<td>15.0</td>
<td>32.0</td>
<td>43.8</td>
<td>13.1</td>
<td>1113</td>
</tr>
<tr>
<td>Barriers/sec</td>
<td>2.5</td>
<td>6.3</td>
<td>0</td>
<td>0.4</td>
<td>0.4</td>
</tr>
<tr>
<td>Locks/sec</td>
<td>582.4</td>
<td>0</td>
<td>16.1</td>
<td>53.9</td>
<td>0</td>
</tr>
<tr>
<td>Msgs/sec</td>
<td>2238</td>
<td>334</td>
<td>404</td>
<td>703</td>
<td>456</td>
</tr>
<tr>
<td>Kbytes/sec</td>
<td>798</td>
<td>415</td>
<td>121</td>
<td>788</td>
<td>164</td>
</tr>
</tbody>
</table>

Figure 4 Execution Statistics for an 8-Processor Run on TreadMarks

Figure 3 Speedups Obtained on TreadMarks

Many short messages. Each message is protected by lock -> a very fine-grained application
Execution Time Breakdown

Execution time breakdown

- Execution time breakdown

- Time spent executing Unix kernel and library code
- Time spent executing Unix kernel and library code
- The amount of time spent waiting for Unix and TreadMarks operations on other nodes
- Detecting changes to shared pages, including twin & diff creation
- Handling consistency information

- Kernel operations to support communication
- Kernel operations to user-level memory management, page protection changes

- Communication
- Synchronization

Figure 5: TreadMarks Execution Time Breakdown

Figure 6: Unix Overhead Breakdown

Figure 7: TreadMarks Overhead Breakdown

Distributed Information Processing, Fall 2013
Execution Time for Water

- Execution time for water
  - Water: the highest communication overhead
  - Comparison of two other communication subtraces
    - UDP over ATM / UDP over an Ethernet
Lazy vs. Eager Release Consistency

- LRC with multiple-writer protocol vs ERC with multiple-writer protocol
  - Speedups / Message rate

**Figure 9**  Comparison of Lazy and Eager Speedups

**Figure 10**  Message Rate (messages/sec)
Lazy vs. Eager Release Consistency

- LRC with multiple-writer protocol vs ERC with multiple-writer protocol
  - Data rate / Diff creation rate

**Figure 11**  Data Rate (kbytes/sec)  
**Figure 12**  Diff Creation Rate (diffs/sec)
Conclusion

- DSM is a user level implementation which runs on DECstation-5000/240’s machine

- It reduces the cost of communication
  - Lazy release consistency
  - Multiple-writer protocols with Lazy diff creation

- It gives good speedups for many benchmark applications

- It conclude that user-level DSM is a viable technique for parallel computation on clusters of workstations connected by suitable networking technology
Thank you