Software Cache Coherence with Memory Scaling

Nikos Hardavellas[‡], Leonidas Kontothanassis[‡], Rishiyur Nikhil[‡], and Robert Stets[†]
[‡] DEC Cambridge Research Lab
One Kendall Sq.. Bldg 700
Cambridge, MA, 02139
{nikos,kthanasi,nikhil}@crl.dec.com

[†] Department of Computer Science
University of Rochester
Rochester, NY, 14627
stets@cs.rochester.edu

April 16, 1998

Software-coherent, distributed shared memory has received conciderable amount of attention as an attractive programming model that leverages inexpensive hardware. DSM systems running on clusters of symmetric multiprocessors provide good performance for a large number of applications and provide an easy path to incremental scalability.

Most DSM systems [3, 1, 19] use the main memory of each node in the cluster as a third level cache and they migrate and replicate data in that memory. Since computer memories tend to be much larger than caches DSM systems have largely ignored memory capacity issues, assuming there is always enough space in main memory into which to replicate data. This design decision places a hard limit on the scalability of DSM applications running on clusters. While adding more nodes in the cluster increases the amount of computational power available to the application it has no effect on the amount of shared memory the application can use, thus limiting the sizes of problems that can be tackled with a DSM approach. In early studies we have seen significant performance degradation when an application attempts to access more memory than is available on a single node of the cluster.

We have designed a new protocol based on the Cashmere DSM system that attempts to take advantage of *all* the memory available in a cluster. The key insight behind the protocol is that while a node may access a larger amount of memory than it has locally, at any point in time the working set is likely to be smaller than the amount of local memory. We have therefore augmented the protocol with the capability to evict coherence blocks when memory pressure gets high.

Our protocol takes advantage of DSM knowledge in order to minimize the cost of evicting page from a node. In particular we catagorize pages in three categories:

Read-only pages: Such pages have been replicated in read-only mode in the node. Furthermore an uptodate copy of the page exists in the home node and thus it can be dropped safely without requiring any communication.

Read-write pages: Such pages have been replicated and subsequently modified in the node. Before dropping them the changes have to be sent to the home node and other sharing nodes have to be notified of the write event.

Home pages: These are pages for which the node is the home node. Furthermore the home-node copy may be the only available copy in the system. For these pages we need to find a new home node that will start serving requests on their behalf and send the page content to that new home node.

We have implemented the protocol on a cluster of DEC AlphaServers connected by a Memory Channel network. Early experimentation shows that in the absence of memory pressure the new protocol imposes minimal (less than 3%) overhead when compared to a protocol that does not attempt to evict coherence blocks.

We have also looked at two eviction policies. The first attempts to minimize the cost of evictions and selects pages based on how expensive it would be to evict them (i.e. select read-only before read-write, and read-write before home pages). This policy results in unaccaptably bad performance for some applications as it can lead to thrashing. We are also experimenting with LRU which appears to provide better results, and upcalls to the application so that the application writer can provide hints as to which data may be unnecessary at any particular point in time.

We believe that eviction-capable DSM protocols add an important dimension to the scalability properties of clusters making them more attractive as compute servers for very large applications.

References

[1] C. Amza, A. L. Cox, S. Dwarkadas, P. Keleher, H. Lu, R. Rajamony, W. Yu, and W. Zwaenepoel. TreadMarks: Shared Memory Computing on Networks of Workstations. In *Computer*, to appear.

[2] D. J. Scales, K. Gharachorloo, and C. A. Thekkath. Shasta: A Low Overhead, Software-Only Approach for Supporting Fine-Grain Shared Memory. In *Proceedings of the Seventh International Conference on Architectural Support for Programming Languages and Operating Systems*, Boston, MA, October 1996.

[3] R. Stets, S. Dwarkadas, N. Hardavellas, G. Hunt, L. Kontothanassis, S. Parthasarathy, and M. L. Scott. CSM-2L: Software Coherent Shared Memory on a Clustered Remote-Write Network. In *Proceedings of the Sixteenth ACM Symposium on Operating Systems Principles*, Saint Malo, France, October 1997.