Distributed Shared Memory: Where We Are and Where We Should Be Headed

John B. Carter
Dilip Khandekar
Linus Kamb
Computer Systems Laboratory
University of Utah

Abstract

It has been almost ten years since the birth of the first distributed shared memory (DSM) system, Ivy. While significant progress has been made in the area of improving the performance of DSM and DSM has been the focus of several dozen PhD theses, its overall impact on "real" users and applications has been small. The goal of this paper is to present our position on what remains to be done before DSM will have a significant impact on real applications. More specifically, we reflect on what we believe have been the major advances in the area, what the important outstanding problems are, and what work needs to be done. Finally, we describe a modest step towards solving these problems, the Quarks DSM system.

1 How We Got to Where We Are

In 1986, Kai Li published his PhD dissertation entitled, "Shared Virtual Memory on Loosely Coupled Microprocessors," thus opening up the field of research that is now known as distributed shared memory or DSM. Although looking back at this work after ten years may make it seem obvious (demand paging + coherence = DSM), we believe that this is more a tribute to the quality of the work than anything else. Since this work, there has been a huge amount of work done to extend the idea to other areas (e.g., distributed object based systems[14, 5, 1, 20, 10] and operating systems[23, 7, 2]) and to improve its performance[11, 4, 24, 15]. However, the impact of DSM on non-research users and applications has been very small. Why is that?

The earliest DSM systems[17, 11, 14, 7] provided the basic functionality of a virtually shared address space spanning a network of machines, but their use of the same coherence protocols as shared memory hardware resulted in poor performance for applications with even a moderate amount of fine-grained sharing. The reasons for this were that the unit of coherence tended to be large (pages in the case of Ivy[17] and Mirage[11] and objects in the case of Emerald[14] and Clouds[7]), the mechanisms used to maintain coherence were heavy weight (page faulting in the case of Ivy, Mirage, and Clouds; object migration in the case of Emerald), and the number of coherence actions, and thus the amount of communication, was excessive (due to the frequent read misses induced by sequentially consistent write-invalidate protocols in Ivy and Mirage and the frequency of object invocation in Clouds and Emerald). Several performance optimizations were introduced to combat these problems, including object locking in Clouds, user-specifiable object placement in Emerald, and request freezing in Mirage, but these optimizations only made a small dent in the problem and research into DSM systems hit a lull.

The second major leap in DSM research came when researchers adopted the relaxed consistency models developed for shared memory hardware and began to take advantage of the greater flexibility possible in a software implementation of coherence[4, 15, 24, 20]. The major contributions of Munin[4], TreadMarks[15], Midway[24] and DiSOM[20] were a dramatic reduction in the amount of coherence messages required to maintain consistency[4, 15] and a reduction in the overhead of the basic DSM mechanisms[24]. Suddenly a far wider spectrum of programs could be usefully solved using DSM, even to the point where moderate-grained programs could be solved as quickly using a

This research was supported in part by the National Science Foundation under Grant CCR-9308879 and by the Advanced Research Projects Agency (ARPA), monitored by the Department of the Army, under Contract DABT63–94–C–0058. The opinions and conclusions contained in this article are those of the authors and should not be interpreted as representing official views or policies of the Advanced Projects Research Agency, the U.S. Government, or the University of Utah.

software DSM system as using dedicated shared memory hardware[8]. This fundamental improvement in DSM performance brought with it another rush of research efforts aimed at overcoming the new bottlenecks that were exposed and extending DSM into yet new areas[16, 12, 9, 10], but it is our opinion that despite the recent flurry of research published in the area, DSM is once again about to hit a lull. Although there are a number of ongoing efforts that we would judge to have the potential to make another fundamental leap in DSM performance and usability, recent results seem to be of the delta-improvement variety rather than the fundamental leaps needed to significantly increase the functionality and impact of DSM. Even worse, the net commercial impact of DSM remains limited to a small number of commercial operations[15, 19] and internal research projects at a number of workstation vendors (e.g., SGI and IBM). Although we know of a relatively large number of companies who have expressed an interest in how DSM might be used to solve their problems (names omitted to protect the guilty), we fear that without further progress DSM will remain an academic novelty, used primarily as a vehicle for acquiring PhD theses with little impact on the "real world." The question is, what should we, as researchers, do about this sad state of affairs?

2 Where We Should Be Heading

The problems as we see them with the state of the art in DSM systems include the following:

- Existing systems are directed almost exclusively at large scale homogeneous scientific applications, a niche too small to drive significant adoption of DSM technology into the mainstream.
- No DSM systems are freely available[1], so even the scientific programmers who might be able to exploit the power of current DSM systems are left using burdensome message passing systems like PVM[13].
- Existing systems are not well integrated with the rest of the software environment such as the compiler (with the notable but limited exceptions of Midway[24], SAM[21], and Blizzard[22]), and thus do not exploit the power of the existing software infrastructure.
- There is a dearth of tools for debugging and tuning the performance of DSM programs.
- The performance of existing systems for fine-grained programs is still relatively poor.

Note that we listed performance as the last of the problems! While we are sure that this is where the majority of the research effort will be spent, since it is the "sexiest" problem to attack and probably the only one that would be accepted for publication at SOSP or OSDI, we believe it is not the most important challenge that needs to be overcome. The state of DSM technology has progressed to the point where it could be used effectively to solve a wide variety of real problems if less "sexy" details, such as making DSM a fundamental system service that can be used by any cooperating applications (not just explicitly parallel programs) and adding DSM support to conventional compilers and debuggers, were addressed. In particular, the lack of a freely available DSM system, including source, has precluded the kind of acceptance by the research and user communities that PVM and similar systems have received.

Here is what we believe needs to be done:

- A quality DSM system that exploits some collection of the recent advances in DSM technology (multiple writer protocols, lazy release consistency, software write detection, etc.) must be released to the research and user communities.
- DSM needs to be made a basic system service so that applications other than homogeneous scientific programs can use it (e.g., to support sharing between distributed file buffer caches or nameservers)[2].
- Work must be done to add compiler support for DSM, as is being explored in SAM[21], Midway[24], and Blizzard[22].
- Tools to debug and tune the performance of DSM systems must be developed.

In the next section we will outline a modest effort in these directions.

3 Quarks

One of the research projects being performed under the umbrella of the Computer Systems Laboratory at Utah is the development of a distributed shared system named Quarks[3]. Quarks consists of a user-level

[1]With the exception of Mether[18], which is a first generation DSM system.

[2]Mach's XMM[2] interface is a step in the right direction, but it is based on dated technology.

[3]The name is derived from our hope that Quarks will be the basic building block on which more sophisticated DSM systems and applications are developed once it has been released into the research universe.

library and associated header files that support DSM on collections of workstations. Currently Quarks runs on SunOS 4.1, HP BSD 4.3, and HPUX; native Mach and IRIX 5.2 ports are in the works. Quarks includes a number of modern DSM features such as support for multiple consistency protocols within an application on a per page basis (e.g., a write invalidate protocol providing strict consistency, a delayed write update protocol providing release consistency, etc.). An effort has been made to make adding new protocols easy, allowing the research community to experiment with new protocols and compiler writers to develop specialized protocols (as in Blizzard[22]). Currently Quarks is aimed primarily at the traditional shared memory parallel programming niche, with built-in support for remote thread forking and synchronization, but we are working to redesign it to target a more general application mix (e.g., cooperating servers or programs that monitor the internal state of other programs).

In conjunction with the Quarks design effort, we have modified the `gcc` compiler to generate DSM-specific code, including embedded support for mixed DSM and RPC programs (as in Carlos[16]), support for static shared data, and (soon) support for software write detection and fine-grained access tracing, which we anticipate using for performance tuning and debugging. In an effort to increase Quarks' impact, we have strived to provide a simple user interface. Quarks supports C and C++ programs using the m4 macros to describe parallelism and synchronization, as in the SPLASH programs. In addition, Quarks uses a simple X-based user interface that supports limited parallel debugging using `gdb`. An alpha-release of a public domain version of Quarks is accessible via ftp to `jaguar.cs.utah.edu:pub/dsm`. Further information about and source code for Quarks can be obtained via the WWW at `http://www.cs.utah.edu/projects/flexmach/quarks.html`. It is our hope that by placing a relatively sophisticated DSM system in the public domain we will increase the number of users and make DSM research easier to perform.

On a separate note, one facet of the ongoing Flex/Mach project at Utah is the replacement of the XMM[2] implementation with a more modern DSM variant to make DSM a basic, efficient system service for Mach applications and servers. This effort will be integrated with Quarks to support DSM for applications other than parallel scientific programs on top of Mach.

4 Conclusions

The performance of software DSM systems has improved dramatically by addressing the problems of false sharing and excessive DSM-related communication [3, 6, 24]. Efficient DSM systems can now perform as well as hardware shared memory for moderate grained programs [6], but the overhead associated with software implementations of DSM limits its value for fine-grained computations. However, although performance remains a problem for fine-grained problems, the primary obstacles that block DSM's acceptance into the mainstream are primarily not performance related, but rather less sexy issues such as debugging and performance tuning tools and greater availability and portability. In this position paper we have outlined what we believe are the most important problems that need to be overcome before DSM will be widely accepted in the user community, and briefly described the Quarks DSM system, a modest step in this direction.

References

References

[1] H.E. Bal, M.F. Kaashoek, and A.S. Tanenbaum. Orca: A language for parallel programming of distributed systems. *IEEE Transactions on Software Engineering*, pages 190–205, March 1992.

[2] J.S. Barrera. A fast Mach network IPC implementation. In *Proceedings of the Mach USENIX Symposium*, pages 1–12, 1991.

[3] J.B. Carter, J.K. Bennett, and W. Zwaenepoel. Techniques for reducing consistency-related communication in distributed shared memory systems. *ACM Transactions on Computer Systems*. To appear.

[4] J.B. Carter, J.K. Bennett, and W. Zwaenepoel. Implementation and performance of Munin. In *Proceedings of the 13th ACM Symposium on Operating Systems Principles*, pages 152–164, October 1991.

[5] J.S. Chase, F.G. Amador, E.D. Lazowska, H.M. Levy, and R.J. Littlefield. The Amber system: Parallel programming on a network of multiprocessors. In *Proceedings of the 12th ACM Symposium on Operating Systems Principles*, pages 147–158, December 1989.

[6] A.L. Cox, S. Dwarkadas, P. Keleher, H. Lu, R. Rajamony, and W. Zwaenepoel. Software versus hardware shared-memory implementation: A case study. In *Proceedings of the 21st Annual International Symposium on Computer Architecture*, pages 106–117, May 1994.

[7] P. Dasgupta, R.C. Chen, S. Menon, M. Pearson, R. Ananthanarayanan, U. Ramachandran, M. Ahamad, R. LeBlanc Jr., W. Applebe, J.M. Bernabeu-Auban, P.W. Hutto, M.Y.A. Khalidi, and C.J.

Wileknloh. The design and implementation of the Clouds distributed operating system. *Computing Systems Journal*, 3, Winter 1990.

[8] S. Dwarkadas, P. Keleher, A.L. Cox, and W. Zwaenepoel. Evaluation of release consistent software distributed shared memory on emerging network technology. In *Proceedings of the 20th Annual International Symposium on Computer Architecture*, pages 144–155, June 1993.

[9] M.J. Feeley, J.S. Chase, V.R. Narasaya, and H.M. Levy. Integrating coherency and recovery in distributed systems. In *Proceedings of the First Symposium on Operating System Design and Implementation*, pages 215–229, November 1994.

[10] P. Ferreira and M. Shapiro. Garbage collection and dsm consistency. In *Proceedings of the First Symposium on Operating System Design and Implementation*, pages 229–241, November 1994.

[11] B. Fleisch and G. Popek. Mirage: A coherent distributed shared memory design. In *Proceedings of the 12th ACM Symposium on Operating Systems Principles*, pages 211–223, December 1989.

[12] V.W. Freeh, D.K. Lowenthal, and G.R. Andrews. Distributed filaments: Efficient fine-grained parallelism on a cluster of workstations. In *Proceedings of the First Symposium on Operating System Design and Implementation*, page 201=214, November 1994.

[13] G.A. Geist and V.S. Sunderam. The PVM system: Supercomputer level concurrent computation on a heterogenous network of workstations. In *Sixth Annual Distributed-Memory Computer Conference*, pages 258–261, 1991.

[14] E. Jul, H. Levy, N. Hutchinson, and A. Black. Fine-grained mobility in the Emerald system. *ACM Transactions on Computer Systems*, 6(1):109–133, February 1988.

[15] P. Keleher, S. Dwarkadas, A. Cox, and W. Zwaenepoel. Treadmarks: Distributed shared memory on standard workstations and operating systems. In *Proceedings of the 1994 Winter Usenix Conference*, pages 115–131, January 1994.

[16] P.T. Koch, R.J. Fowler, and E. Jul. Message-driven relaxed consistency in a software distributed shared memory system. In *Proceedings of the First Symposium on Operating System Design and Implementation*, pages 75–86, November 1994.

[17] K. Li and P. Hudak. Memory coherence in shared virtual memory systems. *ACM Transactions on Computer Systems*, 7(4):321–359, November 1989.

[18] R.G. Minnich and D.J. Farber. The Mether system: A distributed shared memory for SunOS 4.0. In *Proceedings of the Summer 1989 USENIX Conference*, pages 51–60, June 1989.

[19] Myrias Corporation. Pams: A distributed shared memory system for unix workstations, 1994.

[20] N. Neves, M. Castro, and P. Guedes. A checkpoint protocol for an entry consistent shared memory system. In *Proceedings of the 13th Annual ACM Symposium on Principles of Distributed Computing*, August 1994.

[21] D.J. Scales and M.S. Lam. The design and evaluation of a shared object system for distributed memory machines. In *Proceedings of the First Symposium on Operating System Design and Implementation*, pages 101–114, November 1994.

[22] I. Schoinas, B. Falsafi, A.R. Lebeck, S.K. Reinhardt, J.R. Larus, and D.A. Wood. Fine-grain access control for distributed shared memory. In *Proceedings of the 6th Symposium on Architectural Support for Programming Languages and Operating Systems*, pages 297–306, October 1994.

[23] M. Young, A. Tevanian, R. Rashid, D. Golub, J. Eppinger, J. Chew, W. Bolosky, D. Black, and R. Baron. The duality of memory and communication in the implementation of a multiprocessor operating system. In *Proceedings of the 11th ACM Symposium on Operating Systems Principles*, pages 63–76, October 1987.

[24] M.J. Zekauskas, W.A. Sawdon, and B.N. Bershad. Software write detection for distributed shared memory. In *Proceedings of the First Symposium on Operating System Design and Implementation*, pages 87–100, November 1994.