

# An Infrastructure for Portable and Efficient Software DSM

Sven Karlsson and Mats Brorsson  
Department of Information Technology,  
Lund University, P.O. Box 118,  
SE-221 00 LUND, Sweden  
phone: +46-46-222 7579  
Sven.Karlsson@it.lth.se

## 1. ABSTRACT

**In this paper, some of the weaknesses of the current software DSM systems are identified. Among these weaknesses are the lack of user friendly network management functions, limited performance on high performance networks, limited portability, and no standard APIs. OpenMP is proposed as a standard API for future software DSM systems and the network management subsystem and network layer of the Balder system is described as an example of an infrastructure of a future DSM system.**

### 1.1 Keywords

Software DSM systems, Network Protocols, Network Management

## 2. INTRODUCTION

Recent research on software DSM systems [4] has shown that these systems can provide decent performance even when compared to hardware based cc-NUMA machines. Furthermore, software DSM systems yield good price/performance as systems can be built from commodity parts such as network interfaces and workstations or PCs.

In this paper we describe the management and network infrastructure we have chosen for a new Software DSM system, called Balder, which is in development at Lund University, Sweden. The ideas and views expressed are based on years of experience conducting performance studies and development work on existing software DSM systems such as TreadMarks [1] and CVM [6].

There are many evidences that software DSM has the potential to provide a simple programming model for parallel computing on distributed memory systems without hardware support for shared memory. One of the main weaknesses of current software DSM efforts lies in that the

programming interface does not conform to any standard and it is our firm belief that there is a future for software DSM systems only if we can base them on standards such as OpenMP [11]. While this is probably a requirement for acceptance, an OpenMP compiler can also make more extensive analyses of the dataflow in the parallel regions than an ordinary compiler due to the OpenMP directives. The information obtained can be used to further tune the software DSM system.

Furthermore, in order to gain broad acceptance, a great deal of work must be done in the areas of stability, network performance, and network management. Current software DSM systems have not paid attention to these areas enough. As cheap Intel-based SMPs recently have become common, we also feel that future software DSM system should be targeted to SMP nodes.

In Section 3 we describe some of the problems of the current software DSM systems. We describe how the Balder system solves some of these problems in Section 4 and in Section 5 we end the report with some conclusions.

## 3. SOFTWARE DSM SYSTEMS

For several years the dominant state-of-the-art software DSM system for research has been TreadMarks. While decent performance can be achieved on a wide range of applications, TreadMarks has some limitations as described below. These limitations are shared by many other software DSM systems.

TreadMarks does not have any network management support. It is hard to retrieve status regarding a specific node in a running TreadMarks systems. Message passing systems such as PVM [3], LAM [9] etc. are in this respect much more advanced. Since TreadMarks run on loosely coupled systems where hardware or software failures are not uncommon, it is vital to be able to monitor the network. Secondly, in TreadMarks there are no distinct interfaces between the network handling code and the consistency protocol handling code. This means in practice that it is very cumbersome to implement support for new networks or even to optimize the consistency protocols since these two are moulded together. In the end this means that in order to optimize performance on a specific network one will have to rewrite a great deal of the consistency handling routines. This is not only a waste of time, but it is also error prone and it leads to code that is difficult to maintain. It should,



server. From this information the server can decide whether a process has aborted, a deadlock has occurred, a node has failed and so on. The slaves also start the application processes and provide additional services to the network layer like providing identities of the computation nodes in the network. The slaves are, however, not involved in the applications' communication. Finally the slaves can kill stray processes.

When the user submits a job to the system, the server first checks whether there are enough computation nodes for it. If so, a virtual network consisting of the allocated nodes is formed and the job is associated with it. The server daemon orders the slave daemons running on the nodes in the virtual network to run the application. After this, the server has mainly two tasks. It monitors the execution of the job and provides services to the slave daemons. When the application has started on one of the computation nodes and wants to utilize the high performance network, it needs to know the identities of the other nodes. The network layer sends a request for the identities to the slave daemon using a TCP/IP socket and the slave relays the request to the server.

Whenever the network layer needs to know a system specific parameter it asks the slave which might need to consult the server. Each job has its own virtual network and consequently the network layer can only obtain information regarding its job's virtual network. In practice the network layer will not send any requests to the slave during the actual execution of the job so this scheme will not affect performance. The slaves collect the stdout and stderr outputs from the application and relay them to the server which in turn merges them with the outputs of all nodes in the virtual network. Periodically the network layer reports its current state to the slave daemon. This information is used by the system to detect deadlocks in the communication on the high performance network. Deadlocks can occur, for instance, if one node in some way loses connection to the network. The network layer sends these reports relatively seldom and always tries to utilize the time when the node is idle in order not to degrade performance.

Finally, when the job is terminated, all nodes in the virtual network are released and can be used for subsequent jobs. During the job's execution it is possible for the user to monitor each of the computational nodes in the virtual network. It is also possible for the user to kill a running job and the server will also kill the job if it detects or suspects that one of the nodes in the associated virtual network has lost the connection to the network.

## 4.2 Networking support

### 4.2.1 Interface

One of the most important issues in software DSM system design is to provide an efficient interface to the network. We have designed a simple message-passing layer to support our system. The network layer relies on gather/scatter operations to ensure that copying of data is kept at a

minimum. Messages can be delivered in-order or out-of-order. The in-order mode is intended to be used for coherency messages and it is possible to queue messages from several nodes using the same receive queue when ordering among all nodes is needed. Each message can be assigned an integer tag which simplifies the protocol handling code when several messages from the same node are to be received. It is also possible to register callback functions which are run when a message with a specific tag is received. This makes it possible to implement primitives similar to active messages [2] when needed. A simple window based flow control algorithm is used to ensure reliable transfer of all sent data. Checksums are only used if the network hardware does not detect bit errors.

All send and receive operations operate on linked lists of descriptors identifying pieces of memory where the actual message data is located or is to be received, see Figure 3. The head of the lists contains information such as destination and tag regarding the message. The links consist of descriptors that hold information, i.e. addresses and sizes of memory blocks. The same type of links are used for transmission and reception of messages, i.e., the descriptors points to memory where either message data exist or will be put when the message is received. This scatter/gather approach eliminates, in most cases, the need for dedicated receive buffers and reduces the number of copy operations needed to transfer a message. It also makes it possible for the network layer to efficiently build large network messages from smaller messages by simply linking several lists together. It is possible through the API to instruct the layer when to actually send messages over the network. This makes it possible to transfer large amounts of data without increasing the latency of high priority messages like consistency and synchronization messages.

The message layer has been designed to be easily ported to different networks. The underlying network drivers can operate in user or kernel space. We are currently working on UDP, LAPI [8] and SCI [10] ports. Our experience so far is that the scatter/gather approach makes it simple to utilize the special capabilities of the various networks like memory put and get operations.

It should be noted that a standard message passing API such as MPI [7] can be built on top of the network layer or the network layer can be built on top of MPI. This makes it

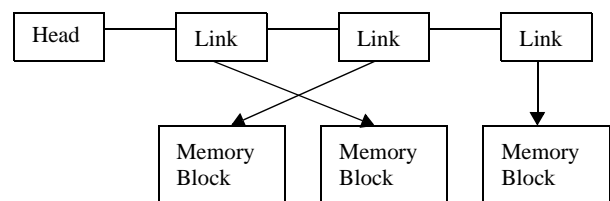


Figure 3. Linked list of descriptors.

possible to have both MPI and for instance OpenMP [11] coexisting in the same system providing a very interesting programming paradigm [12].

The API of the network layer largely consists of functions that manipulate, create and destroy the linked lists of descriptors. There are also functions that are used to register callbacks into the reception system, that sends or receive descriptor lists either synchronously or asynchronously, and that polls or wait for outstanding asynchronous receive operations. Finally there are functions that perform broadcasts among all nodes in a set of nodes.

## 5. CONCLUSIONS

We have in this paper described some of the problems that exist in current software DSM systems and how two of the sub-systems in the Balder software DSM system solves some of these problems. The network management system, which consists of daemons running on the nodes of the system, monitors the network and provides the user with status information and the possibility of controlling the network from a central point. The network layer provides a clean interface between the consistency protocol handling code and the network drivers. The layer has been designed to be portable and is currently being ported to three different networks. The network layer is also suitable for integration of message passing primitives into the software DSM system.

## 6. 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, *IEEE Computer*, Vol. 29, no. 2, pp. 18-28, February 1996.
- [2] T. von Eicken, D. E. Culler, S. C. Goldstein, K. E. Schauer, Active Messages: a Mechanism for Integrated Communication and Computation, in *Proceedings of the 19th International Symposium on Computer Architecture*. Gold Coast, Qld., Australia. May 1992. pp. 256-66.
- [3] G.A. Geist, V.S. Sunderam, Network-Based Concurrent Computing on the PVM System, *Concurrency: Practice and Experience*, 4 (4):293-311, June 1992.
- [4] D. Jiang et al., Application Scaling under Shared Virtual Memory on a Cluster of SMPs, in *Proceedings of the ISC'99*, June 1999. (to appear)
- [5] S. Karlsson and M. Brorsson, A Comparative Characterization of Communication Patterns in Applications using MPI and Shared Memory on an IBM SP2, in *Proceedings of 1998 Workshop on Communication, Architecture, and Applications for Network-based Parallel Computing*, Las Vegas, January 31 - February 1, 1998, pp. 189-201.
- [6] Pete Keleher, The Relative Importance of Concurrent Writers and Weak Consistency Models, in *Proceedings of the 16th International Conference on Distributed Computing Systems*, May 28, 1996.
- [7] Message Passing Interface Forum, MPI: A Message-Passing Interface Standard, version 1.1, June 12, 1995.
- [8] G. Shah, et al., Performance and Experience with LAPI - a New High-Performance Communication Library for the IBM RS/6000 SP, in *Proceedings of IPPS/SPDP'98*, 30 March-3 April 1998, Orlando, Florida.
- [9] Ohio Supercomputer Center, MPI Primer/Developing with LAM, Technical report, The Ohio State University, 1996
- [10] Knut Omang and Bodo Parady, Scalability of SCI Workstation Clusters, a Preliminary Study, in *Proceedings of the 11th IEEE International Parallel Processing Symposium (IPPS'97)*, Geneva, April 1997, pp. 750-755.
- [11] OpenMP Architecture Review Board, OpenMP C and C++ Application Program Interface, Version 1.0, October 1998.
- [12] A. Rodman and M. Brorsson, Programming Effort vs. Performance with a Hybrid Programming Model for Distributed Memory Parallel Architectures, in *Proceedings of EuroPar'99*, September 1999. (to appear)
- [13] R. Samanta, A. Bilas, L. Iftode, and J.P. Singh, Home-based SVM protocols for SMP clusters: Design and performance, in *Proceedings of the 4th IEEE Symposium on High-Performance Computer Architecture (HPCA-4)*, Las Vegas, Nevada, January 1998, pp. 113-124.
- [14] Daniel J. Scales, Kourosh Gharachorloo, and Chandramohan 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*, October, 1996.
- [15] E. Speight and J.K. Bennett, Brazos: A third generation DSM system, in *Proceedings of the 1997 USENIX Windows/NT Workshop*, August, 1997.
- [16] Robert Stets et al., CASHMERE-2L: Software Coherent Shared Memory on a Clustered Remote-Write Network, in *Proceedings of SOSP '97*, Saint Malo, France, October 1997