Designing High Performance DSM Systems using InfiniBand Features *

Ranjit Noronha and Dhabaleswar K. Panda

Dept. of Computer and Information Science
The Ohio State University
Columbus, OH 43210
{noronha,panda}@cis.ohio-state.edu

Abstract

Software DSM systems do not perform well because of the combined effects of increase in communication, slow networks and the large overhead associated with processing the coherence protocol. Modern interconnects like Myrinet, Quadrics and InfiniBand offer reliable, low latency (around 5.0 μs point-to-point), and high-bandwidth (upto 10.0 Gbps in 4X InfiniBand). These networks also support efficient memory- based communication primitives like RDMA-Read and RDMA-Write which allow remote reading and writing of data respectively without receiver intervention. These supports can be leveraged to reduce overhead in a software DSM system. In this paper we propose a new scheme NEWGENDSM with two components ARDMAR and DRAW for page fetching and diffing respectively. These components employ RDMA and atomic operations to efficiently implement the coherency scheme. The scheme NEWGENDSM evaluated on an 8-node InfiniBand cluster using SPLASH-2 and TreadMarks applications shows better parallel speedup and scalability.

Keywords: *DSM Systems, Cache coherency protocol, InfiniBand, System Area Networks*

1 Introduction

Clusters have been widely deployed for providing low-cost high performance computing for a wide-range of applications. There is a wide variety of high-end networking technologies available to connect the machines within a cluster. Technologies like Myrinet [10], Quadrics [4] and InfiniBand [2] offer point-to-point latency of the order of 5.0 μs for small messages and very high unidirectional bandwidth of the order of 10 Gigabits per second (with InfiniBand 4X) for large messages. In addition to the basic communication primitives, these networks offer a variety of services and operations. For example, Myrinet and Quadrics have a programmable network interface card. InfiniBand and Myrinet support hardware-based remote atomic operations [11]. All these networks also support Remote Data Memory Access (RDMA) operations. RDMA allows a process to read or write a location in the memory space of another process over the network. RDMA operations do not require involvement from the receiver, an important consideration when designing scalable software.

Though considerable research has focused on the development of Software Distributed Shared Memory Systems [23, 15, 9], SDSM systems such as TreadMarks [16, 7] and implementations of HLRC [14, 22] have not been found to be scalable. These SDSM system are communication intensive, and depend critically on performance of networking technologies like Fast Ethernet, Giganet [12] and the earlier generation of Myrinet [10]. Heavy-weight protocols like TCP could not keep up with the communication rate of SDSM's [20]. Modern networks are fast out-pacing the capacity of processors to keep them filled to capacity. These modern networks might not only impact the performance of SDSM systems but allow one to explore new previously inconceivable protocols for SDSM.

Communication in SDSM can largely be characterized by the client-server request-response model shown in Figure 1. This form of communication is inefficient in a modern day setup. The client cannot continue computation till the server responds. In addition, application computation at the server is halted while servicing the request. With increasing cluster size, the server may on an average process multiple requests from a large number of clients. This increases application processing time. Worse the delay at the client is not only communication latency, but also a non-negligible queuing time at the server. These overheads have a direct impact on the scalability of SDSM's. Is there an alternative to the request-response communication model, which can take advantage of

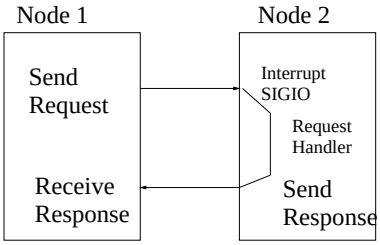

Figure 1. DSM client server communication model.

*This research is supported in part by Department of Energy's Grant #DE-FC02-01ER25506, and National Science Foundation's grants #CCR-0204429 and #CCR-0311542.

modern primitives in high performance networks ?

Modern day interconnects allow the user to read and write data elements from another process's user space using RDMA operations. Remote atomic operations allow one to maintain datums in a coherent state. These operations do not require receiver based intervention. They can be used to move protocol processing into the client. In this paper we take the first step towards removing such asynchronous protocol processing with InfiniBand mechanisms and studying the associated performance benefits. Page fetching and diff creation in SDSM's are traditionally performed using the asynchronous handler. A request message is sent to the manager or home node of the page. The home node then sends a response back to the requester with the page contents. For diff's, the modified page is compared with its clean copy called a twin, and a run-length encoding of the differences is sent to the server or home node. The server or home node then applies these diff's to the page. There is clearly unnecessary copying and overhead involved here. Can these overhead be reduced using network based primitives like RDMA and atomic operations ?

We have proposed a new scheme NEWGENDSM to replace traditional asynchronous model of protocol processing with a synchronous model. The two components of NEWGENDSM are ARDMAR and DRAW. ARDMAR uses the RDMA Read operation in InfiniBand to directly read the page from the memory space of the remote process. DRAW uses RDMA Write to directly apply diffs's to the home page at synchronization points. Coherency is maintained through the combination of a light weight thread and the InfiniBand remote atomic operation *compare and swap*. This new protocol is evaluated with both micro-benchmarks and application-level benchmarks. Application-level evaluation shows an improvement of up to 1.63 in running time on 8 nodes.

The rest of this paper is organized as follows. Section 2 describes the implementation of HLRC and its main features along with an overview of the networking interconnect InfiniBand. Section 3 presents design possibilities of HLRC with InfiniBand mechanisms. Section 4 explores the design issues and alternatives used while implementing the page fetch and diffing using RDMA Read and network level atomic operations. Section 5 evaluates the design using micro-benchmarks and various applications. Section 7 presents conclusions and future directions.

2 Background Information

In this section we discuss the basic concepts behind the SDSM package HLRC with an emphasis on its communication model primitives. We also take a look at the InfiniBand standard with a focus on the main communication operations provided by this interconnection technology.

2.1 Overview of HLRC

Since the development of the first sequentially consistency SDSM system IVY [17], there has been a large body of research into the issues with SDSM. Unfortunately, SDSM has not been found to be scalable, largely because of the effects of protocol and communication overhead. The lazy release consistency model was the next advance, which postponed coherence activities to synchronization points, reducing the amount of communication. The home based lazy release consistency protocol (HLRC) [14] improved upon LRC by assigning pages to homes, with a home node being updated with modifications at every synchronization point.

HLRC reduced not only the communication associated with non-home based protocols, but also the memory footprint. In HLRC every page and lock is assigned a home node. At every synchronization point, the diffs for a particular page are sent to the home node and the memory for the diffs are released. A home node can be assigned in a variety of ways; the default behavior is that the default home of the page assigns it to the node that first requests that page.

An implementation of HLRC [22] over the Virtual Interface Architecture (VIA) [5] (HLRC-VIA) was carried on GigaNet [1]. The implementation of HLRC-VIA was multithreaded. The application thread would compute while an associated signal handler would take care of coherence activity on a page fault or miss. A separate thread would listen for incoming requests from other remote processes such as page fetches and lock requests. HLRC-VIA makes use of the RDMA constructs provided by VIA. Request messages or messages for services are sent via RDMA Write with immediate data. This generates an asynchronous request at the receiver which then processes and responds to this request by either forwarding this request to some other nodes or itself satisfying the request through several RDMA Write operations. The requester meanwhile polls a particular location in memory (to which the remote server writes using RDMA Write) to see whether the request has completed. In this paper we use a version of HLRC-VIA modified to work over the InfiniBand fabric. The next section briefly discusses the InfiniBand architecture.

2.2 Overview of InfiniBand

The InfiniBand standard is a framework for a System Area Network for connecting processing and I/O nodes. It defines various communication and management functions that are necessary to operate the interconnection fabric. InfiniBand uses a switched, channel-based interconnection fabric, which allows for higher bandwidth, more reliability and better QoS support. Interface to the fabric is through a Host Channel Adapter (HCA) on the processing node and a Target Channel Adapter (TCA) on the I/O node. Seman-

tics of various operations are defined via InfiniBand Verbs. The Mellanox implementation of the InfiniBand Verbs API called VAPI [3] supports the basic send-receive model and the RDMA operations read and write. There is also support for atomic operations and multicast. More details on InfiniBand can be obtained from [2].

3 HLRC Design Possibilities with InfiniBand Mechanisms

Let us now examine the potential for integrating network based support into HLRC. HLRC duplicates activities either already provided or which could be done with less overhead by network level services in InfiniBand. Figure 2 shows some of the matches between InfiniBand level primitives and HLRC protocol activities. More specifically the following should be possible :

- Asynchronous handling could be eliminated through the combination of atomic operations and RDMA Read support. Page fetching operations could potentially benefit from this type of support.
- Diff propagation in HLRC uses RDMA Write with immediate data which requires activation of the asynchronous handler. Diff processing can be potentially eliminated by performing RDMA Read operations. In this design, whenever a particular portion of a page is needed, it can be fetched from the current owner by issuing an RDMA Read operation. The owner does not have to be interrupted to perform this operation.
- Write notice and Barrier notification propagate via RDMA Write. Since these go to all other nodes, hardware based multicast could provide an efficient basis for this operation. This could potentially reduce significantly the amount of traffic needed for synchronization in an SDSM system.
- Locking could be achieved through the use of remote atomic operations. This could potentially benefit applications which frequently use locks; as the need to frequently process lock requests at the manager node and the last owner is eliminated.
- Asynchronous request messages could potentially propagate via higher priority service levels achieving better response time.

In this paper we focus on the first 2 options; eliminating asynchronous handling through the combination of atomic operations and RDMA Read while executing a page fetch operation and diff propagation through RDMA Write. Since other features (such as reliable multicast and service levels) are not yet completely operational in current generation InfiniBand hardware, we plan on investigating other enhancements in the future.

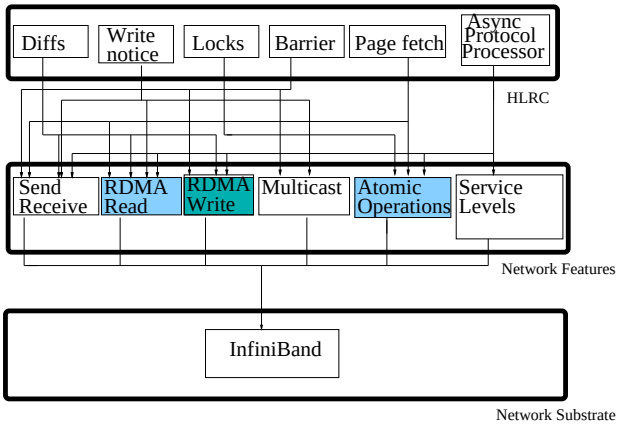

Figure 2. The SDSM primitives which could benefit from network support

4 Design of the NEWGENDSM protocol

In this section we discuss the design of our proposed protocol termed as NEWGENDSM. We start out by examining the existing protocol termed ASYNC. Following that is a description of the design of NEWGENDSM and then finally we examine some of the benefits that could accrue from NEWGENDSM.

4.1 Base ASYNC protocol

ASYNC employs the home based lazy release consistency protocol. In this protocol every page and lock is assigned a home by the protocol. All requests for accesses to a page or a lock go to the home node. Similarly all updates for a page and a lock go to the home node. In ASYNC updates or diffs for a page propagate to the home node at synchronization points such as a lock release or a barrier. We now examine page fetching and diffing in ASYNC.

4.1.1 Page fetching in ASYNC

First consider a page fetch operation. Figure 3 shows the protocol activity required by ASYNC for an example scenario. Initially all pages are assigned default home nodes. Let us assume that the default home for page two is node two. Now node one first requests page two from node two by sending it

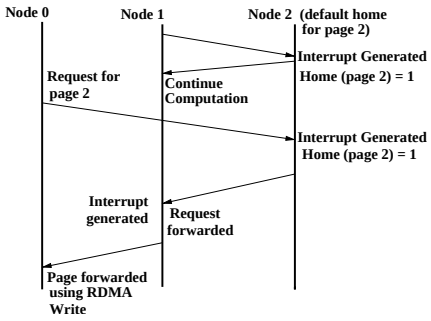

Figure 3. The original ASYNC protocol on a page fetch (for an example scenario)

a message (RDMA Write with immediate data) which is processed by the asynchronous protocol handler. The handler on node two appoints node one as the home node and updates its data structures. It then finishes servicing the request by sending a reply to node one (RDMA Write). Node one on receiving this reply updates its data structures and continues computing. Now when node zero requests this page for the first time by sending a request to the default home node one, node two forwards this request to node one, which in turn processes the request and replies to node zero with the correct version of the page. We define *page fetch time* or *page time* as the elapsed time between sending a page request and actually receiving the page.

4.1.2 Diffing in ASYNC

Now we go through the protocol steps when computing and applying diffs. As shown in Figure 4 node zero arrives at a synchronization point such as a barrier or a lock. At this point, node zero must propagate all updates it has made to all pages to the home node. Assume node zero has modified pages X and Y. It computes the diff which is a run-length encoded string of the differences between the original page and the modified page. Following that it sends these differences to the home node one through RDMA Write followed by a message containing the time-stamp. The home node then applies these diffs. It then sends an ACK back to node zero, which acts as a signal to node zero that the buffer on node one, which were used to receive the diffs to page X have been freed up and may be reused. Node zero now computes the diffs for page Y and sends them to node one and so on. Multiple buffers at the receiver may be used to speedup the process. We will now see how parts of this protocol can be enhanced with the features available in InfiniBand.

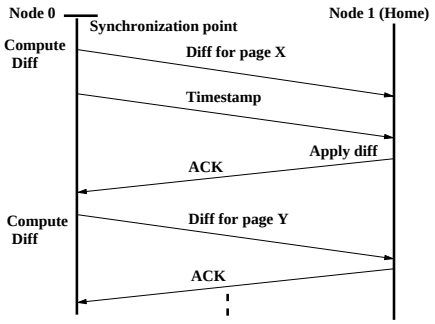

Figure 4. The original ASYNC protocol on a diff (for an example scenario)

4.2 NEWGENDSM protocol

In this section we discuss our proposed scheme NEWGENDSM. NEWGENDSM consists of two component; ARDMAR (Atomic and RDMA Read) and DRAW (Diff with RDMA Write). ARDMAR uses the InfiniBand operations atomic operation Compare and Swap and RDMA Read for page fetching and synchronization. DRAW uses RDMA Write for diffing. First we describe ARDMAR followed by DRAW.

4.2.1 ARDMAR

In this section we describe the design of ARDMAR which is shown in Figure 5 for an example scenario. The Atomic operation compare and swap have been combined with the RDMA Read operation to completely eliminate the asynchronous protocol processing. Let us assume the same pattern of requests for a page as shown in Figure 3. Here assume that node one wants to access page two for the first time. Let $home_n(x)$ denote the last known value for the home of page x at node n. Initial values for $home_n(x)$ are -1 at the default home node (indicating that a home has not been assigned) and x *mod* (number of nodes) at a non-home node. Initially $home_2(2) = -1$. Let us denote an issued atomic compare and swap operation as *CMPSWAP(node, address, compare with value, swap with value)* where address points to some location in node. Node 1 issues an atomic compare and swap CMPSWAP(2,$home_2(2)$,-1,1). The compare succeeds and now $home_2(2) = 1$. Now on completion of the atomic operation, node 1 knows that it is the home node ($home_2(2) = 1$) and can continue computation after appropriately setting the appropriate memory protections on the page.

Now assume that node zero wants to access page two for the first time. It also issues an atomic compare and swap operation CMPSWAP(2,$home_2(2)$,-1,0) which fails since $home_2(2) = 1$. Simultaneous with the atomic compare and swap, node zero also issues an RDMA Read to read in $home_2(2)$. The atomic compare and swap having failed node zero looks at the location read in by the RDMA Read. This location tells node zero that the actual home is now node one. Node zero now issues two simultaneous RDMA Reads. The first RDMA Read brings in the version of the page, while the second RDMA Read brings in the actual page. If the version does not match, both RDMA's are reissued until the correct version is obtained.

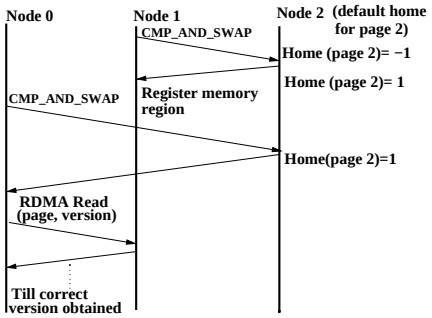

Figure 5. The proposed protocol ARDMAR (for the example scenario)

4.2.2 DRAW

Let us look at the design of DRAW. Figure 6 shows the protocol activity. DRAW uses RDMA Write to directly write the diffs to the page on the destination node. Again consider the diff creation and application activity shown in Figure 4. DRAW moves most of the diffing activity into node zero as shown in Figure 6. Assume that node zero has now initiated computing the diff at the synchronization point. Let *modified(X,n)* refer to the n'th position in page X. Let *clean(X,n)* refer to the n'th position in the twin of X where twin refers to a clean copy of page X. Let *buffer(t,n)* refer to the n'th position in communication buffer t. Let *RWRITE(source,dest,t,s,len)* denote an RDMA Write descriptor initiated at node source bound for node dest, using buffer t, starting at location s and of length len. Let us assume further that *modified(X,i..j)* differ from *clean(X,i..j)*. In this case, DRAW copies *modified(X,i..j)* into *buffer(t,i..j)* and creates *RWRITE(0,1,t,i,j-i)*. Similarly, assume that *modified(Y,b..f)* differ from *clean(y,b..f)*. DRAW copies *modified(Y,b..f)* into *buffer(t+1,b..f)* and creates *RWRITE(0,1,t+1,b,f-b)*. Now if pages X and Y are the only modified pages, at node zero DRAW issues *RWRITE(0,1,t,i,j-i)* and *RWRITE(0,1,t+1,b,f-b)*. In addition, a message containing the timestamps of pages X and Y is also sent. At node one, on receiving the messages containing the timestamps, DRAW updates the timestamps for pages X and Y.

4.3 Potential Benefits

NEWGENDSM could provide significant benefit to applications programmed with SDSM protocols both directly as well as indirectly. Applications cannot compute while the handler is being serviced. Removing the asynchronous handler allows more CPU resources to be allocated to the application. Asynchronous handling forces requests to be serialized, increasing response times. ARDMAR allows for requests to be serviced in parallel improving throughput. Page copying at the home node is also eliminated.

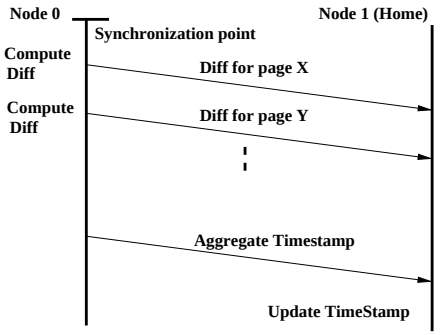

Figure 6. The proposed protocol DRAW (for the example scenario)

The benefit from DRAW is two-fold. On the one hand diff application at the home node is eliminated. This significantly cuts down on asynchronous handler time at the home node. This is akin to the benefits from a 0-copy protocol. Also diffs for all the modified pages can be sent to the home node in a single operation. This allows us to proceed at a much faster rate, since we don't need to wait for the ACK from the destination node for every page diffed. Finally, we do not need diff receive buffers at the destination node, reducing memory consumption.

5 Performance Evaluation

This section evaluates the performance of the ARDMAR, DRAW and NEWGENDSM with respect to ASYNC. Evaluation is in terms of overall execution time, page fetch time, lock time, and barrier times (which includes diff time) discussed in the following sections. First we describe the hardware setup. Following that the implementation is evaluated in terms of various micro-benchmarks. The application level evaluation is presented. Following that the effect of ARDMAR on page fetch time and asynchronous protocol processing are studied.

5.1 Experimental Test Bed

The experiments were run on an 8 nodes cluster connected through an InfiniScale MT43132 Eight 4x Port InfiniBand Switch. The HCAs are Mellanox InfiniHost MT23108 DualPort 4X HCA's. Each of the machines is a SuperMicro SUPER P4DL6 having a dual Pentium Xeon 2.4 GHz processors with 512 MB of main memory and a 133 MHz PCI-X bus. The SMP version of Linux 2.4.7-10 is the kernel running on each of these machines.

5.2 Micro-benchmark level evaluation

ARDMAR and *ASYNC* are both evaluated using the page fetch micro-benchmark modified from the original version implemented for the TreadMarks SDSM package [16]. *Page fetch time* is the elapsed interval between sending a request for a page and actually getting the page. It is measured as follows. The first node (master node) initially touches each of 1024 pages so that the home node is assigned to it. Following that each of the remaining nodes reads one word from each of the 1024 pages. This results in all the 1024 pages being read from the first node. As the number of nodes increases the contention for a page at the master node increases. The time of the second phase is measured. Figure 7 shows these results. *ASYNC* performs slightly better than *ARDMAR* at two and four nodes. *ARDMAR* performs better then *ASYNC* at eight nodes. *DRAW* and *ASYNC* are evaluated using a modified diff micro-benchmark also from the the TreadMarks SDSM package. The modifications take into account the home based nature of the SDSM protocol. Diff time can be measured in terms of two components, namely *Diff creation time* and *Diff application time*. *Diff creation time* encompasses the time needed to compare

a dirty page to its clean twin and create a run-length encoding of the differences, plus the time to post a descriptor to send these differences. *Diff application time* is the time spent in the asynchronous handler at the receiver to apply the differences to the page.

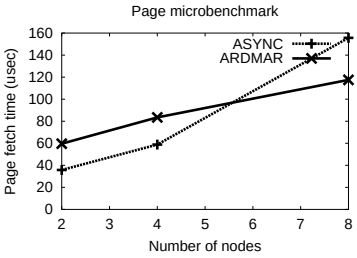

Figure 7. Performance results for the Page micro-benchmark. Time required to fetch a page from a given node when all the remaining nodes are contending for it.

Two nodes are used in the evaluation. Node zero initially touches 1024 pages so that the home is assigned to it. Following that node one either changes a single byte in each of the pages (*small*) or all the bytes in every page (*large*). Finally a barrier is executed. The time to compute the diff at node one is measured (*Compute* time). At node zero, time spent in the asynchronous handler is measured (*Apply* time). Figure 8 shows these results. *DRAW* consistently outperforms *ASYNC* in terms of both *Compute* time as well as *Apply* time. When *DRAW* is replaced by *ASYNC*, *Compute* time decreased by 2.47 and 2.29 in the case of small and large diffs respectively. *Apply* time decreased by 3.32 and 3.73 in the case of small and large diffs respectively.

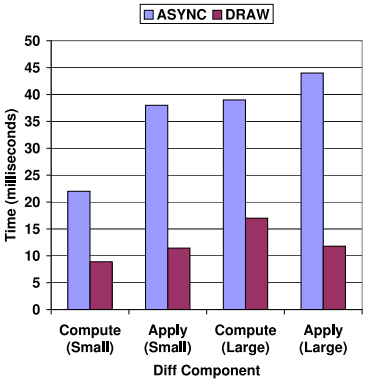

Figure 8. Performance results for the Diff micro-benchmark.

5.3 Application level evaluation

In this section we evaluate our implementation using four different applications; Barnes-HUT (Barnes), Traveling Salesman Problem (TSP), Radix sort (Radix), three dimensional FFT (3Dfft). Out of these applications Barnes and RS have been taken from the SPLASH-2 benchmark suite [25] while TSP and 3Dfft have been taken from the TreadMarks [16] SDSM package. The application sizes used are shown in Figure 9. All other parameters were kept the same as originally described in the [25, 16]. We will now discuss the performance numbers for the different applications.

Application	Parameter	Size
Barnes	Bodies	32678
TSP	Tour size	20 (large)
RS	num of keys	2621440
3Dfft	Grid size	128

Figure 9. Application sizes

5.3.1 Parallel Speedups

Figure 10 shows the parallel speedups for the different applications with eight nodes. *ARDMAR* shows better parallel speedup than *ASYNC* for Barnes and 3Dfft. *DRAW* exhibits better parallel speedup than *ASYNC* for all applications. *NEWGENDSM* shows better speedup for Barnes, 3Dfft and RS. RS with *NEWGENDSM* is 1.63 times faster than *ASYNC*.

5.3.2 Effect on execution time

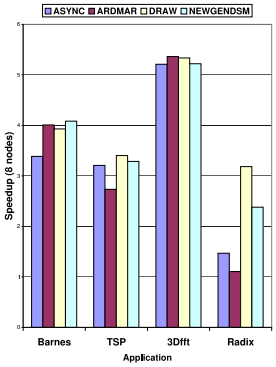

Figure 10. Application parallel speedup.

The breakdown in execution time for the different applications with eight nodes are shown in Figure 11. *Computation time* is the time spent in application processing. *Protocol time* is the time spent in executing the DSM coherency protocol. *Page time* is the time spent waiting for a page request to be serviced described in more detail in section 5.3.3. *Lock time* is the time spent in waiting for a lock. *Barrier wait time* is the time spent waiting for everybody else to arrive at the barrier. *Barrier compute time* is the time spent in creating diffs and sending write notices. Both of these are described in more detail in section 5.3.4. From this figure it be can seen that page time increases for *ARDMAR* in all applications compared to *ASYNC* except for Barnes. This is to be expected because of the synchronous nature of the protocol. Surprisingly, page fetch time and lock acquire time decreases for *DRAW* and *NEWGENDSM* in all cases. This is an indirect effect of the reduced load on the asynchronous protocol handler, discussed in the next section.

5.3.3 Effect on asynchronous protocol handling and page fetching time

Figure 12 shows the average time spent in the handler across all applications using *ASYNC*, *ARDMAR*, *DRAW* and *NEW-*

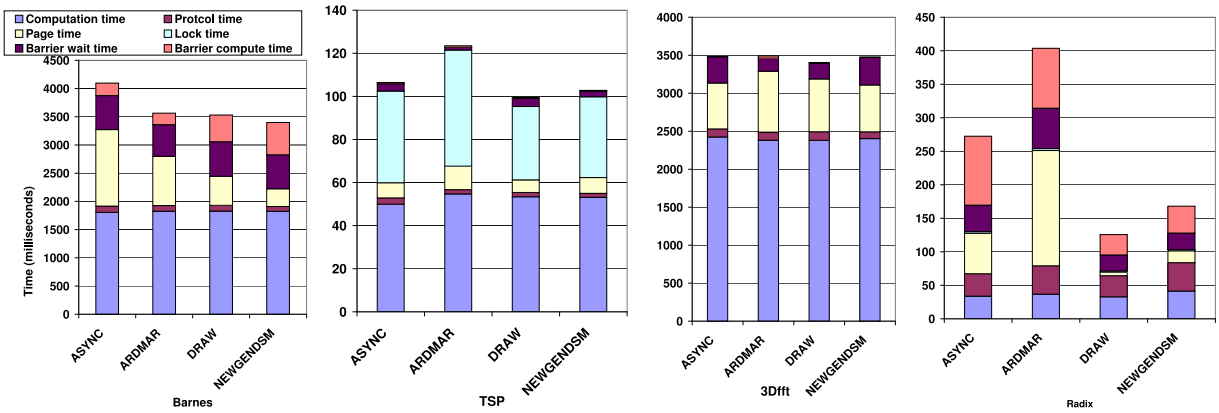

Figure 11. Breakdown of execution time for different applications.

GENDSM. In all cases asynchronous handler time using *ARDMAR*, *DRAW* and *NEWGENDSM* is significantly reduced. This shows that page and diff copying is a significant overhead in the asynchronous request handler. Eliminating these overheads frees up CPU time. It also allows faster turnaround for other asynchronous requests such as lock acquires. This reduces turn-around time and improves scalability.

5.3.4 Effect on diff time

Finally we discuss the effect on diff time. Diff time as described earlier has two components, namely *Diff Compute time* and *Diff apply time*. *Diff compute time* appears as a component of *Barrier compute time* shown in Figure 11. The barrier is split into three phases. The first phase computes and sends the diffs to the home node. The second phase sends the write notices to everybody and applies the write notices from everybody. The third phase consists of sending an acknowledgment to everybody. *Diff compute time* is more or less the same for *ASYNC*, *DRAW* and *ARDMAR* as shown in Figure 11. There is a correlation between the number of diffs applied and the time spent in the asynchronous handler. Barnes and 3Dfft create a large number of diffs. Correspondingly, handler time is substantially and reduce overhead. *NEWGENDSM* performs worse than *DRAW* in the case of TSP and Radix. This is because an aggregate time-stamp is sent for all pages after the diff's for those pages are sent to the home node. Pipelining can improve the performance of NEWGENDSM. This can be achieved by sending the timestamps for the page immediately after the diff. This will reduce the time by which the latest version of the page becomes available. Correspondingly this will reduce the wait time for processes waiting for a particular version of a page.

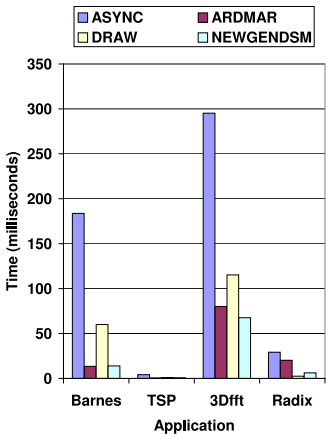

Figure 12. Asynchronous handler time.

6 Related Work

Ever since the proposal for the first SDSM system IVY [17] there has been considerable research conducted into the SDSM systems. However SDSM was not found to be scalable. The benefits of implementing HLRC over low level protocol like VIA was examined in [22]. Also implementing a SDSM package like Treadmarks directly over a low level protocol like VIA or GM over Myrinet, was shown to substantially reduces wait times and improves scalability in [8, 20]. Special network mechanisms were not employed in these implementations. A study of the effect of removing the interrupt handler in HLRC (GeNIMA) through the use of NIC support on Myrinet is discussed in [6]. Four different techniques, namely remote deposit for direct diff application, remote fetch for page fetching and Network Interface Locks (coherency information stored at the NIC) were implemented. Our work differs in that native atomic support provided by InfiniBand has been added to enhance the base protocol and the benefits have been studied in current generation clusters with InfiniBand. A comparison of benefits of using network based support as opposed to a migratory home protocol in the Cashmere SDSM system was studied in [23]. Four different techniques; Network Total Ordering, Broadcast and Remote-Write were studied. The interconnect used in this study was memory channel. Integrating network based get operations into the sequentially consistent DSZOOM SDSM over the SCI interface is dis-

cussed in [21]. Cache entries are locked using atomic fetch and set operations before being modified by remote put operations. We use a lazy release consistency (LRC) model rather than a sequentially consistent model. Network interface support was used to perform virtual memory mapped communication in addition to DMA based communication along with protected, low-latency user-level message passing in the SHRIMP project [18]. We have used RDMA rather than virtual memory mapped support, which does not involve reprogramming the NIC. A proposal for using active-memory support for SDSM systems to achieve software DSM with hardware DSM performance is discussed in [13] . [24] explores the effect of kernel level access to InfiniBand primitives on SDSM performance. Other research in SDSM has focused on changing the SDSM protocol rather than using network support. Reducing the effect of false sharing is discussed in [19].

7 Conclusions and Future Work

In this work we have examined the impact of reducing the need for asynchronous protocol processing in a home based SDSM system. This was achieved through the deployment of network based support in the form of atomic operations; RDMA-Read and RDMA-Write available with modern interconnects like InfiniBand. These supports were deployed in a new protocol NEWGENDSM. NEWGENDSM with components ARDMAR and DRAW for page fetching and diffing respectively was evaluated using micro-benchmarks and applications. Micro-benchmark evaluation showed that with increasing system size and network load, the response time of RDMA-Read is better as compared to an asynchronous protocol processor. RDMA-Write helps realize a zero-copy protocol improving performance. Application level evaluation were also performed. An improvement of up to 1.63 in the execution time of the application was observed.

Significant improvements to the protocol can be still be made. Remapping read only areas of the kernel page data structures into application space would avoid the need for a system call. RDMA Read operations on InfiniBand could also help in reducing the effects of false-sharing and achieve finer granularity. This can be achieved by restarting the computation early during a page fetch, when only the needed portion of a page has arrived. Barrier could potentially benefit from integration with the native InfiniBand multicast support. Locking could potentially show better performance using atomic operations. We are currently exploring these issues.

References

[1] Giganet. *www.giganet.com*.

[2] Infiniband Trade Association. *www.infinibandta.org*.

[3] Mellanox Technologies. *www.mellanox.com*.

[4] Quadrics Ltd. *www.quadrics.com*.

[5] Virtual Interface Architecture Specification. http://www.viarch.org.

[6] C. L. A. Bilas and J. Singh. Using Network Interface Support to Avoid Asynchronous Protocol Processing in Shared Virtual Memory Systems. *International Symposium on Computer Architecture*, May 1999.

[7] C. Amza, A. Cox, et al. Treadmarks: Shared Memory Computing on networks of workstations. *IEEE Computer, 29(2):18-28*, feb 1996.

[8] M. Banikazemi, J. Liu, . K. Panda, and P. Sadayappan. Implementing TreadMarks over VIA on Myrinet and Gigabit Ethernet: Challenges Design Experience and Performance Evaluation. *Int'l Conference on Parallel Processing*, sep 2001.

[9] J. Bjoerndalen, O. J. Anshus, B. Vinter, and T. Larsen. Comparing the performance of the pastset distributed memory system using TCP/IP and M-VIA. *The Second International Workshop on Software Distributed Shared Memory*, 1995.

[10] N. J. Boden, D. Cohen, et al. Myrinet: A Gigabit-per-Second Local Area Network. *IEEE Micro*, pages 29–35, Feb 1995.

[11] D. Buntinas, D. K. Panda, and W. Gropp. NIC-Based Atomic Operations on Myrinet/GM. *SAN-1 Workshop, held in conjuction with High Performance Computer Architecture (HPCA)*, 2002.

[12] H. Frazier and H. Johnson. Gigabit Ethernet: From 100 to 1000 Mbps.

[13] M. Heinrich and E. Speight. Providing Hardware DSM Performance at Software DSM Cost. *Technical Report No. CSL-TR-2000-1008, Cornell University, Ithaca, NY*, November 2000.

[14] L. Iftode. Home Based Shared Virtual Memory. *PhD Thesis, Technical Report TR-583-98, Princeton University*, 1998.

[15] A. Itzkovitz, A. Schuster, and Y. Talmor. Harnessing the power of fast low-latency networks for software dsms. *The First Workshop in Software Distributed Shared Memory*, 1999.

[16] P. Keleher, A. L. Cox, S. Dwarkadas, and W. Zwaenepoel. TreadMarks: Distributed Shared Memory on Standard Workstations and Operating Systems. In *Proceedings of the 1994 Winter Usenix Conference*, Jan. 1994.

[17] K. Li. IVY: A Shared Virtual Memory System for Parallel Computing. In *Proceedings of the International Conference on Parallel Processing*, pages 94–101, Los Alamitos, CA, 1988.

[18] M.A. Blumrich, C. Dubnicki, E.W. Felten, Kai Li, M.R. Mesarina. Two Virutal Memory Mapped Network Interface Designs. *Proc. of the Hot Interconnects Symp.*, 1994.

[19] L. Monnerat and R. Bianchini. Efficiently Adapting to Sharing Patterns in Software DSMs. *High-Performance Computer Architecture (HPCA)*, Feburary 1997.

[20] R. Noronha and D. K. Panda. Implementing TreadMarks over GM on Myrinet: Challenges, Design Experience and Performance Evaluation. *Workshop on Communication Architecture for Clusters (CAC'03), held in conjuction with IPDPS '03*, April 2003.

[21] Z. Radovic and E. Hagerstern. Implementing Low Latency Distrbuted Software-Based Shared Memory. *Workshop on Memory Performance Issues, held in conjuction with ISCA '01*, feb 1996.

[22] M. Rangarajan and L. Iftode. Software Distributed Shared Memory over Virtual Interface Architectur: Implementation and Performance. *Proc. of the Annual Linux Showcase, Extreme Linux Workshop, Atlanta*, October 2000.

[23] R. Stets, S. Dwarkadas, L. Kontothanassis, U. Rencuzogullari, and M. L. Scott. The Effect of Network Total Order, Broadcast, and Remote-Write Capability on Network-Based Shared Memory Computing. In *International Symposium on High-Performance Computer Architecture*, 2000.

[24] T. Birk, L. Liss and A. Schuster. Efficient Exploitation of Kernel Access to InfiniBand: a Software DSM Example. *Hot Interconnects*, August 2003.

[25] S. C. Woo, M. Ohara, E. Torrie, J. P. Singh, and A. Gupta. The SPLASH-2 Programs: Characterization and Methodological Considerations. In *International Symposium on Computer Architecture*, pages 24–36, 1995.