Designing High Performance DSM Systems using InfiniBand: Opportunities, Challenges and Experiences *

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 based DSM systems have traditionally not performed well because of the combined effects of increase in communication, slow networks and the large overhead associated with processing the coherence protocol. Recent modern interconnects like Myrinet, Quadrics and InfiniBand offer reliability, low latency (around 5.0 µs point-to-point), and high-bandwidth (up to 10.0 Gbps in 4X InfiniBand). Besides the traditional channel-based send/receive communication primitives, these networks also support memory-based communication primitives like RDMA Read and RDMA Write allowing remote reading and writing of data respectively without receiver intervention. InfiniBand also provides hardware support for remote atomic operations (fetch and add). These architectural supports open up new challenges in exploring low-overhead, high performance, and scalable cache coherency protocols for software DSM systems. In this paper, we take on such a challenge and develop an enhanced coherency protocol by taking advantage of RDMA Read and atomic fetch-and-add primitives. This enhanced protocol (termed as ARDMAR - Atomic and RDMA Read) helps to reduce the load at the default and actual home nodes on a page fetch request, leading to reduced page fetch time and handler processing time. The ARDMAR protocol is integrated into the HLRC DSM system and evaluated on a 16-node InfiniBand cluster for five different applications (Barnes, Radix Sort, Integer Short, 3D FFT, and Water) with various problem sizes. For 16-node systems and various problem sizes, the ARDMAR protocol is demonstrated to reduce the application-level execution time by a factor of up to 2.25. These results demonstrate potential to build high performance DSM systems using modern clusters with InfiniBand.

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

*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.

1 Introduction

Clusters are becoming increasingly popular for providing cost-effective high-performance computing for a wide range of applications. Networking technology is improving dramatically. Increasingly, a variety of advanced interconnects like Myrinet [10], Quadrics [4] and very recently InfiniBand [2] have become popular. These network offer very low 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 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.

Considerable research has focused on the development of Software Distributed Shared Memory (SDSM) Systems [24, 16, 9] in the past. SDSM systems such as TreadMarks [17, 7], SHRIMP [12] and HLRC [15, 23] provide an intuitive development environment as opposed to the Message Passing Interface (MPI) standard. These environments were developed earlier for clusters with slower interconnects (Fast Ethernet, Giganet [13] and the earlier generation of Myrinet [10]). These environments did not catch on earlier primarily because of the communication intensive nature of SDSM protocols which quickly choked networks and exhibited poor scaling properties. Modern networks have shifted the balance between processor speed and interconnection speed. These networks can potentially

sustain the communication rate of SDSM protocols, which has the potential to improve the scalability of SDSM protocols [21]. Thus, it makes an interesting challenge to examine the sources of overhead in the current SDSM protocol design when running over these new interconnects.

The request-response type of communication of the client-server model of SDSM could possibly be one of the main sources of overhead. With increasing system and application size, request serialization at the server drives up response time and affects scalability. InfiniBand offers RDMA and remote atomic operations. Thus, it is possible to replace the asynchronous protocol processing using these mechanisms. Another interesting question is how much performance benefit can be achieved by such a replacement om InfiniBand based clusters ? 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 operations in SDSM traditionally are 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 requestor with the page contents. We have proposed a new scheme to use RDMA Read operation in InfiniBand to directly read the page from the memory space of the remote process. Coherency is maintained through the InfiniBand remote atomic operation *compare and swap* which is used to assign a home node for the page. This new protocol is evaluated with both micro-benchmarks and application-level benchmarks. Micro-benchmark evaluation shows that RDMA operations dramatically reduce the wait time required for page fetching operations. Furthermore, application-level evaluation show an improvement of up to 2.25 in running time on 16 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 using RDMA Read and network level atomic operations. Section 5 evaluates the design using a micro-benchmark 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. We also compare the performance of InfiniBand with the networks Myrinet, Quadrics and Giganet using the metrics of latency and bandwidth.

2.1 Overview of HLRC

Since the development of the first sequentially consistency SDSM system IVY [18], 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) [15] improved upon LRC by assigning pages to homes, with a home node being updated with modifications at every synchronization point.

HLRC was designed with the goal of reducing not only the communication associated with non-home based protocols like TreadMarks, but also to reduce 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 in HLRC is that the default home of the page assigns it to the node that first requests that page.

An implementation of HLRC [23] over the Virtual Interface Architecture (VIA) [5] was carried on GigaNet [1]. The implementation of HLRC was multi-threaded. 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 over 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 requestor meanwhile polls a particular location in memory (to which the remote server writes using RDMA Write) to see whether the request has completed. The next section briefly discuss 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 Adaptor (HCA) on the processing node and a Target Channel Adapter (TCA) on the I/O node. Semantics 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].

2.3 Comparing InfiniBand Performance with Other Interconnects

We will now examine how InfiniBand performs compared to other networks. Latency and bandwidth are among the most basic metrics of performance comparision between different networks. Latency can be measured using the standard ping-pong test. The ping side posts 2 descriptors. One is a send descriptor and the other is a receive descriptor. It then polls for the completion of the receive request. The pong side posts a single receive descriptor, polls for its completetion and then posts a send descriptor. The whole process is repeated a large number of times to reduces timing error. Then the measured round-trip time is divided by two to obtain the one-way latency. Figure 1 shows the latency for the networks InfiniBand, Myrinet and Quadrics and compares it with the earlier generation networking technology Giganet.

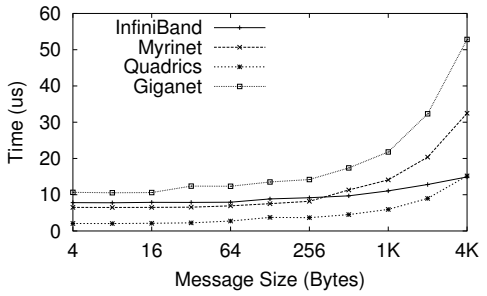

Figure 1. Latency for messages on different networks.

Unidirectional bandwidth of a network is the maximum data rate that can be sustained at the network level. To compute the bandwidth, messages are sent out repeatedly to the receiver node for a number of times and then the sender waits for the last message to be acknowledged. The time for sending these back-to-back messages is measured and the timer is stopped after receiving the acknowledgement for the last message. The number of messages sent is kept large so that the transmission time for the acknowledgement is small in comparision to the total time. Figures 2 shows the bandwidth for the networks InfiniBand, Myrinet and Quadrics and compares it with the earlier generation networking technology Giganet. InfiniBand has several times the bandwidth of other networks for large messages. This could impact the performance of SDSM protocols. Exploiting the resource rich nature of the InfiniBand network is one of the focuses of this paper.

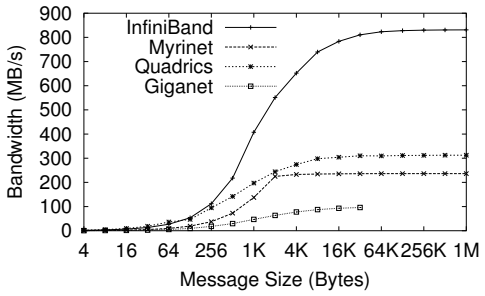

Figure 2. Bandwidth for messages on different networks[1]

3 HLRC Design Possibilities with InfiniBand Mechansims

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 3 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

[1] Due to experimental error, bandwidth numbers above 32K could not be obtained for Giganet and will be available in the final version of the paper

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 option; eliminating asynchronous handling through the combination of atomic operations and RDMA Read while executing a page fetch operation. 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. In traditional SDSM's co-

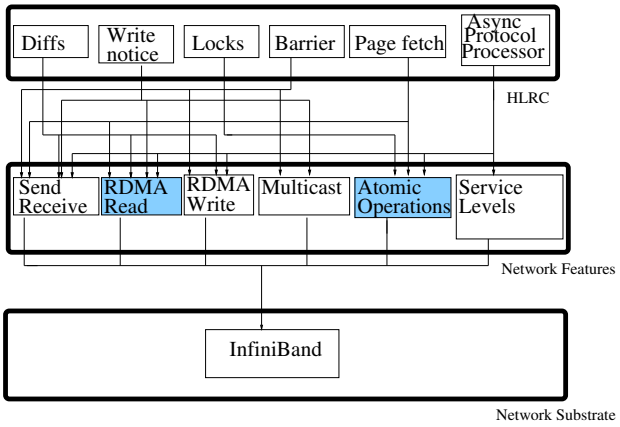

Figure 3. The SDSM primitives which could benefit from network support

herence is maintained through the natural ordering of messages imposed by the network and serialization of requests in the asynchronous protocol handler. A similar effect can be achieved by employing remote atomic operations in InfiniBand. Figure 4 shows how atomic and RDMA Read based operations could potentially benefit the basic HRLC protocol. The request-response model used on a page fetch can be replaced by the RDMA Read operation which directly reads a page from the process space of a remote node. Thus, these two architectural features, atomic and RDMA Read can be integrated into the HLRC base protocol. This can allow us to eliminate the asynchronous protocol processing needed for a page fetching operation. This enhancement also has the added benefits of a home based protocol along with stronger integration with the network to produce enhanced scalability.

4 Design of the ARDMAR protocol

In this section we discuss the design of our proposed protocol termed as ARDMAR - Atomic and RDMA Read. We start out by examining the existing HLRC protocol. Following that is a description of the design of ARDMAR and

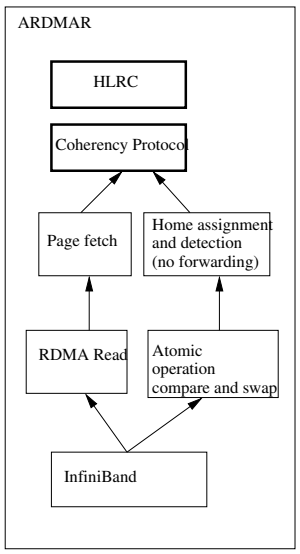

Figure 4. How RDMA Read and atomic operations could potentially enhance the base HLRC protocol

then finally we examine some of the benefits that could accrue from ARDMAR.

4.1 Base HLRC protocol

HLRC 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 HLRC updates or diffs for a page propagate to the home node at synchronization points such as a lock release or a barrier.

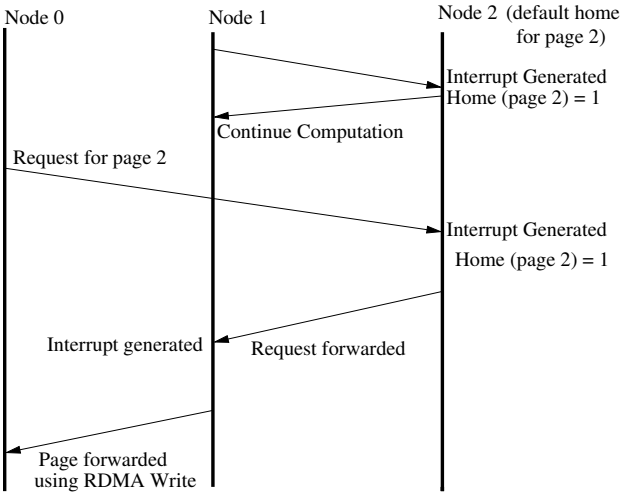

Figure 5. The original HLRC protocol (for an example scenario)

Figure 5 shows the protocol activity required by HLRC for an example scenario. Initially all pages are assigned a default home node. The default home for page 2 is node 2. Now node 1 first requests page 2 from node 2 by sending it a message (RDMA Write with immediate data) which is processed by the asynchronous protocol handler. The handler on node 2 appoints node 1 as the home node and updates its data structures. It then finishes servicing the request by sending a reply to node 1 (RDMA Write). Node 1 on receiving this reply updates its data structures and continues computing. Now when Node 0 requests this page for the first time by sending a request to the default home node 2, node 2 forwards this request to node 1, which in turn processes the request and replies to node 0 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. We will now see how parts of this protocol can be enhanced with the features available in InfiniBand.

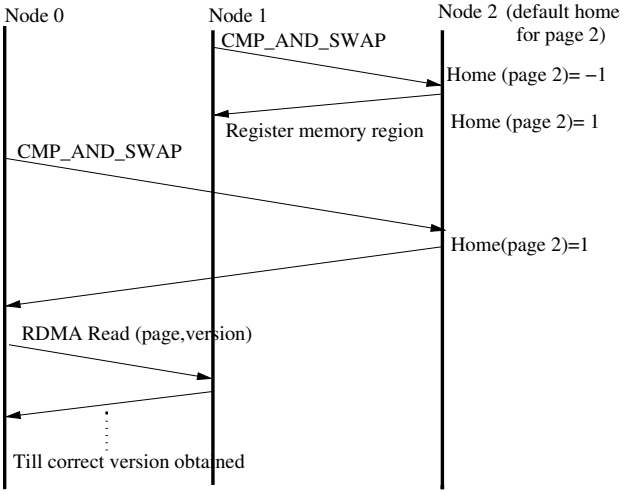

Figure 6. The proposed protocol ARDMAR (for the example scenario)

4.2 ARDMAR protocol

In this section we describe the design of the proposed protocol ARDMAR which is shown in Figure 6 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 5. Here assume that node 1 wants to access page 2 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 0 wants to access the page 2 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 0 also issues an RDMA Read to read in $home_2(2)$. The atomic compare and swap having failed node 0 looks at the location read in by the RDMA Read. This location tells node 0 that the actual home is now node 1. Node 0 now issues 2 simultaneous RDMA Reads. The first RDMA Read bring 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. This approach requires that initially all access permissions on all pages be initially set to read. This is different from the original implementation, where initially all pages which are at the default home are in read-write mode (exclusive). Therefore on a write to a page at the default node there is no page fault, but which results in a page fault for the new implementation. This could result in an increased number of page faults at the home node for applications with that particular write pattern and this effect will be discussed in section 5.3.3.

4.3 Potential Benefits

ARDMAR 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. As shown in Figure 7 asynchronous handling forces requests to be serialized, increasing response times. ARDMAR allows for requests to be serviced in parallel improving throughput. Further, there is the potential for the network to optimize multiple requests through caching behavior and even through optimizations like DMA chaining which allow multiple DMA transfers to be performed more quickly across the PCI bus.

5 Performance Evaluation

This section evaluates the performance of the proposed implementation with atomic and RDMA Read support *ARDMAR* against the original implementation of HLRC

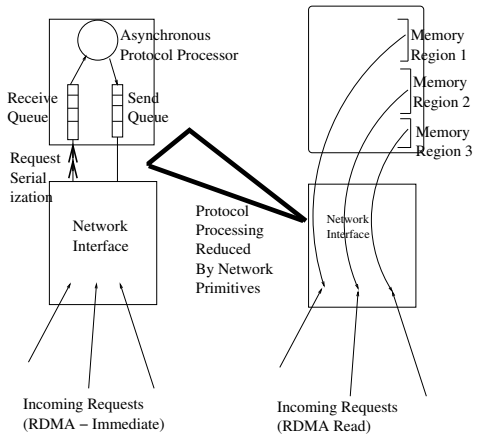

Figure 7. Bottlenecks with asynchronous protocol processing

with asynchronous protocol processing support *ASYNC*. 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. The implementation is evaluated both in terms of varying system size as well as application size.

5.1 Experimental Test Bed

The experiments were run on a 16 nodes cluster connected through a Topspin [26] 360 24 port 4x InfiniBand switch. The HCAs are Topspin InfiniBand 4x HCA's. Each of the machines is a Microway dual Pentium Xeon 2.4 GHz processors with 2 GB of main memory and a 133 MHz PCI-X bus on a Tyan 2721-533 motherboard. The SMP version of Linux 2.4.18-10 is the kernel running on each of these machines. The InfiniBand interconnect has a latency for small messages of about 6.5 μs and a bandwidth of about 6.3 GBps. Table 1 summaries the latencies of some of the operations pertinent to HLRC. The isolated latency (i.e. in the absence of any other network traffic) of a RDMA-Read for a message size of 4096 bytes which is also the pagesize used in HLRC and is the most common message size for RDMA-Read is 36.39 μs. The latency of a remote atomic *compare and swap* is 23 μs. This value is expected to reduce as support for atomic operations improves on InfiniBand. In the next section we look at some micro-benchmarks which were used in the evaluation.

[2]It is to be noted that atomic operations over InfiniBand adapters have just become operational. Thus they have not been optimized. These operations are expected to be optimized over the next several months and will bring additional benefit to the proposed framework

[3]These refer to the time required to post a descriptor to the send queue on the senders side

Operation	Mesg. size (bytes)	Latency (μs)
Remote atomic	8	23 [2]
Local atomic	8	20 [2]
RDMA Read	4096	36.39
RDMA Write	4096	9.5 [3]
RDMA Write	140	6.7 [3]
RDMA Write (imm)	140	8.69 [3]

Table 1. Latencies of some basic operations on InfiniBand

5.2 Micro-benchmark level evaluation

ARDMAR and *ASYNC* were both evaluated using the page fetch micro-benchmark modified from the original version implemented for the TreadMarks SDSM package [17]. In this micro-benchmark 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 8 shows these results. *ASYNC* performs slightly better than *ARDMAR* at 2 and 4 nodes. For *ASYNC* at 2 nodes, the breakdown of timing is as follows:

- 8.69 μs RDMA Write with immediate data (Time to post Request message)
- 9.5 μs RDMA Write for the page (Response message)
- 6.7 μs RDMA Write for the control message (Response Control Message)
- 8.44 μs Pre-posting receive descriptor (to guarantee that the next message will be received)

This adds up to a total of 33.33 μs for ASYNC. For *ARDMAR* the breakdown in timing at 2 nodes is :

- 23 μs Atomic compare and swap (Request message for home assignment)
- 36.39 μs RDMA Read for the page (Page fetch)

This totals up to 59.49 μs for *ARDMAR*. This time is expected to come down as the support for atomic operations on InfiniBand improves. As the number of nodes increases, the time required to fetch a page using *ASYNC* increases exponentially demonstrating that the overhead for the asynchronous protocol handler is considerable and has poor scaling properties. However, with the ARDMAR protocol as the number of nodes increases, the page-fetch time levels off and slightly decreases from 8 to 16 nodes. This demonstrates that RDMA-Read seems to be a more scalable alternative than an asynchronous protocol processing scheme. The next section discusses the application level evaluation.

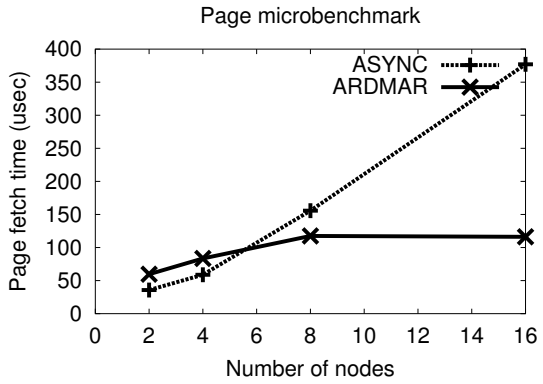

Figure 8. Performance results for the Page microbenchmark. Time required to fetch a page from a given node when all the remaining nodes are contending for it.

5.3 Application level evaluation

In this section we evaluate our implementation using five different applications; Barnes-HUT (Barnes), Radix sort (RS), Integer Sort (IS), three dimensional FFT (3DFFT) and Water spatial (Water). Out of these applications Barnes, RS, and Water have been taken from the SPLASH-2 benchmark suite [27] while IS and 3DFFT have been taken from the TreadMarks [17] SDSM package. The applications were evaluated both in terms of varying system size as well as application size. For system size, the applications were run on two, four, eight and 16 nodes respectively. For application size, different parameters within the applications were changed. These sizes are listed in Table 2. All other parameters were kept the same as originally described in the [27, 17]. We will now discuss the performance numbers for the different applications. Following that is a discussion of the effects of using RDMA and atomic operations on asynchronous protocol handling and page fetching time. Finally, we will look at how ARDMAR alters the the distribution of page faults in the applications.

Application	Parameter	Large size	Small size
Barnes	Bodies	32678	16384
RS	num of keys	2621440	52488
IS	$2^{num.of keys}$	23	21
3DFFT	Grid size	128	64
Water	num of molecules	8192	4096

Table 2. Application sizes

5.3.1 Effect on execution time

The execution time for the different applications are shown in Figure 10. The breakdown of execution time for the larger size of different applications on 16 nodes is shown in Figure 11. From this figure it can seen that page time

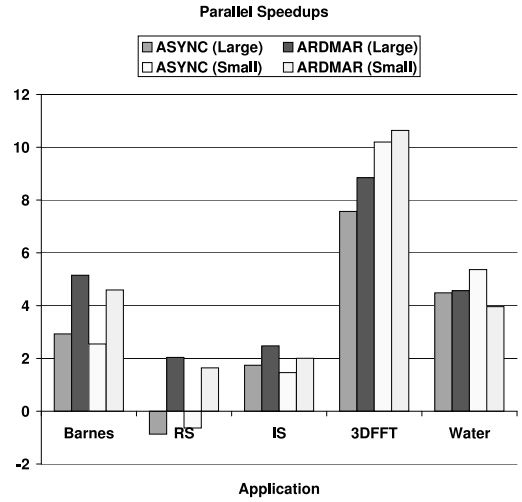

Figure 9. Parallel speedups on 16 nodes for different application sizes.

decreases for *ARDMAR* in the case of Barnes, RS, IS and 3DFFT as compared to *ASYNC*.

Figure 9 shows the parallel speedups for the different appplications at 16 nodes. *ARDMAR* shows better parallel speedup than *ASYNC* in all cases and in fact shows a speedup for RS while *ASYNC* shows a slowdown. From Figure 10 we can observe that for Barnes with application size 32678, ARDMAR is 1.73 times faster than ASYNC on 16 nodes. Also ARDMAR shows good scaling property as compared to ASYNC. Again for application size 16384, ARDMAR is 1.9 times faster than ASYNC and shows good scaling properties compared to ASYNC. For both application sizes, the execution time for ASYNC increases with increase in system size while it decreases for ARDMAR. As we see in the following subsections, the asynchronous protocol handling time increases substantially for ASYNC, while it remains constant for ARDMAR. Page time also increases dramatically for ASYNC while increasing at a much slower rate for ARDMAR.

Again referring to Figure 10, we see that RS shows good scalability for the larger application size 2621440. ARDMAR at 16 nodes is 2.25 times faster than ASYNC. Also for both application sizes it can be seen clearly that ARDMAR scales better than ASYNC. The time spent for page fetching decreases with increase in system size which explains improvement in performance for RS.

We will now discuss the results for IS, 3DFFT and Water. From Figure 10 it can be seen that for IS at size 23, *ARDMAR* is 1.41 times faster than *ASYNC* on 16 nodes. For 3DFFT size 128, *ARDMAR* is 1.18 times faster than *ASYNC* on 16 nodes, while for Water for size 8192 *ARDMAR* is 1.02 times faster than *ASYNC* on 16 nodes. For Water *ARDMAR* does not perform as well as *ASYNC* for the smaller application size 4096 because of an increase in the number of page faults. This effect will be explored further in section

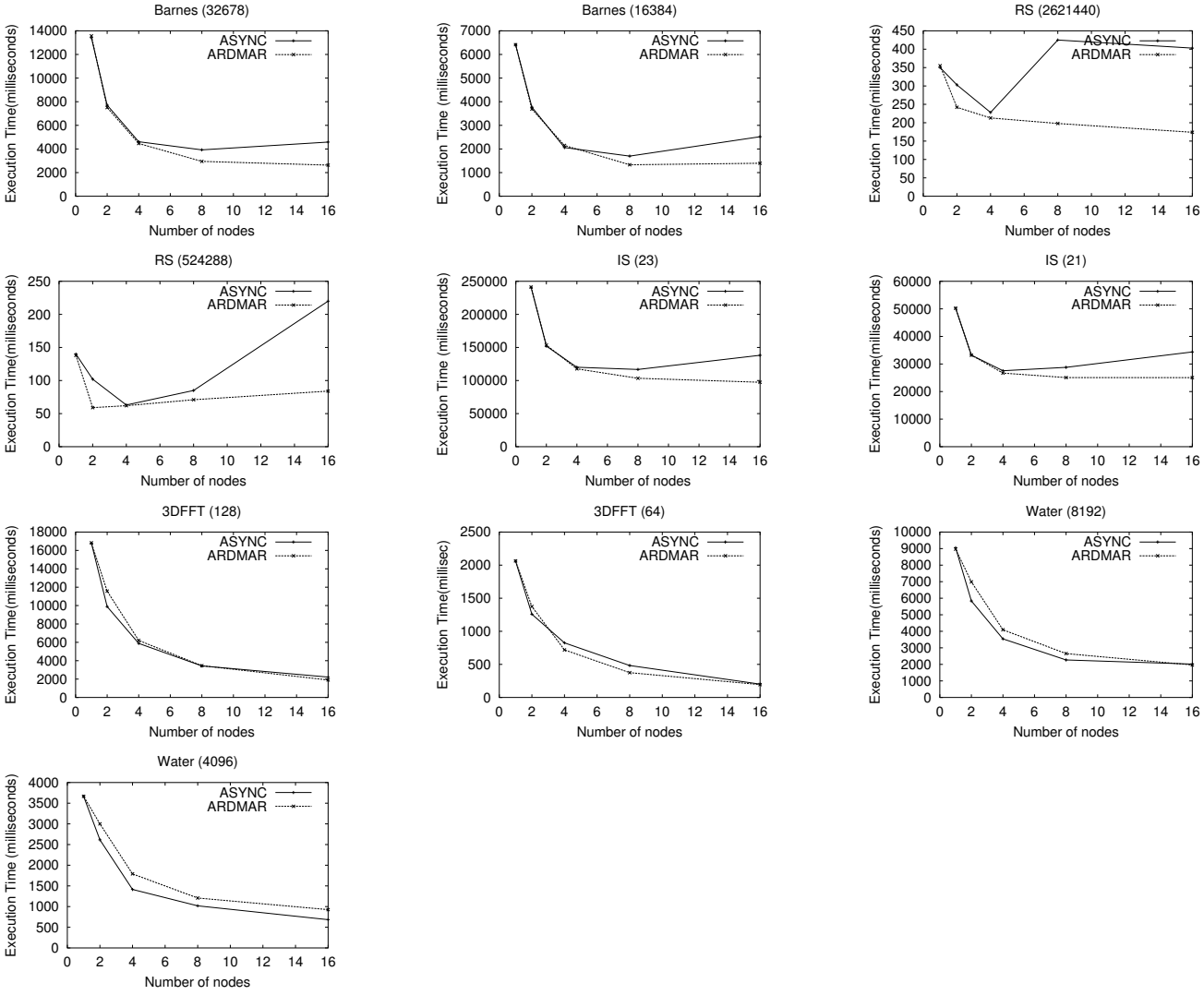

Figure 10. Overall execution times of the applications for the two protocols (ASYNC and ARDMAR) with varying system and application sizes.

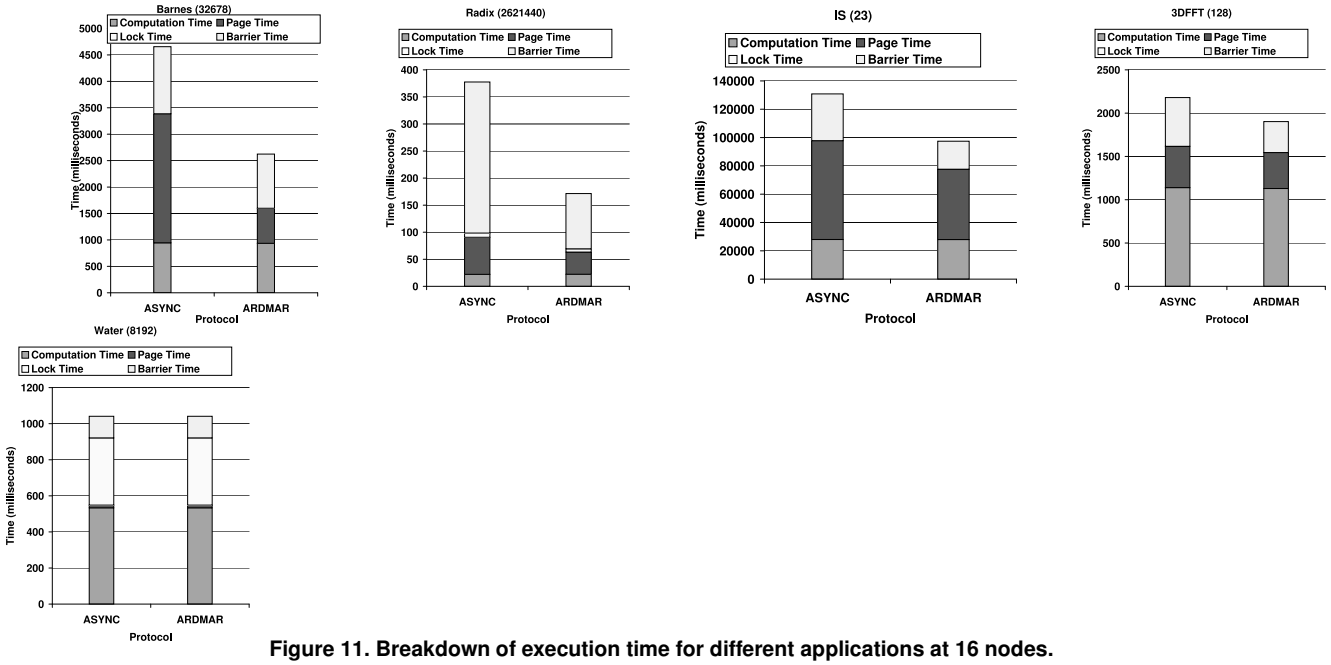

Figure 11. Breakdown of execution time for different applications at 16 nodes.

5.3.3. For all three applications with larger problem size, *ARDMAR* shows better scaling property as the system size is increased for the larger application size. In the next section we will discuss the effect of *ARDMAR* on page fetch time and handler time.

5.3.2 Effect on asynchronous protocol handling and page fetching time

RDMA-Read considerably reduces asynchronous protocol processing and page time. Figure 12 shows the average time spent in the handler across all applications using *ASYNC* as compared to *ARDMAR*. For 3DFFT and Water there is virtually no overhead for the asynchronous handler in *ARDMAR*, while there is considerable asynchronous protocol handling overhead for *ASYNC*. For the remaining applications, the time spent in the handler is mainly because of lock requests and requests for applying diffs at synchronization points. Asynchronous protocol processing time in RS is more dependent on the extensive locking distribution which accounts for the variance in the time. As discussed in Section 3 and shown in Figure 3, it is possible to completely eliminate the asynchronous protocol handler through the use of atomic support from the network and is the focus of our future work.

Figure 13 shows the average time spent across all nodes for a page fetch. For virtually all applications it can be clearly seen that with increase in system size, the time spent in page fetching decreases for *ARDMAR* and ultimately becomes less than *ASYNC*. In fact for the applications Water, it can be seen that the page fetching time increases with *ASYNC* but decreases with *ARDMAR*. These results can be explained using results from the page micro-benchmark explained in Section 5.2 which showed that with increasing contention and increasing number of nodes, RDMA-Read performs better as compared to the asynchronous protocol processing mode of operation. Barnes, RS and IS experience an increasing number of page misses with increasing system size, which explains why page time increases for both *ASYNC* and *ARDMAR*. While for 3DFFT the number of page misses decreases with system size which explains why the page fetching time decreases with increasing system size.

5.3.3 Effect on page fault distributon

Figure 14 shows the average number of page faults per node for each of the larger application sizes. It can be seen that the number of page faults is a considerable overhead for both *ASYNC* and *ARDMAR* and the page fault overhead is higher for *ARDMAR* for the applications 3DFFT and Water. This is a facet of the implementation, which requires the permissions on all page to be set to read-only as opposed to read-write-exclusive in *ASYNC*. Thus, if an application at the home node frequently writes to different pages in its process space for the first time, this will result in an increase in the number of page faults (which we seen in the case of 3DFFT and Water). The time required to invoke the page fault handler and then restart the computation is about 6.6 μs. It is possible to reduce the effect of and need for page-fault handling by invoking kernel level primitives at synchronization points. These primitives can avoid expensive system calls by page remapping to indicate which pages have been written. This is one of the problems we are looking into and the results will be included in the final version.

6 Related Work

Ever since the proposal for the first SDSM system IVY [18] 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 [23]. 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, 21]. 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 [24]. 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 discussed in [22]. 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 [19]. 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 [14] . [25] explores the effect of kernel level access to InfiniBand primi-

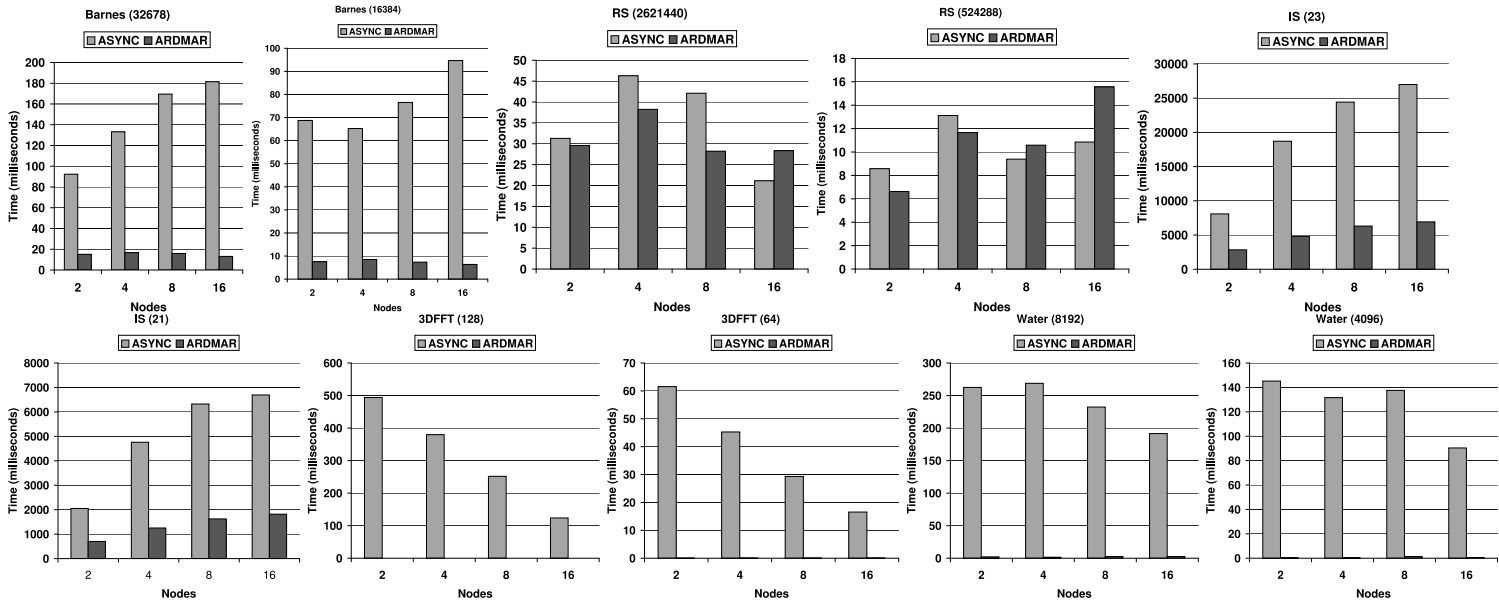

Figure 12. Average time spent in asynchronous protocol processing per node in different applications.

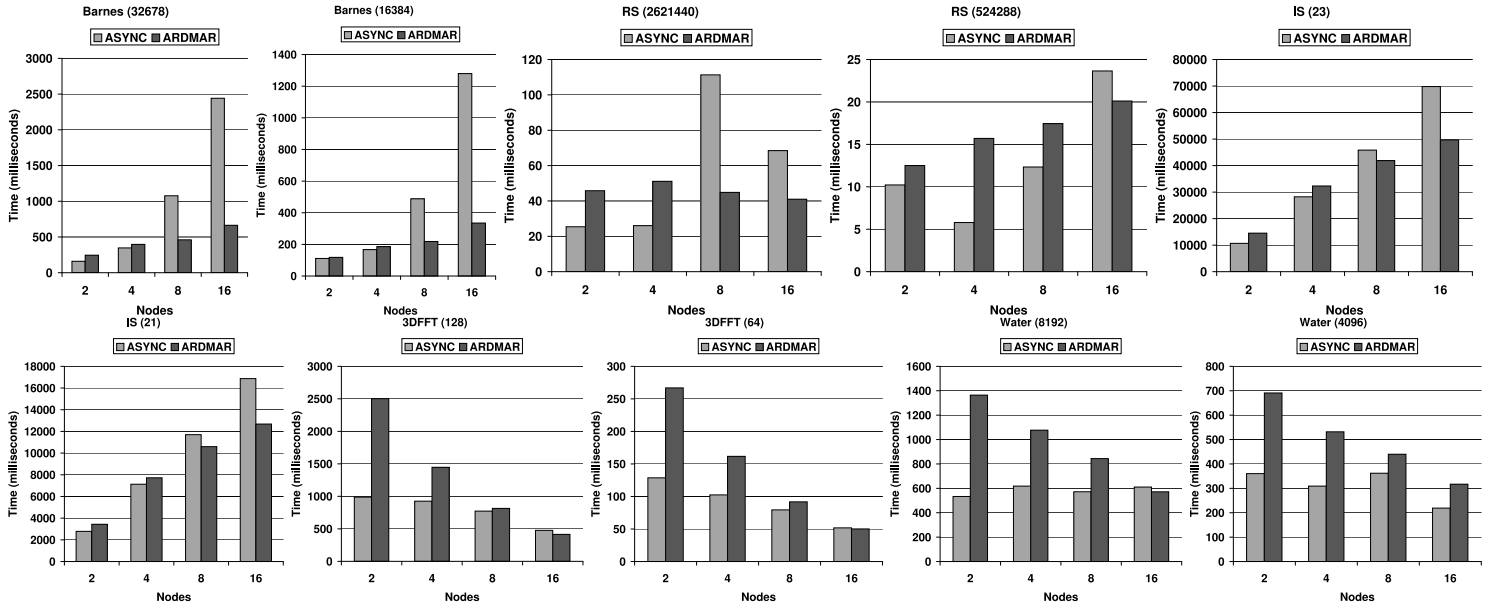

Figure 13. Average time spent for page fetching per node in different applications.

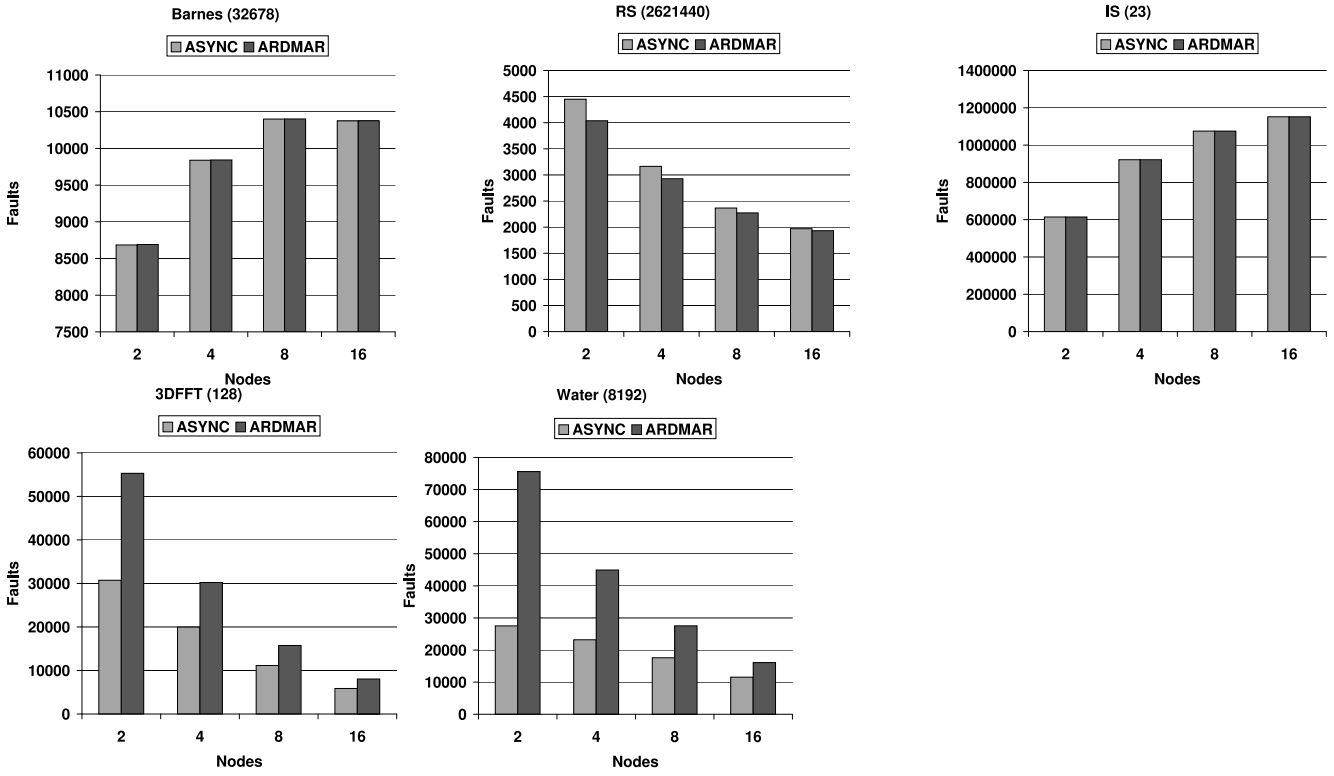

Figure 14. Average number of page faults per node for each of the larger application sizes

tives 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 [20].

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 HLRC. This was achieved through the deployment of network based support in the form of atomic operations and RDMA-Read available with modern interconnects like InfiniBand. 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. Application level evaluation in the form of system size scaling and application level scaling was performed. An improvement of up to 2.25 in the execution time of the application was observed. The proposed implementation ARDMAR also showed superior scaling properties, when compared with ASYNC the implementation of HLRC with the asynchronous protocol processing support.

Significant improvements to the protocol can be still be made. The overhead of page faults can potentially be reduced through kernel level primitives which detect writes to a page. 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] E. W. Felten, R. A. Alpert, A. Bilas, M. A. Blumrich, D. W. Clark, S. N. Damianakis, C. Dubnicki, L. Iftode, and K. Li. Early Experience with Message-Passing on the SHRIMP Multicomputer. In *International Symposium on Computer Architecture (ISCA)*, pages 296–307, 1996.

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

[14] 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.

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

[16] 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.

[17] 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.

[18] 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.

[19] 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.

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

[21] 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.

[22] 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.

[23] 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.

[24] 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.

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

[26] Topspin Communication. http://www.topspin.com. *www.topspin.com*.

[27] 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.