High-Performance Networking for Software DSMs *

Rodrigo Weber dos Santos, Ricardo Bianchini, and Claudio L. Amorim

COPPE Systems Engineering
Federal University of Rio de Janeiro
Rio de Janeiro, Brazil 21945-970

{rodrigo,ricardo,amorim}@cos.ufrj.br

Abstract

Several messaging software architectures (MSAs) have been proposed and implemented for high-performance local-area networks (LANs). Several of these MSAs have been successful at providing low latency and high bandwidth to user-level processes that communicate via explicit message passing. In this paper we claim that these MSAs are suboptimal for page-based software distributed shared-memory systems (software DSMs), as they do not consider the specific characteristics of these systems. We support our claim by studying the communication behavior of several applications running on top of the TreadMarks system and by showing that no previously-proposed architecture is ideal for the observed behavior. Finally, we propose a novel MSA for the Myrinet LAN that is tailored to software DSMs. This new design includes isolated features from some other MSAs, offering reliable message delivery, optimizations for both short and long messages, and several options for message arrival notification. In addition, our proposed MSA relies on a communication model that simplifies buffer management, while reducing latency response for request-reply operations.

1 Introduction

The recent advent of high-performance local-area networks (LANs), such as ATM and Myricom's Myrinet, has increased the impact of the messaging software on the overall communication performance. Unfortunately, the traditional messaging software architecture (MSA) incorporated in Unix leads to various time-consuming operations, such as several crossings of the operating system boundary, plenty of data copying at both ends, and frequent protection checks, that hurt performance tremendously. As a result, several new MSAs have been proposed that entirely remove the operating system from the critical communication path, providing direct user-level access to the network interface and avoiding excessive data copying. Protection checks are still performed by the operating system however, but protection is only enforced once, instead of every time a message is sent. These features allow these MSAs to reduce the messaging software overhead significantly, bringing messaging latency and bandwidth close to the network hardware limits.

From our point of view, the main problem with these recently-proposed MSAs is that they are optimized for user-level processes that communicate via explicit message passing. Under most MSAs, applications that communicate via some form of shared memory must use exactly the same mechanisms as message-passing applications use. However, we claim that shared-memory systems have several characteristics that can be used in improving the messaging performance even further.

In order to support this claim, we focus on MSAs [15, 5, 8, 1, 13, 7, 14, 6, 11, 3] that have been proposed and implemented for the Myrinet LAN and on applications running on top of page-based software distributed shared-memory systems (software DSMs). More specifically, we support our claim by studying the communication behavior of several applications running on top of the

*This work was supported by CNPq, CAPES and FINEP.

TreadMarks software DSM system and by showing that no previously-proposed MSA is ideal for the observed behavior. These MSAs rely on one or more features that do not match the TreadMarks communication requirements.

Based on the limitations of previously-proposed systems, we propose a novel MSA for the Myrinet LAN that is tailored to software DSMs. This new design includes isolated features from some other MSAs, offering reliable message delivery, optimizations for both short and long messages. and several options for message arrival notification. In addition, our proposed MSA relies on a communication model that simplifies buffer management, while reducing latency response for request-reply operations.

The remainder of this paper is organized as follows. The next section describes the Myrinet hardware, presents the main characteristics of TreadMarks, and discusses the main issues involved in MSAs. Section 3 presents the TreadMarks communication behavior and shows that no previously-proposed MSA is ideal for software DSMs. Section 4 proposes a new MSA that is tailored to software DSMs in general and TreadMarks in particular. Finally, section 5 presents our conclusions and discusses work that we intend to do in the near future.

2 Background

2.1 The Myrinet LAN

The Myrinet [4] is a high-speed LAN produced by Myricom. It consists of three basic components: a switch, a network interface (NI) card per node, and the cables that connect each card to the switch. Myrinet can deliver 1.28 + 1.28 Mbits/s full duplex bandwidth per link ensuring hardware flow control via back-pressure, in-order delivery, and extremely low error bit rates.

The Myrinet switch is a wormhole routing switch that is based on the source routing method, i.e. the routing information must be attached to the head of the message at the source. The NI card contains three DMA engines, a special network controller called LANai and up to 512 KBytes of fast SRAM. One of the DMA engines takes care of extracting incoming messages from the network link to the SRAM, another moves data in the opposite direction, and a third engine moves data from NI SRAM to host memory and vice-versa. The LANai processor runs at 33 MHz, controls the DMA operations, and runs the low-level layer of an MSA. A complete MSA also involves software for the host processor providing the interface with the NI.

2.2 Page-Based Software DSMs

Implementing DSM in software is not a trivial task, since the shared data management must be performed without generating excessive software overhead. The so-called page-based software DSMs approach this problem by taking advantage of hardware built into most microprocessors (the virtual memory protection bits) to detect potential coherence problems and enforce coherence at the page level. In order to minimize the impact of false sharing, these DSMs seek to enforce memory consistency only at synchronization points, and allow multiple processors to write the samė page concurrently [2].

TreadMarks is an example of a page-based DSM system that enforces consistency lazily. In TreadMarks, page invalidation happens at lock acquire points, while the modifications (diffs) to an invalidated page are collected from previous writers at the time of the first access (fault) to the page. The modifications that the faulting processor must collect are determined by dividing the execution in *intervals* associated with synchronization operations and computing a *vector timestamp* for each of the intervals. A synchronization operation initiates a new interval. The vector timestamp describes a partial order between the intervals of different processors. Before the acquiring processor can continue execution, the diffs of intervals with smaller vector timestamps than the acquiring processor's current vector timestamp must be collected. The previous lock holder is responsible for comparing the acquiring processor's current vector timestamp with its own vector timestamp and sending back write notices, which indicate that a page has been modified in a particular interval. When a page fault occurs, the faulting processor consults its list of write notices to find out the diffs it needs to bring the page up-to-date. It then requests the corresponding diffs and waits for them to be (generated and) sent back. After receiving all the diffs requested, the faulting processor can then apply them in turn to its outdated copy of the page. A more detailed description of TreadMarks can be found in [10].

2.3 Messaging Software Architecture

While avoiding operating system calls has proven extremely beneficial to high-performance networking, there are other important issues related to the functionality and performance of MSAs: the communication model; whether data transfers are implemented via DMA or programmed I/O operations; whether it is possible to transfer data without making intermediate copies; whether communication is reliable; whether the NI implements any pipelining of messages; and how the destination of messages are notified of their arrival. In the next few subsections, we discuss these issues in turn.

2.3.1 The Communication Model

The communication model has to do with the way communication operations are effected. Ideally, the communication model supported by the MSA should match the communication model of higher-level layers and applications. The most common communication models are:

Remote Memory Write. In this communication model, the sender (via a address translation setup phase) knows where to store the data at the destination; each message carries the memory address where the data should be stored. Messages in this model are usually sent explicitly and received implicitly.

Message Passing Rendezvous. In this model, communication only takes place if a receive primitive is executed before the message arrives, forcing send and receive primitives to synchronize.

Queue of Buffers. In this model, communication uses primitives that manage queues of buffers. It is up to the receiver to define where an incoming message should be placed, by using a queue of free buffers. Each entry in the queue comprises a buffer descriptor (address and length) that is used when a message arrives. After transferring the message to the specified buffer, the descriptor is moved to another queue, the reception queue, from which the user may check where received messages have been placed. Another queue of descriptors is used for sending messages. The sender simply provides the descriptors for where the data to be transferred has been placed.

Handler-Carrying Message. Rather than a communication model per se, handler-carrying messaging is more like a mechanism that can be used in most other communication models. In this mechanism, a message carries the address of the handler that should be executed on the receiver side upon message arrival. The handler task is then used to extract the message from the network and integrate it into the on-going computation.

2.3.2 Data Movement via DMA vs. Processor Programmed I/O

During communication, messages must be transferred from the host memory to the NI and vice versa. There are two strategies for effecting these transfers: via a direct-memory-access (DMA) device or via programmed I/O by the host itself. DMA operations are usually more efficient than programmed I/O and free the host processor from spending time on data movement. For short messages however, the DMA setup time may be as long as the data transfer itself, making programmed I/O more appropriate. In addition, given that the DMA device only deals with physical addresses while user processes deal with virtual addresses, the user cannot configure the

DMA to transfer data straight to the NI. The usual approach is then to copy data to a buffer inside the operating system kernel or to a pinned-down buffer (a region of memory in user space but with a well-known physical address and with pages marked as unswappable) before initiating the DMA operation. This generates extra overhead since the memory copy performance of today systems is as good as network performance. Thus, using the processor itself to transfer data can eliminate the extra copy, achieving the so-called zero-copy data transfers.

2.3.3 Zero-Copy Data Transfers

The downside of letting the host processor move the data itself to the NI is that the processor is kept involved in the network operation for too long when transferring long messages. To avoid this and yet achieve zero-copy transfers, several MSAs have proposed that a TLB-like structure should take care of the virtual-to-physical address translations, allowing the host to use the DMA device more easily. Managing this TLB structure normally involves the operating system for two operations: assigning physical pages to represent virtual ones and marking all pages with translations in the TLB unswappable.

Sometimes the zero-copy techniques are not enough to really eliminate extra copies. For instance, if a message should contain data spread over the user-space, the user may have to pack the spread data into a contiguous buffer before issuing the network operation. To avoid this extra copy, the MSA should provide scatter/gather operations.

Extra copies may also be necessary when implementing techniques for ensuring reliable message delivery (discussed in the next subsection). When retransmissions may be required for reliability, message data must be copied to special buffers in a way that the data is kept unmodified by the user process until the delivery is guaranteed.

2.3.4 Reliable Message Delivery

Most distributed applications require that reliable message delivery be guaranteed for proper execution. Message transfers may be unreliable however, due to unreliable network hardware and/or flow control (buffer overflow) problems. Flow control problems are usually eliminated with window-based strategies, which maintain in-order message delivery.

Traditionally, reliable delivery has been implemented by reliable (but slow) protocols such as TCP or by the user with unreliable (and faster) protocols such as UDP. The latter option is usually preferred when performance is an important issue. Thus, modern MSAs that provide reliable message delivery can be extremely useful in that they reduce the messaging overhead while avoiding the cost and complexity of message source buffering, timeout, and retry at the application level. In addition, since the lower software layers guarantee message delivery, the buffers involved during a send operation can be re-used by the application as soon as the data is transferred to the NI. This feature can eliminate undesired extra copies, while reducing buffer management overheads.

2.3.5 Message Pipelining

There can be up to four DMA operations involved during a message transfer: from host memory to NI memory (host-send), from NI memory to network (NI-send), from network to NI memory (NI-recv), and from NI memory to host memory (host-recv). One way of increasing throughput is to overlap multiple message transfers by pipelining DMA operations on the sender and/or receiver side. On the sender side of a transmission, for instance, a host-send operation may start a new message transfer even before the NI-send operation of previous message finishes. We refer to this technique as inter-message pipelining.

Another technique extends this idea by overlapping different DMA operations belonging to a single message transfer. This strategy not only increases throughput but also reduces per message latency in the same way as wormhole routing does. On the sender side, for instance, the NI-send operation for a message may start even before the host-send operation has finished. We refer to this technique as intra-message pipelining.

2.3.6 Message Arrival Notification

Message arrival notification is another important issue in modern MSAs in that this mechanism has a direct impact on the messaging performance. The host processor may be notified of message arrival by polling flags in the NI or by receiving interrupts triggered by the NI. The tradeoff between these two mechanisms is one of messaging latency and overhead. Using interrupts, the host processor does not need to check for message arrival since it is notified as soon as a message arrives. However, servicing an interrupt is extremely expensive in most modern microprocessors. To avoid this overhead, the host processor may check for message arrival by polling a status flag maintained by the NI. The question here is how often it should do it. If it polls too frequently, it may generate too much overhead wasting time on unnecessary checks. If it pools too seldomly, it may end-up increasing the messaging latency significantly.

2.4 Messaging Software Architectures for Myrinet

Several MSAs have been developed for the Myrinet LAN. Table 1 lists their main characteristics.

MSA	Comm Model	DMAxI/O Send	DMAxI/O Receive	0-copy Send	0-copy Receive	Scatter/ Gather	Reliabil	Pipe	Notif
LAM	Handler-carry	DMA	DMA	no	no	no	yes	no	poll
FM2.1	Handler-carry	I/O	DMA	I/O	no	yes	yes	no	poll
U-Net	Queue	hybrid	DMA	no	no	no	no	no	both
BIP	Rendezvous	hybrid	hybrid	TLB	TLB	no	no	intra	poll
PM1.2	Queue + RMW	DMA	DMA	TLB	TLB	no	yes	intra	both
VMMC2	RMW + redir	hybrid	DMA	TLB	TLB	no	yes	inter	both
Trapeze	Queue	hybrid	hybrid	TLB	TLB	no	no	intra	both
BDM	Queue	I/O	I/O	I/O	I/O	no	yes	no	poll
MyriAPI	Queue	DMA	DMA	no	no	yes	no	no	poll
LFC	Queue	I/O	DMA	I/O	TLB	no	yes	no	both

Table 1: Summary of Features

3 Software DSM Behavior and Implications to MSAs

In this section we discuss the communication behavior of software DSMs using TreadMarks as a representative example of this type of system. This behavior is then related to the characteristics an MSA should have in order to support software DSMs effectively.

3.1 TreadMarks Communication Behavior

Communication in invalidation-based software DSMs such as TreadMarks always occurs in request-reply fashion. These software DSMs exhibit the request-reply behavior in data management, through data (in the form of pages and diffs in case of TreadMarks) request and reply messages, as well as in synchronization, through lock request and grant messages, and barrier arrival and departure messages. In general, for both data management and synchronization messages, the actions taken on the requesting and replying sides are the same. On the requesting side, the node issues a request message and immediately starts waiting for the corresponding reply. On the replying side, the node is notified of the arrival of a request, the appropriate handler extracts the request from the network, services it, and issues a reply.

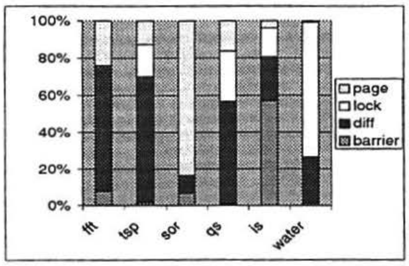

Figure 1: Normalized Number of Request-Reply Operations.

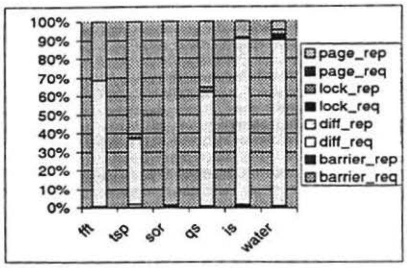

Figure 2: Normalized Number of Bytes per Message Type.

This request-reply behavior is an important characteristic of software DSMs, however more detailed information about the communication requirements of these systems is necessary for a careful study of MSAs. More specifically, to determine whether data transfers in software DSMs should be implemented via DMA or programmed I/O operations and what type of message pipelining (if any) should be implemented, it is important to assess the frequency and size of each request-reply operation and message type. To determine whether zero-copy transfers are a viable option, it is important to assess the potential for high page pinning and unpinning overhead. To determine whether reliable message delivery is necessary, it is important to assess the number of unnecessary message retransmissions. To determine whether scatter/gather features are useful, it is important to determine if messages that scatter or gather data are common and the number of chunks involved in the messages is significant.

Below we collect and present these statistics for the particular case of TreadMarks. Our experiments were performed on an 8-node system composed by two SparcStation20 nodes and six SparcStation4 nodes connected by 10 Mbits/s Ethernet network. We measured the system running several applications (FFT, QS, IS, SOR, TSP, and WATER) with diverse data access and synchronization characteristics. FFT, IS, SOR are dominated by single-writer pages that are written almost entirely whenever they are touched, while TSP and Water are dominated by multiple-writer pages, only a small fraction of which are written when touched. In terms of synchronization, applications can be classified as those that use locks and barriers (WATER, QS, IS), those that only use barriers (FFT and SOR), and those that only use locks (TSP). The input sizes used in our experiments are the default ones suggested by their corresponding distributions.

DMA, programmed I/O, and pipelining. We present the frequency and size of each request-reply operation and message type in figures 1 and 2. Figure 1 presents the relative number of request-reply operations classified as pages, diffs, locks, and barriers. The figure shows that more than 90% of the request-reply operations are either page or diff-related for most applications. Exceptions are the IS and WATER applications, for which page and diff-related operations account for almost 30% of total number of request-reply transactions. From these numbers we can observe that page and diff replies account for about 40% of all messages in most cases.

Figure 2 presents the relative number of bytes transferred per message type. The figure demonstrates that more than 90% of all bytes transferred are related to diff and page reply messages for all applications. Messages are generally short, except for diff and page reply messages; page replies are 4 KBytes long, while diff replies are longer than 1 KByte for 4 of our applications. The combination of all these results demonstrates that TreadMarks relies strongly on very long messages to amortize the significant costs of implementing shared memory in software.

Zero-copy. We assessed the potential for high page pinning and unpinning overhead in TreadMarks by counting the number of page and diff reply messages that used a specific page for buffering. The greater the number of messages per page buffer, the lower is the overhead.

Page re-use in page reply messages can be as good as 6 on 8 nodes, as we have observed for TSP and IS, while being greater than 4 for all other applications, except for SOR, where re-use is insignificant. Page re-use in diff replies is even better than in page replies; it is more than 10 for WATER, FFT, and TSP. The other applications exhibit page re-use of 3 or more. These results show that the amount of re-use is significant in the vast majority of cases.

Reliable message delivery. TreadMarks uses the unreliable (and operating system-based) UDP protocol for communication and ensures delivery by treating reply messages as acknowledgements and using timeouts. Retransmission of request messages is effected whenever the timeout expires. Out of the retransmissions generated by the system, we measured the ones that were not necessary, i.e. the timeout expired but the reply was in fact coming; it simply was late. What we found is that, depending on the application, unnecessary retransmissions can overload the network as well as generate extra overhead on the replying side. For instance, we found that 40 and 65% of the barrier arrival messages in SOR and WATER, respectively, are useless retransmissions due to the load imbalance. As another example, lock contention generates more than 10% unnecessary lock request retransmissions in TSP.

Scatter/gather. We assessed the usefulness of scatter/gather features by determining the average number of diffs that are included in diff reply messages, since these diffs are normally spread all over the virtual address space. These measurements show that TSP, QS, IS and WATER must gather, on average, more than 3 diffs per diff reply message, while FFT and SOR must gather about 2 diffs on average.

3.2 Implications to MSAs

The communication behavior and measurements discussed in the previous section have several consequences with respect to the features that MSAs must provide to software DSMs. We will now discuss each of the main features listed in section 2.3 in light of the previous section, commenting on the features presented by the MSAs described in section 2.4.

3.2.1 The Communication Model

The communication model provided by the MSA should allow asynchronous messaging. This requirement comes from the fact that the arrival of a request message cannot be predicted. As a result of this asynchrony, the rendezvous model (BIP) becomes inappropriate, as it requires communicating processors to synchronize at the communication point.

The model should also facilitate buffer management for asynchronous messages if the request-reply behavior is to be relaxed as in page-based software DSMs such as ADSM [12], AEC [16], or HLRC [17], where some form of update coherence is applied. This restriction makes the remote memory write model (VMMC2) inadequate, since neither senders nor receivers can control the receive buffer usage. In order to avoid overwriting messages in this model, a large amount of buffer space must be coupled with explicit user management of buffers.

For TreadMarks, buffer management is not such a significant problem in the remote memory write model, because of the request-reply communication style of the system. We can simply export N-1 receive buffers for an N-node system. Note however that the size of exported buffers should be enough to accept an entire request message. The results in section 3.1 show short average request message sizes, but the maximum size of requests was sometimes significant. For instance, we find the longest barrier request messages in WATER and SOR to be 1400 and 1100 bytes long, respectively. Thus, the remote memory write model could be used in TreadMarks, but this would simply waste memory and lead to poor scalability.

3.2.2 Data Movement via DMA vs. Programmed I/O

Adopting a hybrid scheme where short messages are transferred with programmed I/O instructions and long messages are transferred with DMA operations seems ideal for software DSMs.

As we saw in section 3.1, requests are usually short messages, so the host should transfer the data to the NI using programmed I/O instructions when sending a request. When receiving a request, programmed I/O by the receiving host should certainly perform well, but using the DMA to transfer the message to the host memory could also be useful. The DMA transfer could be overlapped with the interrupt overhead. Replies are long messages and, thus, sending or receiving a reply should use the help of the DMA.

The MSAs we consider fulfill these send and receive requirements, allowing for hybrid implementations of data movement, except for AM, FM, PM, LFC, and BDM.

3.2.3 Zero-Copy Data Transfers

Given the large size and number of page and diff reply messages in software DSMs such as TreadMarks, zero-copy transfers should be provided by the MSA to avoid extra-copies of these messages on the requesting side and, as a result, decrease latency response. Zero-copy data transfers should achieve good performance for TreadMarks, since sending a page or diff reply may take advantage of previously-pinned pages. The results in section 3.1 show that the page re-use is usually significant in TreadMarks for both page and diff replies. In addition, section 3.1 suggests that a gather interface is required for diff replies.

Even though the page re-use numbers for diff replies suggest that zero-copy should not need an excessive amount of pinning, diffs are frequently allocated on different pages. This can potentially increase overhead, as several pages may have to be pinned for a single diff reply message. We propose an efficient implementation of zero-copy that should overlap unavoidable page pinning overheads with the reply latency, since a TLB lookup and page pinning system call together may cost as much as actually making a copy of the data.

Yet another approach for the efficient implementation of zero-copy is to allocate a chunk of contiguous unswappable pages in the kernel memory statically, avoiding dynamic pinning overheads. The physical address of the chunk should be placed on the TLB. The user should then implement copying but only when space is exhausted. This approach could be implemented for TreadMarks, which allocates 1 MByte of memory for placing diffs.

Pinning pages on demand without any control as done under VMMC-2, Trapeze, and BIP could induce excessive overhead and consume too much memory space on the node. To alleviate the memory consumption problem, one should use the pin-down cache technique proposed by PM. To avoid the excessive pinning overhead and alleviate the consumption problem, one should use our proposed approach.

3.2.4 Reliable Message Delivery

As aforementioned, unnecessary retransmissions in TreadMarks can overload the network as well as generate extra overhead on the replying side. The problem is that tuning the timeout value is a very hard task for any system, as this value depends not only on the network in use, but also on the specific timing of distributed systems. Implementing mechanisms for reliable message delivery on top of a MSA that does not provide reliability has been shown inefficient in certain cases; this type of implementation can increase communication overhead up to 200% [9].

Thus, to avoid problems associated with a large number of retransmissions, message delivery should be reliable in software DSMs. In-order delivery is not necessary for systems based on the request-reply model, since a node can issue a single request at a time. Efficiently guaranteeing reliable delivery for a high-performance LAN that does not drop packets (Myrinet being an example) boils down to implementing flow control without timeouts. Several flow control strategies provide in-order delivery for free.

Not all previously-proposed MSAs provide reliable message delivery. The ones that do are AM, FM, VMMC-2, PM, LFC, and BDM.

3.2.5 Message Pipelining

Again, given the large size and number of page and diff reply messages in software DSMs, an important feature of an MSA for these systems is intra-message pipelining. However, we can see in table 1 that only PM, BIP and Trapeze provide this feature.

3.2.6 Message Arrival Notification

Given the request-reply communication style of invalidate-based software DSMs, it becomes clear that for a requesting node all that matters is latency response, while for a replying node all that matters is the overhead that servicing the request will entail. For this reason, after issuing a request, a node must poll for message arrival. On the replying side however, as surprising as this may sound, the best option is interrupt-based notification in most cases.

Although interrupts normally generate high and undesired overheads, it is very difficult to implement a polling strategy that would not increase latency response for software DSMs. The reason for this is that, in the absence of compiler or executable code editing techniques, polling could only happen when running the software DSM code itself. This limitation would most likely make polling two infrequent to be useful.

Interrupt-based notification is not always the best option when receiving a request. Barrier arrival requests, for instance, should never interrupt the barrier manager for best performance. The barrier manager should simply check for these messages when it arrives at the barrier and poll for other arrival messages, if necessary. Thus, the ability to control whether a message should interrupt the receiver is a useful feature of MSAs. Only two MSAs provide this feature however: VMMC-2 does it by including a notification in the message itself and PM allows it through channels with different properties.

In summary, we find that both polling and interrupts must be provided by the MSA. From table 1 we see that this single requirement disqualifies most current MSAs developed for Myrinet (AM, FM, BIP, BDM, and MyriAPI). These systems are tailored to the message-passing programming model and strongly depend on polling for good performance.

4 Proposal for a New MSA

It is clear from the discussions above that no previously-proposed MSA for Myrinet is ideal for software DSMs such as TreadMarks. We believe that PM is the MSA that fulfills most of the communication requirements we observed. PM ensures reliable in-order delivery, provides intra-message pipelining, and enables zero-copy data transfers by extending its queue of buffers communication model with remote memory write primitives. In addition, the processor can be notified of message arrival by either polling or interrupts. In order to fulfill all software DSM requirements four new features should be supported by PM:

- Data gathering must be provided in such a way that zero-copy can actually be achieved when sending diff replies. PM has no support for gathering;
- The user must have the choice of sending a message by using programmed I/O or by using DMA, as we have observed that programmed I/O is ideal when sending request messages and DMA is ideal when sending reply messages. PM uses DMA to transfer messages to the NI both on send and receive operations;

- Zero-copy should be implemented with statically-allocated, pinned-down buffers for sending reply messages. PM uses the pin-down cache technique, which is efficient when page locality is unpredictable, but would involve unnecessary pinning overheads for relatively small and fixed structures such as the pool of diffs in TreadMarks; and
- As we have proposed above, zero-copy overheads should be moved away from the critical path of messages. In the PM remote memory write model, TLB lookup, page pinning, and buffer address exporting should be moved away from the critical path when sending request messages. Our new technique implements this using four new objects: a reply counter, a reply TLB, a reply flag and a reply_addr() primitive. The reply TLB is stored on the NI memory. Each reply table entry contains a reply number, buffer (physical) addresses, and length. If a send operation is issued with a reply flag set to one, the reply counter is incremented and the value is sent with the request message. The remote processor may send this value back with the reply message. As soon as the request message is sent the host may pin the buffer region that will receive the reply message and inform the NI of its physical address and size. These two values, together with the reply number are used to update the reply TLB via the reply_addr() primitive. When the reply message arrives, if the message does not carry a reply number, the message is transferred as usual by using information on the queue of free buffers. If it carries a reply number, then the reply TLB is checked. If a hit occurs, zero-copy takes place. If the reply number entry is missing or the message size is greater than the buffer size, the message is transferred by using information on the queue of free buffers again.

Our entry on table 1 would then be (* marks optimizations we suggested to existing mechanisms):

MSA	Comm Model	DMAxI/O Send	DMAxI/O Receive	0-copy Send	0-copy Receive	Scatter/ Gather	Reliabil	Pipe	Notif
ModPM	Queue+RMW*	hybrid	DMA	TLB+I/O	TLB*	gather	yes	intra	both

Table 2: Summary of Features

5 Conclusions and Future Work

In this paper we have listed the main characteristics of ten messaging software architectures for the Myrinet local-area network. In addition, we have shown that none of these architectures is ideal for supporting page-based software distributed shared-memory systems such as TreadMarks. Finally, we have proposed several modifications that should tailor one messaging architecture to TreadMarks. In the near future, we plan on implementing our proposal, comparing it against other architectures, and extending it to include deviations from the request-reply model of TreadMarks.

References

[1] A. Basu, V. Buch, W. Vogels, and T. von Eicken. U-Net: A User-Level Network Interface for Parallel and Distributed Computing. In *Proceedings of 15th ACM SOSP*, pages 40–53, December 1995.

[2] J. K. Bennett, J. B. Carter, and W. Zwaenepoel. Munin: Distributed Shared Memory Based on Type-Specific Memory Coherence. In *Proceedings of the 2nd PPoPP*, pages 168–176, March 1990.

[3] R. Bhoedjang, T. Ruhl, and H. Bal. Design Issues for User-Level Network Interface Protocols on Myrinet. Technical report, Dept of Mathematics and Computer Science, Vrije Universiteit. 1998. To appear in IEEE Computer.

[4] N. Boden, D. Cohen, R. Felderman, A. Kulawik, C. Seitz, J. Seizovic, and W. Su. Myrinet: A Gigabit-per-Second Local Area Network. *IEEE MICRO*, 15(1):19–36, February 1995.

[5] C. Dubnicki, A. Bilas, K. Li, and J. Philbin. Design and Implementation of Virtual Memory-Mapped Communication on Myrinet. In *Proceedings of the 1997 IPPS*, pages 388–396, April 1997.

[6] T. Eicken, D. Culler, S. Goldstein, and K. Schauser. Active Messages: A Mechanism for Integrated Communication and Computation. In *Proceedings of the 19th ISCA*, pages 256–266, May 1992.

[7] G. Henley, N. Doss, T. McMahon, and A. Skjellum. BDM: A Multiprotocol Myrinet Control Program and Host Application Programmer Interface. Technical Report MSSU-EIRS-ERC-97-3, Mississippi State University, May 1997.

[8] H.Tezuka, A. Hori, Y. Ishikawa, and M. Sato. PM: A Operating System Coordinated High Performance Communication Library. In *High-Performance Computing and Networking '97*, volume 1225, pages 708–717, April 1997.

[9] V. Karamcheti and A. Chien. Software Overhead in Messaging Layers: Where Does the Time Go? In *Proceedings of ASPLOS-IV*, 1994.

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

[11] Andrew J. Gallatin Kenneth G. Yocum, Jeffrey S. Chase and Alvin R. Lebeck. Cut-Through Delivery in Trapeze: An Exercise in Low Latency Messaging. In *Proceedings of HPDC*, August 1997.

[12] L. Monnerat and R. Bianchini. Efficiently Adapting to Sharing Patterns in Software DSMs. In *Proceedings of the 4th HPCA*, Feb 1998.

[13] Myricom. Myrinet Specifications. http://www.myri.com/myricom/document.html, 1995.

[14] S. Pakin, M. Lauria, and A. Chien. High Performance Messaging on Workstations: Illinois Fast Messages (FM) for Myrinet. In *Proceedings of Supercomputing 95*, San Diego, CA, 1995.

[15] L. Prylli and B. Tourancheau. BIP: A New Protocol Designed for High-Performance Networking on Myrinet. In *Proceedings of IPPS/SPDP98*, 1998.

[16] C. B. Seidel, R. Bianchini, and C. L. Amorim. The Affinity Entry Consistency Protocol. In *Proceedings of the 1997 ICPP*, Aug 1997.

[17] Y. Zhou, L. Iftode, and K. Li. Performance Evaluation of Two Home-Based Lazy Release Consistency Protocols for Shared Memory Virtual Memory Systems. In *Proceedings of the 2nd OSDI*, October 1996.