# An Extensible Message Layer for High-Performance Clusters

*Craig Ulmer and Sudhakar Yalamanchili*

Critical Systems Laboratory
School of Electrical and Computer Engineering
Georgia Institute of Technology
Atlanta, GA, 30332-0250
Email: {ulmer, sudha}@ece.gatech.edu

Keywords: Middleware, Clustering, NI Flow Control

## Abstract

*Resource rich clusters are an emerging category of computational platform where cluster nodes have both CPUs as well as high-performance I/O cards. These clusters target specific applications such as digital libraries, web servers, and multimedia kiosks. The presence of communication endpoints at locations other than the host CPU requires a re-examination of how middleware for these clusters should be constructed.*

*A key issue of middleware design is the management of flow control for the reliable delivery of messages. We propose using a network interface based optimistic flow control scheme to address resource rich cluster requirements. We implement this functionality with a message layer called GRIM, and compare its general performance to other well-known message layers. This implementation suggests that the necessary middleware functionality can not only be constructed efficiently, but also in a way that provides additional middleware benefits.*

## 1. Introduction and Motivation

While work in clustering to date has effectively used commercial processor nodes, emerging multimedia and network-based applications are impacting cluster architectures through the inclusion of powerful co-processor hardware. For example, demands for high performance web servers have resulted in programmable I/O cards that directly control arrays of disks and serve network requests without host intervention. In other applications, special purpose hardware devices are being used to enhance media processing and search engine capabilities. An emerging challenge in cluster computing is the effective management of diverse hardware resources. This paper deals with the design issue of providing an extensible message layer that allows for flexible and efficient communication in this emerging form of “resource rich” clusters.

Communication in clusters is typically facilitated by low latency, high bandwidth system area networks (SANs). Commercial SANs such as Myrinet [1], ServerNet [2], and Scalable Coherent Interface (SCI) [3] have provided major leaps in performance that is two to three orders of improvement over traditional LAN hardware. A number of custom message layer packages have been written for these SANs to provide low-overhead communication among host CPUs in a cluster [4]. While this “CPU-centric” approach is ideal for clusters that perform all computations at the host level, it is generally prohibitive in terms of providing communication for endpoints located outside of the host CPU.

Thus as clusters evolve we observe that node architectures are becoming increasingly heterogeneous, where powerful peripheral devices may themselves serve as sources and sinks of data. Examples include the following.

- *Multiple Network Substrates:* Clusters often contain multiple communication interfaces for a number of reasons. Ideally these interfaces collaborate without host intervention and provide bridges between network substrates.

- *Intelligent Storage Devices:* Equipment such as the $I_2O$ server adapter card present massive storage options to peripheral devices without host intervention.

- *Hardware Accelerators:* Special-purpose co-processor devices such as FPGA cards are available for graphics acceleration, search engines, and media transformation.

- *I/O Devices:* Additional I/O devices such as cameras, video displays, and video capture devices are all common among clusters with multimedia applications.

The presence of these peripheral devices leads to the notion of *resource rich* clusters where processing is performed not only in host CPUs, but also in peripheral devices. As shown in Figure 1, these clusters have a variety of hardware resources, all of which require integration into the message layer. An example application driving this form of cluster technology is a digital library that provides access to enormous data sets to a large number of simultaneous users. An individual cluster node would be equipped with i) a disk array housing a fraction of the overall database, ii) a number of Ethernet ports to service incoming requests from remote clients, and iii) a SAN interface for high speed intra-cluster communication to provide a coordinated response to external requests. The message layer for such an application must provide efficient communication among devices in the cluster to maintain a high response throughput.

For such clusters we find it necessary to re-examine the functionality of message layers. The primary issue is that communication endpoints for the message layer can be located at multiple locations within a single cluster node. In addition to enabling communication among host-level programs, the message layer must also support communication originating and terminating at the peripheral devices. This can be accomplished by managing endpoints for all devices at the host. However, such an approach requires multiple traversals across the I/O and memory hierarchies as well as routing all traffic through the host. Moving away from a CPU-centric model of managing communication in favor of multiple endpoints per node produces a number of conceptual challenges:

- *End-to-End Flow Control:* Peer devices within a node generally do not have the same magnitude of memory and compute resources as the host CPU. Therefore the overheads of interacting with the NI become quite important. We argue that end-to-end flow control should be moved to the NI, thus reducing the responsibilities of the endpoint and simplifying the endpoint operation.

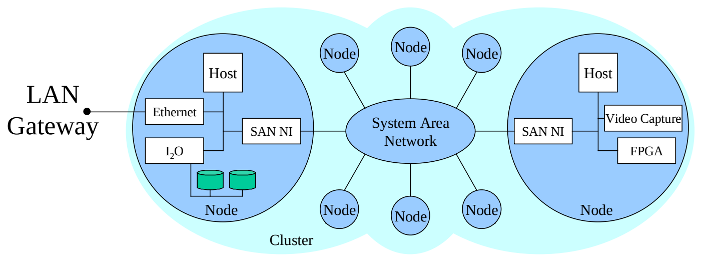

**Figure 1 : Resource Rich Cluster**

- *Multiple, Concurrent Writers:* The presence of multiple communication endpoints in a node creates the problem of multiple writers to the SAN interface. Without a means of synchronization among endpoints, it is possible that buffers will be overwritten and messages lost. Synchronization among writing endpoints must be lightweight.
- *Receive Processing:* It is important for an endpoint to be able to specify a well-defined set of methods for processing incoming messages since operations performed on reception are specific to the receiving device. While host level endpoints can have a variety of mechanisms to process incoming messages, other devices such as cameras and disks generally have specific operations that are performed on received messages.

This paper addresses one specific issue in this environment, namely the movement of flow control operations to the NI. In particular we advocate the use of optimistic forms of flow control [6] as being advantageous and present the design of an optimistic flow control protocol implemented within a Myrinet interface. The following section provides an overview of a user-level message layer that employs NI-based optimistic flow control. The remainder of the paper addresses design and performance issues related to this approach.

## 2. GRIM: General-purpose Reliable In-order Messages

GRIM is an extensible framework for user level messaging that is designed to facilitate the addition of multiple communication endpoints within a cluster node. Extension refers to ability to easily extend the functionality of the message layer to accommodate new endpoint features. Conceptually GRIM is designed with three specific characteristics to meet the needs of resource rich clusters: NI managed flow control, logical channels, and an active message style of packet reception. The NI-based flow control is specifically addressed in this paper.

### 2.1 NI-based Flow Control

To meet the basic needs of resource rich clusters, we propose moving the buffer flow control mechanisms of the message layer to the NI. In NI-based flow control, NIs use control messages to co-ordinate the reliable transfer of messages between NIs, as illustrated in Figure 2. While this approach increases the amount of work the NI must perform, there are a number of advantages that make this option attractive. A key benefit for simple endpoints is that the placing of flow control in the NI reduces the amount of network management that must be performed at the endpoint. An endpoint simply checks to see if the NI has room for a new message, and then hands the message to the NI for reliable delivery. Further, unlike host-based flow control schemes that stall injections until buffer space is available at *both* sending and receiving NIs, injection in NI-based schemes is stalled only when buffer space is unavailable at the sending NI. This approach also permits NIs to play a more active role in message transmission. By using information gathered at runtime from its neighbors, NIs can make more informed decisions as to which messages should be transmitted next. This transforms NIs from simple data transmitters to providers of QoS.

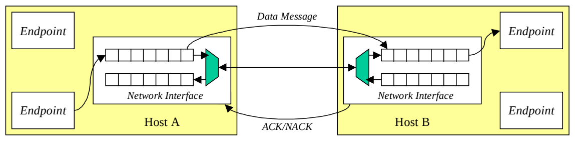

**Figure 2 : NI-based Flow Control**

NI-based flow control has been implemented in previous research projects for various reasons. The original FM work [6] first addressed the use of NI-level flow control as a means of providing improved dynamic performance. This approach focused on "optimistic flow control", where a message is transmitted under the assumption that the destination NI has sufficient space to receive the message. When a receiving NI's buffers are full, incoming messages are returned to their senders for retransmission at a later time. It follows that the sender must buffer all messages until reception is confirmed via acknowledgements. While FM performance exceeded contemporary message layers in many respects, NI complexity was a major issue and in FM 2.0 a streamlined host-level flow control scheme was adopted. In the LFC message layer [5], a NI-based flow control scheme was implemented to improve dynamic bandwidth utilization as well as provide a suitable framework for NI services such as multicast. Unlike FM, LFC was written using a NI-level sliding window protocol for flow control. This work demonstrated that complex NI functionality is not unreasonable for even low performance NIs.

### 2.2 Optimistic Flow Control

In this paper we implement a NI-based optimistic flow control strategy (a variant of that first described in [6]). NIs manage incoming and outgoing message queues and transmit individual messages assuming that ample buffer space is available at the destination NI. Sending NIs retain messages until an acknowledgement message is received from the destination NI. State information maintained in the NIs supports block acknowledgements and buffer recovery across multiple outstanding messages.

In the case of heavy loads a receiving NI may not be able to accept a message due to a lack of buffer space. At such times the NI will issue a negative acknowledgement (NACK) message to the sending NI and drop the incoming message. NIs that receive a NACK must rollback the outgoing message queue to the dropped message and retransmit all successive messages to the particular destination after a back-off time period. This "*go back n*" approach was adopted over a selective repeat scheme since network bandwidth in SANs is a more abundant than NI computational power.

While we believe that NI-based flow control is valuable for resource rich clusters, it is important to consider how the added NI complexity affects general performance. An optimistic flow control scheme can potentially degrade message layer performance for a number of reasons. Messages must contain additional information such as sequence numbers in order for NIs to remain synchronized. NI firmware must be able to decode and handle multiple types of messages as well as maintain network state information. Given the typical low processing power of SAN NIs, complex NI-based software can significantly impact performance. Thus the implementations must be carefully crafted to gain reliability and dynamic bandwidth benefits without incurring general performance penalties.

### 2.3 Logical Channels

A second characteristic of GRIM is the use of logical channels as a simple but effective mechanism for providing isolation between multiple writers of different traffic streams. At injection time users can specify a logical channel identifier that associates the message with a given traffic stream. GRIM guarantees in-order delivery only for messages with the same logical channel identifier. Thus each endpoint in a node is assigned a distinct set of logical channels eliminating the need for injection synchronization among endpoints. Logical channels are multiplexed at the NI with the use of a link scheduler. The current implementation uses a simple round robin scheduler.

### 2.4 Active Message Style Packet Handling

The active message abstraction [7] is particularly well matched to the needs of interfacing multiple endpoints within a node. In GRIM a function handler is associated with each injected message so that sending endpoints can specify precisely how the message should be received. An endpoint's capabilities are therefore abstracted into a cluster's API by defining sets of function handlers available at the endpoint. Function handlers are registered with a master node in the cluster so that all endpoints can make use of the cluster's capabilities. The active message style interface can also be used at the application level to emulate other common APIs such as MPI or sockets.

## 3. Implementation and Performance

We have implemented a basic version of GRIM over Myricom's Myrinet SAN. Our primary objective for this implementation is to demonstrate that the basic features of GRIM can be implemented without substantial penalties to traditional host-level performance. Therefore the work presented here implements the core of GRIM functionality, but provides interfaces only to host-level endpoints.

Standard performance tests were used to analyze the characteristics of the GRIM implementation and are presented with other reported results in Table 1. Latency and bandwidth measurements are based on round trip timing experiments where a single message is transmitted to a node and returned. Bandwidth tests used large (one-megabyte) messages and fragmented the messages into individual 48 KByte packets. The bandwidth half-power message size was observed at 6 KBytes. From these measurements we see that GRIM performance is comparable to other message layer implementations.

In addition to latency and bandwidth, we measured the overhead and gap parameters as defined in the Berkeley LogP model [8]. The key observation here is that the buffer management scheme affects the value of the gap parameter. As is common with most message layers, GRIM demonstrated a very small injection overhead for the host, approximately 1 μs for small sized messages. However, significant differences exist for the value of the gap, which is defined as the minimum time between successive message injections. The gap value was obtained by measuring the total time required to inject a burst of small messages. The number of messages that can be successively injected is limited by the value of the gap, which in turn is determined by the flow control protocol. Our experiments compared two flow control protocols: a host credit-based protocol and the optimistic flow control protocol implemented in the NI for GRIM. For all experiments the total buffer size in the NI was limited to 128 short messages. As seen in Figure 3, the gap between messages converges at about 21 μs for both NI-based flow control and host level flow control.

For a comparative evaluation of NI flow control, we imposed injection policing on top of GRIM to obtain the performance of host-based flow control. In these tests hosts were limited to having a fixed number of outstanding messages at any given time. This restriction reflects static barriers that host level flow control schemes assert to provide reliable delivery. For example, in a 16-node cluster where we have buffer space for 128 messages, static credit-based flow control schemes allow only eight outstanding messages to any given node. Achievable message latencies and overall bandwidth is throttled. From these tests we can see that injection policing required by credit based schemes causes injecting nodes to unnecessarily block for a larger amount of time compared to the optimistic

| | AM [9] | FM [10] | LFC [5] | BIP [11] | GM [12] | GRIM |
|---|---|---|---|---|---|---|
| Host-to-Host Latency | 10 μs | 9.6 μs | 10.4 μs | 4.8 μs | 18 μs | 13 μs |
| Maximum Bandwidth | 38 MB/s | 100 MB/s | 76.2 MB/s | 126 MB/s | 140 MB/s | 106 MB/s |

**Table 1 : Comparison of Middleware Performance**

implementation. For message bursts of 256 messages or less, the differences in the value of gap are substantial. Therefore optimistic flow control matches burst demand with the dynamically allocated buffer space.

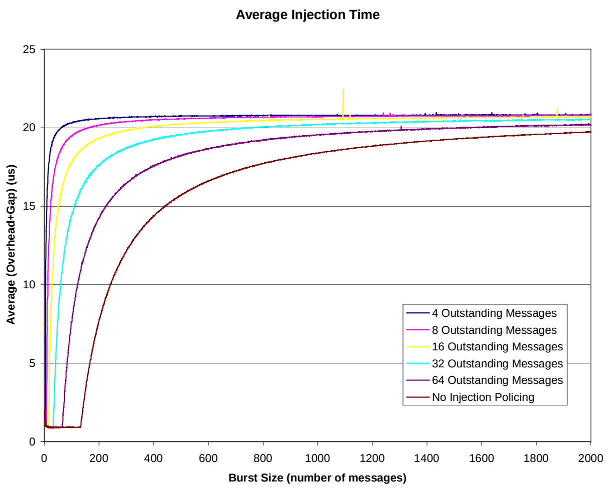

**Figure 3: Injection Overhead and Gap for Bursts**

## 4. Conclusions

The emerging category of resource rich clusters is a commercially important domain that warrants the re-examination of middleware design. With the proper adjustments it is possible to adapt previous middleware techniques to address the needs of these clusters. At the core of middleware design is the reliable management of buffer space, for which NI-based flow control schemes can have a significant impact on messaging performance. Through optimistic NI-based flow control, we have demonstrated that increased NI functionality can be accomplished without serious degradation to general performance. This functionality adds a great amount of value to middleware by reducing endpoint network interactions, increasing general reliability, and allowing for network scalability. These characteristics are essential for the emerging next generation of clusters.

## References

[1] 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, Vol.15, No.1 1995.

[2] R. Horst and D. Garcia. *Servernet SAN I/O Architecture*. In Hot Interconnects Symposium V, August 21-23 1997

[3] *Scalable Coherent Interconnect,* IEEE Standard 15961992, 1992

[4] R. Bhoedjang, T. Ruhl, and H. Bal. *User-Level Network Interface Protocols*. IEEE Computer, Vol.31, No.11, P53-60, 1998.

[5] R. Bhoedjang, T. Ruhl, H. Bal. *LFC: A Communication Substrate for Myrinet* in Fourth Annual Conference of the Advanced School for Computing and Imaging, 1998

[6] S. Pakin, M. Lauria, and A. Chien. *High Performanc Messaging on Workstations: Illinois Fast Messages (FM) for Myrinet* in Supercomputing '95, 1995

[7] T. von Eicken, D. Culler, S. Goldstein, and K. Schauser. *Active Messages: A Mechanism for Integrated Communication and Computation*. Proceedings of the 19th Annual International Symposium on Computer Architecture (ISCA). May 1992.

[8] D. Culler, R. Karp, D. Patterson, A. Sahay, K. Schauser, E. Santos, R. Subramonian, and T. von Eicken, *LogP: Towards a Realistic Model of Parallel Computation*, Proceedings of the Fourth ACM SIGPLAN Symposium on Principles and Practice of Parallel Programming, May 1993.

[9] R. Martin, L. T. Liu, V. Makhija, and D. Culler. *Lanai active messages (lam).*

[10] S. Pakin, V. Karamcheti, and A. Chien. *Fast Messages (FM): Efficient, portable communication for workstation clusters and massively-parallel processors*. IEEE Parallel and Distributed Technology, 1997.

[11] L. Prylli and B. Tourancheau, *BIP: A New Protocol designed for High-Performance Networking on Myrinet*. In Proceedings of PC-NOW IPPSSPDP98, 1998.

[12] Myricom, Inc. The GM message layer, http://www.myri.com