Sender Coordination in the Distributed Virtual Communication Machine

Marcel-Cătălin Roşu and Karsten Schwan
Georgia Institute of Technology, College of Computing
Atlanta, GA 30332-0280
{rosu, schwan}@cc.gatech.edu

Abstract

The Distributed Virtual Communication Machine (DVCM) is an extensible communication architecture for tightly-coupled clusters of workstations (COWs) connected by high-speed networks. The DVCM is designed for off-the-shelf network interface cards equipped with communication coprocessors. Its main component is an "active backplane" implemented in firmware running on the coprocessors. This "backplane" can be extended with modules that implement application-specific functionality and have access to some of the application's state. Consequently, non-trivial collective computations can be implemented as DVCM extensions.

We present a DVCM extension module that provides application-specific network flow control by coordinating the resource-competing components of a parallel application running on an ATM LAN. Our experiments show that this extension module helps eliminate message loss and achieve high link bandwidth utilization when there is significant link contention.

1. Introduction

The high availability of fast workstations and high-speed interconnects for local area networks makes possible the use of moderate-cost clusters of workstations (COWs) for high-performance computing. However, the performance of parallel applications running on COWs does not only depend on the node's speed or on the interconnect's bandwidth but also on the communication overheads and latencies experienced by applications. Unfortunately, in COWs, data transfers between application and interconnect exhibit relatively large overheads and/or latencies, as an interconnect attaches to a node through the workstation's I/O bus.

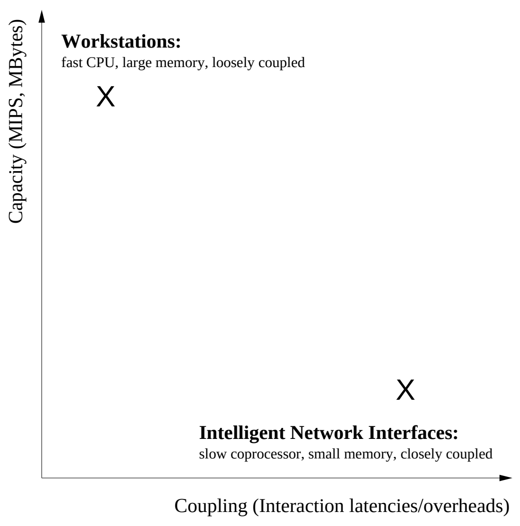
Our research is focused on building communication architectures for COWs that attempt to minimize the negative effects of application-network interaction overheads and of cluster latencies on parallel application performance. We aim to achieve this while avoiding CPU- or bandwidth-consuming communication protocols, thereby preserving the benefits of current technological trends, i.e., the availability of fast CPUs and high-bandwidth interconnects.

We propose a cluster communication framework, the Distributed Virtual Communication Machine (DVCM), designed to provide low latency, high bandwidth, and low overhead communication to cluster applications that benefit from a tightly-coupled parallel execution platform. The two principles guiding the DVCM design are:

- enable the integration of application-specific modules into the DVCM framework; and
- take advantage of the existence of low-cost off-the-shelf intelligent network interface (NI) cards.

The DVCM acts like an "extensible and active backplane" for the components of a cluster application. It proposes novel application-level abstractions of the cluster network that can be implemented efficiently in software and firmware running on the workstation's CPU and on the NI's communication coprocessor, respectively. The DVCM extensible architecture is motivated by our belief that no single predetermined set of primitives can best satisfy the needs of all existing and future parallel and high-performance distributed applications. Since most related research provides highly efficient cluster communication with a predetermined set of primitives, our research complements rather than replaces such solutions.

On clusters based on intelligent NIs, we identify a two-level hierarchy of computational resources: NI resources (coprocessor and memory) constitute the lower level, and workstation resources (CPU and memory) constitute the upper level. Compared to the workstations' resources, the NIs' resources are less powerful but, latency-wise, they are much "closer" to each other, as they are separated only by the switch fabric. In addition, the NIs' resources are more closely coupled than the workstations' resources because delays in handling coprocessor-to-coprocessor messages are reduced further due to their dedication to this task (see Figure 1). These observations guide our approach to using both NI and workstation resources to run cluster applications as follows:



Capacity (MIPS, MBytes)
Workstations:
fast CPU, large memory, loosely coupled
X
X
Intelligent Network Interfaces:
slow coprocessor, small memory, closely coupled
Coupling (Interaction latencies/overheads)

Figure 1. Workstation vs. NI Resources

- *NI resources* are dedicated to the modules that involve significant collective coordination/communication, little processing, and small amounts of state, while
- *workstation resources* are used for the CPU- and memory-intensive modules that require little internode interaction.

As a consequence, the DVCM enables parallel applications to use NI resources for the modules that can benefit most from the close coupling of the cluster NIs, and to use workstations for the CPU- and memory-intensive modules. Sample benefits for modules running on NIs includes those derived from eliminating expensive application-level synchronization or derived from the use of messaging services with lower latencies than available at user level.

Clearly there are limitations to the placement of additional functionality on NIs, typically due to the restricted NI resources. However, we have found that commercial NIs have resources in addition to those needed for basic NI functionality. This is because, in most cases, off-the-shelf components are used for both NI coprocessor and memory. Consequently, it is very unlikely that a perfect match exists between the resources installed on the NI and the resources required for driving the network link at full speed. Most likely, every well-designed NI card hosts a certain amount of unused resources. Furthermore, our previous work on the Virtual Communication Machine (VCM) [17] has demonstrated that the amounts of additional resources required by DVCM plus typical application-specific extensions are small.

This paper demonstrates the benefits of the DVCM extensible architecture with an extension module that implements a task useful in most parallel applications: coordination among the resource-competing components of a parallel/distributed application. This extension addresses situations in which there is contention for communication resources (e.g., link bandwidth, receiver buffers) among application components. Many applications running on COWs exhibit very poor performance when the network is overloaded. In particular, this problem is significant for networks without physical-layer flow control, such as ATM, which exhibit dramatic message losses when links are overloaded.

We demonstrate that this DVCM extension module enables the elimination of message loss while it adds no overhead to the workstation CPU. The DVCM extension provides network flow control with a simple, token-based protocol. The experiments using the DVCM extension module show negligible message loss and high link bandwidth utilization.

The next four sections of the paper (1) introduce the DVCM architecture and compare it with our earlier work on the Virtual Communication Machine, (2) describe the DVCM extension model, (3) describe the extension module for sender coordination and its performance, and (4) provide a brief overview of the related research.

2. The Distributed Virtual Communication Machine

The DVCM architecture is the natural extension of our previous work on the Virtual Communication Machine (VCM) ([16, 17]). The VCM is a programmable abstraction of the network interface. It consists of an execution unit and an address space. The VCM execution unit implements a simple set of commands, most of which are used for assembling/disassembling application data from/into network messages. The command set is extensible, enabling the transfer of communication-related processing from the application to the VCM implementation. Although all of our VCM implementations only allow compile-time extensions of the command set, dynamic extensions are also possible. The VCM address space comprises all of the memory pages registered with the network abstraction by local applications. All registered pages must be pinned in host memory. Application components interact with local VCMs through the shared VCM address space. Protection is based on the application's ownership of the memory pages in the VCM address space.

A DVCM can be viewed as a collection of VCMs using an efficient messaging layer to communicate with each other. In effect, the DVCM is designed to act like an "active backplane" for the components of a parallel or high-performance distributed application.

Similar to the VCM, the DVCM is designed to run on coprocessor-enabled network interfaces for high-speed networks such as ATM, Myrinet, and Fast/Gigabit Ethernet.

In contrast to the VCM, the DVCM is an abstraction of the entire cluster network. It consists of multiple execution units - one for each intelligent NI in the cluster. Similar to the VCM, applications interact with the local execution unit through shared memory. Execution units extend the capabilities of the VCM as they can execute commands issued not only by local applications, but also by remote applications and other execution units. Furthermore, the DVCM has a distributed address space, which is the union of the address spaces of all individual VCMs. Remote memory operations on this address space can be implemented using DVCM extension modules.

The DVCM implementation is based on an internal reliable messaging layer, which is used by its execution units to communicate with each other. These messages, called *control messages*, are generally small (less than 100 bytes) and optimized for low latency. Reliability of the control messages is implemented using a sliding window protocol between any pair of execution units.

The default services provided by DVCM to applications comprise a set of low-overhead, zero-copy, unreliable messaging primitives. These primitives can be used for assembling/disassembling network messages directly from/into the data structures of each local application component. The DVCM protection model is based on capabilities.

The distinguishing characteristic of the DVCM is its extensibility: the DVCM "backplane" can be extended with modules that implement application-specific functionality. These extensions are typically distributed across all of the DVCM's execution units. The DVCM extension modules can use the DVCM's framework primitives: control messages, timeout, and dynamic memory allocation services.

3. The DVCM Extension Model

Research on configurable operating and communication systems [2, 7, 19, 26] has demonstrated the fact that a single set of system primitives cannot easily satisfy the requirements of every user-provided application program. One of the proposed solutions is to customize the application interaction with the network interface [8, 16, 17]. The research reported here goes further and considers customizing the interaction of the application with *the entire network*, viewed as a single entity.

The DVCM architecture provides two mechanisms with which applications can customize their interactions with the network. First, applications may construct custom DVCM programs (i.e., instruction sequences) and then signal the interface to execute them. Second, applications can extend the DVCM firmware with code implementing application-specific commands that can be executed both locally and remotely. Both of these mechanisms are designed to help reduce communication overheads. The first mechanism is very similar to the one in the VCM architecture and it is described in [16, 17]. In the following, we describe the second mechanism.

Extending the DVCM with application-specific commands enables the transfer of application-level functionality to the network interface. Although this approach moves processing from a faster to a slower processor, there are arguments that support it:

Latency. The DVCM's execution units are "much closer" to each other than the components of the cluster application because latencies between coprocessors are always smaller than latencies between application components – on our ATM LAN, coprocessor-to-coprocessor latencies are $\approx 18\mu s$ while application-to-application latencies are $\approx 60\mu s$.

Host CPU Overhead. Workstation CPUs are neither interrupted nor do they need to synchronize with each other when collective computations are implemented by network coprocessors – on our UltraSparc workstations the interrupt overhead is $\approx 9\mu s$ when serviced in a kernel handler and $\approx 50\mu s$ when serviced in a signal handler.

A typical DVCM extension module consists of a collection of event handlers and state variables, and it uses the timeout and the dynamic memory management services provided by the DVCM framework. The event handlers can use two DVCM timeout services, which implement distinct tradeoffs between accuracy and overhead. State variables are used by the handlers to store information between consecutive handler invocations, and to exchange information between extension and host application(s). Memory for the state variables can be allocated either statically, when the extension module is added to the DVCM framework, or dynamically, using a heap mechanism provided by the framework.

An extension module adds to the DVCM framework one or more commands and types of control messages. Towards this end, the extension registers with the DVCM framework one or more of the following event handler types:

Application-specific command handlers, which are invoked when a DVCM command implemented by the extension is executed as part of a program.

Control message handlers, which are invoked when a specific control message is received. Control messages are generally used to trigger a computation in a remote DVCM execution unit. Parameters required for handler execution are typically included in the message.

Timeout handlers, which are invoked when extension-specific timers expire. DVCM extension modules can use the DVCM timeout services to handle network unreliability and to implement repetitive executions with no involvement of the local application component.

Any DVCM extension module includes an initialization procedure that registers the command and message handlers of the extension. The same procedure can also register an initial set of timeout handlers. A deactivation procedure is included, as well.

Although all of our implementations provide only for static (compile-time) extensions of the DVCM, a more complex implementation can provide for dynamic extensions of DVCM functionality. Based on the results described in [7, 25], we believe that the DVCM extension model can provide for safe dynamic extensions of the base DVCM.

Furthermore, the DVCM extension model can be combined with the extension model of a host kernel to provide a framework useful for implementing efficient cluster-wide mechanisms such as scheduling in a manner similar to [21].

4. Sender Coordination

Performance degradation due to network overload is a significant challenge for parallel computations on COWs. In this section, we describe our DVCM-based solution for preventing network overload when running parallel/distributed applications in ATM-based COWs.

4.1. Motivation

The advantages of using COWs for parallel and high-performance distributed computing are well known and documented [1]. However, significant challenges arise when the traffic loads generated by parallel applications do not match the traffic models supported by the COW's underlying network. These mismatches can lead to network overload and, consequently, to dramatic reductions in message throughput. In general, COWs built around unreliable networks and off-the-shelf switches are not able to handle hot spots in the traffic generated by a parallel computation.

In particular, ATM-based COWs are promising solutions for high-performance distributed computing on metropolitan and wide-area clusters [23] due to their availability, scalability and to the continuing improvements in their cost/performance ratios. However, ATM-based networks are hard to use for parallel computing because the traffic patterns that can be enforced by ATM switches rarely match those of parallel applications and it is prohibitively slow to change the connections' parameters on demand. Application-level solutions are prone to exhibit low performance because of multiple factors, including: (1) the relatively high latencies of ATM switches, (2) the lack of a common clock or special purpose hardware for node synchronization, and (3) the lack of support for parallel scheduling in the operating systems typically run on cluster nodes.

We prove that using the DVCM architecture, in particular, the DVCM extension capability, enables effective parallel computation over ATM networks. More specifically, we design and implement a DVCM extension that prevents network overload by synchronizing the transmissions from multiple cluster nodes to a single destination.

Our approach places minimal requirements on network switches, and it assumes that the coprocessor on the host ATM interface is dedicated to the DVCM firmware. Hosts are required to be equipped with a second, possibly slower, network interface or to have the operating system kernel modified to use the DVCM for handling ordinary network traffic. We only assume Unspecified Bit Rate (UBR) connections on our ATM switches. This choice is motivated by the high cost of establishing ATM connections on demand and the bursty traffic characterizing parallel applications.

4.2. Sender Coordination as a DVCM Extension Module

The current ATM-based implementation of the DVCM architecture runs on a cluster of UltraSPARC-I machines connected to a ForeRunner ASX-200WG switch (the network has a star topology). Each of the workstations has one FORE SBA-200E (ATM) card, running the DVCM firmware. The implementation uses AAL5 packets for both application-level and control messages. In addition, each host is equipped with a Fast Ethernet network interface used for ordinary network traffic.

This section describes a DVCM extension module designed to prevent overloading the cluster ATM network. The extension implements DVCM-level flow control with a simple token-based protocol. Toward this end, the extension provides handlers for a new command, called SEND_WITH_FLOW_CONTROL, and four types of control messages, which are GRANT_LINK, UNI_REQ_LINK, BCAST_REQ_LINK, and OWN_LINK_HINT. The extension does not use timeouts. The current version consists of approximately 500 lines of C code. The algorithm implemented by the DVCM extension is presented next.

A token is associated with each network link. A source can send a message only if it holds all of the tokens for the links between itself and its destination. In our ATM network, with a star topology and full duplex links, in order to send, the source is only required to hold the token for the link between the switch and the destination node.

If the source does not hold the token, it requests the token either from the suspected holder or it broadcasts a request. Upon receiving a token request, the suspected holder sends the token to the requester provided it still holds it and it is not currently sending a message; otherwise, it broadcasts the request on behalf of the requester. Upon receiving a broadcast request, the holder sends the token to the requester provided it is not currently sending a message; the other nodes register the request. In either way, the current holder does not relinquish the token until it completes the current transmission. When transmission is completed, if there is no registered request, the token holder broadcasts that it holds the token. This "hint" message permits all other nodes to consider its sender as the suspected holder.

UNI_REQ_LINK or BCAST_REQ_LINK messages are used to request the token. GRANT_LINK messages are used to pass the token and OWN_LINK_HINT messages are used to disseminate the token location.

When contention for a given destination exists, we expect the destination adapter and the link towards it to be relatively busy. Therefore, the algorithm is designed such that it involves communication only between the source adapters, thereby not involving the destination adapter.

The algorithm can be extended to deal with more complex network topologies. For instance, if a message has to traverse more than two links, the source might need to acquire multiple tokens before sending. Similarly, in MANs and WANs, sources should coordinate before using links in the public domain.

When no contention is present, the coordination mechanism adds up to the duration of a control message roundtrip to the latency of each message. While this should not be a problem for long messages, it can be a very poor choice for short or very short messages. This is because, in general, the achievable outgoing data rate is low and the network cannot be overloaded when using small messages.

Therefore, the transmission of small messages should probably not be controlled by the flow control algorithm. Even for large messages but small data bursts, the buffering capabilities on the switch can substitute for the lack of flow control. The maximum message and burst sizes for which messages can be sent without prior coordination among sources depend on the buffering capabilities of the network switch(es) in the cluster.

Methods for reducing the latency of token passing include: (1) sending some cells before the token is received and (2) releasing token ownership before the last cell of a message is sent. The first method is a form of a more general technique called sub-datagram flow control that was explored in [3]. We have experimented with the second method and the experimental results are presented in the next subsection.

Token-based protocols may also be used to handle other shared resources. For instance, senders may allocate buffer space at the destination using a token carrying the necessary information. Buffers are added to the token by the receiver node and consumed by sender nodes. Similar to the management of link bandwidth, DVCM-level buffer management adds no overhead to the workstation CPUs in the cluster.

4.3. Performance

To determine the effectiveness of the sender coordination mechanism, we measured the bandwidth received by a node when one or more sources are simultaneously sending data to it. In most parallel applications, the number of hosts sending data to the same destination is generally independent of the size of the cluster and is determined by the problem decomposition. For instance, in applications using grids for domain decomposition, there are typically four or six competing data sources. Although parallel applications schedule their communication to avoid network overloading, contention between two or three of these sources is likely to exist, as documented in numerous experimental evaluations on distributed memory machines [10, 20].

Consequently, we experiment with one, two, and three data sources. The message sizes are varied from zero to 10 Kbytes and the data burst sizes are chosen to exceed the buffering capacity of the ATM switch.

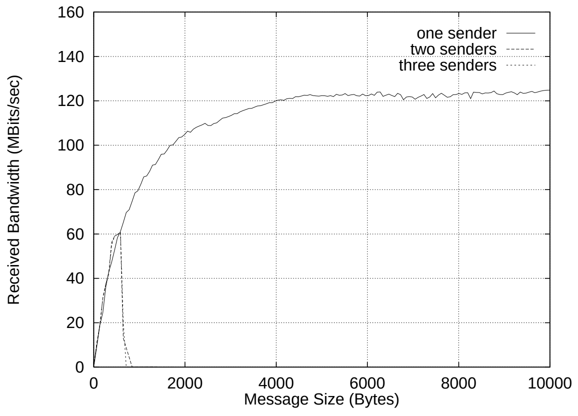


Figure 2. Receiver Bandwidth without Flow Control

Figure 2 presents the receive bandwidth when there is no coordination among sources. For messages larger than $\approx$ 600 bytes, receive bandwidth drops almost to zero, for both two and three simultaneous senders. Performance results for connections with and without the Early Packet Discard[1] feature enabled are very similar.

The benefits of using the flow control module to coordinate senders are presented in Figure 3. In all three experiments, the message loss rate is under .1%, and the receive

[1]Switches implementing the Early Packet Discard mechanism drop all the remaining cells in an AAL5 packet once they drop one of the cells in the packet.

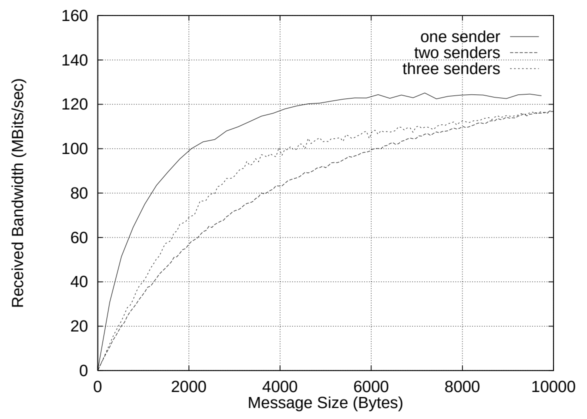


Figure 3. Receiver Bandwidth with Flow Control

bandwidth is a large fraction of the achievable link bandwidth. The overhead induced by the coordination mechanism is very small. This is shown by the one-sender experiments in which there is no contention and the receive bandwidth is only slightly smaller in the experiment with flow control (Figure 3) than without flow control (Figure 2).

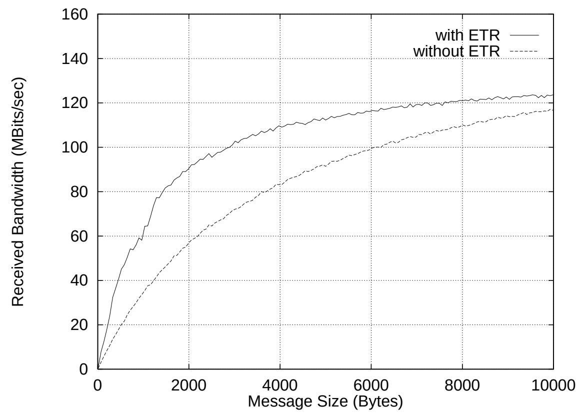


Figure 4. Receiver Bandwidth with two senders: with and without Early Token Release (ETR)

As previously mentioned, the performance enabled by the flow control module can be improved by releasing the token before the last cell in a message is sent (Early Token Release – ETR). Figure 4 shows the effects of ETR in an experiment with two contending sources. The token is released ten cells before the message transmission is completed; this number is determined by the token passing latency and the maximum cell transmission rate.

Note that ETR significantly improves receive bandwidth. When ETR is used, the receive bandwidth for two contending senders is close to the bandwidth achieved under no contention, i.e., with only one sender. In addition, we did not observe any message loss induced by ETR. Therefore, we are successful in hiding (at least a large fraction of) the token passing latency.

5. Related Work

Over the past several years there has been considerable interest in building faster and lighter-weight communication architectures for COWs based on off-the-shelf components. Most of these systems offer a set of well-optimized user-level messaging primitives [4, 5, 14, 24]. Other systems implement an efficient RPC mechanism [22] or an efficient distributed shared memory abstraction [6]. In contrast, the DVCM enables the integration of application-specific modules into a framework that provides only a small set of basic primitives.

The benefits of extending and configuring the communication substrate have been studied for a wide variety of platforms, ranging from coherence protocols for distributed memory parallel machines to environments for high-performance computing over wide-area networks.

The Stanford FLASH project [11] investigates the advantages of a flexible communication layer by replacing the traditional hardwired controller of a distributed parallel machine with a programmable microcontroller. Detailed simulations show that the cost of adding flexibility is small, as a set of optimized parallel applications running on the same distributed memory machine with a hardwired controller and with a programmable node controller exhibit very similar performance. In comparison, the DVCM architecture is a lower-cost solution to providing improved performance to parallel applications running on COWs.

The Globus toolkit [9] provides developers with the ability to configure the communication infrastructure used by high-performance applications and tools running across wide-area networks. DVCM and Globus target different classes of parallel applications and while the focus of DVCM is on the providing an extensible communication substrate for COWs, the Globus project is concentrating on providing a configurable execution environment across wide-area networks.

Other projects have identified the potential benefits of running communication-intensive modules of a cluster application in a "closer-to-the-network" context. [12] considers the decomposition of collective computation algorithms into "core", performance-critical operations, implemented on dedicated NI resources, and more complex but less frequently invoked operations, implemented on the workstations in the cluster. Simulation results show that important performance improvements are possible when NI resources are used for at least three collective operations: reliable multicast, symmetric broadcast, and barrier synchronization. In contrast, the DVCM infrastructure is designed to accommodate a larger set of performance-critical operations on the NI, i.e., any application-specific operation that can be implemented in the DVCM extension model.

Decomposing cluster applications into Application-Specific Handlers (ASHs), run in kernel context in response to a message arrival, and the rest of the application, run in user context, is explored in [26]. Although this work does not consider the NI resources, it takes an approach similar to the DVCM architecture. The message handlers that execute in kernel context can be assimilated with a second layer in a computational resource hierarchy: their execution is always bounded and the latencies between remote handlers are smaller than the latencies between user-level modules. Experiments show that ASHs enable better latency and throughput in a COW based on an ATM LAN.

6. Conclusion

This paper introduces the DVCM extensible communication architecture for COWs. The DVCM is a software/firmware-only architecture, designed to build a moderate-cost parallel computing platform as a tightly-coupled COW. Towards this end, the DVCM design takes advantage of the availability of low-cost intelligent network interfaces. Motivated by our belief that there is no best set of communication primitives for cluster computing, we designed the DVCM as a communication framework that can be extended with application-specific modules. The benefits of the DVCM design are illustrated with an extension module that implements an efficient coordination mechanism between the components of a cluster application, a task needed in many such applications. Experiments on an ATM-based COW show that it is possible to achieve negligible message loss rates and high link bandwidth utilization under significant link contention when using the coordination DVCM extension module.

Decomposing a cluster application into modules running on the NI coprocessor and on the workstation CPU might appear as a difficult programming job. However, constraining the extension modules to a collection of event handlers, as in the DVCM extension model, makes the programmer's job easier. Alternatively, this decomposition can fit into the multi-threaded programming model, with one or more threads running on the communication coprocessor. Having the parallel application component and the local DVCM execution unit communicate through shared memory enables this programming model.

In the future, the cost-performance ratio for COWs will continue to decrease, rendering them even more attractive for parallel computing. This is because the speed of the workstation's CPUs and the interconnect's bandwidth will continue to increase, driven by technological advances and by end-user demands, while their widespread deployment will keep their price low.

Unfortunately, it is unlikely that future workstations will be equipped with I/O architectures that better match the capabilities of existing and future interconnects, such as memory-integrated network interfaces. This is mainly because applications run by the average workstation user are almost always characterized by low or moderate message rates. Inter-node latencies are expected to remain an order of magnitude or more larger in COWs than in comparable[2] parallel machines for similar reasons: few if any of the applications in wide use can benefit significantly from LAN latencies on the order of a few microseconds or less.

It is expected that the capacity of the resources installed on intelligent NI cards will increase, driven by technology advances. More specifically, the speed of communication processors, which has already reached 100MHz, will continue to increase. Current trends of placing both processor and memory on the chip [13, 15, 18] are expected to decrease the cost of intelligent NIs, and therefore, to diminish the cost difference between intelligent and ordinary NIs. Consequently, we believe that it is reasonable to expect that most future adapters for high speed networks will include the resources needed by architectures similar to the DVCM.

In this context, extensible cluster communication architectures, like the DVCM, are expected to be the most cost-effective way of building closely-coupled parallel computing platforms as clusters of workstations.

References

[1] T. E. Anderson, D. E. Culler, D. A. Patterson, and the NOW Team. A Case for Networks of Workstations: NOW. *IEEE Micro*, Feb. 1995.

[2] Bershad, B. and Savage, S. and Pardyak, P. and Sirer, E. and Fiuczynski, M. and Becker, D. and Chambers, C. and Eggers, S. Extensibility, safety, and performance in the SPIN operating system. *Proceedings of the 15th ACM Symposium on Operating System Principles, ACM SIGOPS Notices*, Dec. 1995.

[3] J. C. Brustuloni. Exposed Buffering and Sub-Datagram Flow Control for ATM LANs. *Proceedings of the 19th Conference on Local Computer Networks*, Oct. 1994.

[4] G. Buzzard, D. Jacobson, M. Mackey, S. Marovich, and J. Wilkes. An implementation of the Hamlyn sender-managed interface architecture. *Proceedings of the 2nd Symposium on Operating Systems Design and Implementations*, Oct. 1996.

[5] P. Druschel, L. L. Peterson, and B. S. Davie. Experiences with a High-Speed Network Adaptor: A Software Perspective. *Proceedings SIGCOMM '94*, Aug. 1994.

[2]COWs and parallel machines are considered *comparable* when they use same generation hardware technologies.

[6] C. Dubnicki, A. Bilas, Y. Chen, S. Damianakis, and K. Li. VMMC-2: Efficient Support for Reliable, Connection-Oriented Communication. *Hot Interconnects V*, Aug. 1997.

[7] D. Engler, M. Kaashoek, and J. O'Toole. Exokernel: an operating system architecture for application-level resource management. *Proceedings of the Fifteenth ACM Symposium on Operating System Principles, ACM SIGOPS Notices*, Dec. 1995.

[8] M. E. Fiuczynski and B. N. Bershad. SPINE - A Safe Programmable and Integrated Network Environment. *Proceedings of the Work In Progress session of the 16th ACM Symposium on Operating Systems Principles*, Oct. 1997.

[9] I. Foster and C. Kesselman. The Globus Project: A Status Report. *IPPS/SPDP '98 Heterogeneous Computing Workshop*, Apr. 1998.

[10] G. C. Fox, M. A. Johnson, G. A. Lyzenga, S. W. Otto, J. K. Salmon, and D. W. Walker. *Solving Problems On Concurrent Processors*. Prentice-Hall, 1988.

[11] M. Heinrich, J. Kuskin, D. Ofelt, J. Heinlein, J. Baxter, J. P. Singh, R. Simoni, K. Gharachorloo, D. Nakahira, M. Horowitz, A. Gupta, M. Rosenblum, , and J. Hennessy. The Performance Impact of Flexibility in the Stanford FLASH Multiprocessor. *Proceedings of the 6th International Conference on Architectural Support for Programming Languages and Operating Systems*, pages 274–285, Oct. 1994.

[12] Y. Huang and P. K. McKinley. Efficient Collective Operations with ATM Network Interface Support. *Proceedings of the 1996 International Conference on Parallel Processing*, pages 34–43, Aug. 1996.

[13] P. M. Kogge. EXECUBE – A New Architecture for Scalable MPPs. *Proceedings of the 1994 International Conference on Parallel Processing*, 1994.

[14] S. Pakin, M. Laura, and A. Chien. High Performance Messaging on Workstations: Illinois Fast Messages (FM) for Myrinet. *Supercomputing*, Dec. 1995.

[15] D. Patterson, T. Anderson, N. Cardwell, R. Fromm, K. Keeton, C. Kozyrakis, R. Thomas, and K. Yelick. A Case for Intelligent RAM: IRAM. *IEEE Micro*, Apr. 1997.

[16] M.-C. Roşu, K. Schwan, and R. Fujimoto. Supporting Parallel Applications on Clusters of Workstations: The Intelligent Network Interface Approach. *Proceedings of the 6th IEEE International Symposium on High Performance Distributed Computing*, Aug. 1997.

[17] M.-C. Roşu, K. Schwan, and R. Fujimoto. Supporting Parallel Applications on Clusters of Workstations: The *Virtual Communication Machine*-based Architecture. *Cluster Computing*, 1:1–17, Jan. 1998.

[18] A. Saulsbury, F. Pong, and A. Nowatzyk. Missing the Memory Wall: The Case for Processor/Memory Integration. *Proceedings of the 23rd Annual International Symposium on Computer Architecture*, Jun. 1996.

[19] K. Schwan, T. Bihari, B. W. Weide, and G. Taulbee. High-Performance Operating System Primitives for Robotics and Real-Time Control Systems. *ACM Transactions on Computer Systems*, 5(3):189–231, Aug. 1987.

[20] K. Schwan and W. Bo. Topologies – Distributed Objects on Multicomputers. *ACM Transactions on Computer Systems*, 8(2):111–157, May 1990.

[21] P. G. Sobalvarro, S. Pakin, W. E. Weihl, and A. A. Chien. Dynamic coscheduling on workstation clusters. *Submitted for Publication*, March 1997.

[22] C. A. Thekkath and H. M. Levy. Limits to Low-Latency Communication on High-Speed Networks. *ACM Transactions on Computer Systems*, 11(2):179–203, May 1993.

[23] H. Topcuoglu, S. Hariri, W. Furmanski, J. Valente, I. Ra, D. Kim, Y. Kim, X. Bing, and B. Ye. The Software Architecture of a Virtual Distributed Computing Environment. *Proceedings of the 6th IEEE International Symposium on High Performance Distributed Computing*, pages 40–49, Aug. 1997.

[24] T. von Eicken, A. Basu, V. Buch, and W. Vogels. U-Net: A User-Level Network Interface for Parallel and Distributed Computing. *Proceedings of the 15th ACM Symposium on Operating Systems Principles*, Dec. 1995.

[25] R. Wahbe, S. Lucco, T. E. Anderson, and S. L. Graham. Efficient Software-Based Fault Isolation. *Proceedings of the 14th ACM Symposium on Operating System Principles*, pages 203–216, Dec. 1993.

[26] D. A. Wallach, D. R. Engler, and M. F. Kaashoek. ASHs: Application-specific handlers for high-performance messaging. *Proceedings of ACM SIGCOMM'96 Conference on Applications, Technologies, Architectures and Protocols for Computer Communication*, pages 40–52, Aug. 1996.