An Adaptive Message-Passing System for ATM-Based Wide-Area Clusters

Sung-Yong Park
Department of Computer Science
Sogang University
Email: parksy@ccs.sogang.ac.kr

Abstract

In this paper we present an architecture, implementation, and performance evaluation of an adaptive communication system (ACS) for wide-area ATM clusters. Our approach capitalizes on thread-based programming model to overlap computation and communication and develop a dynamic message-passing environment with separate data and control paths. This leads to a flexible and adaptive message-passing environment that can support multiple flow control, error control, and multicasting algorithms. We show the performance of ACS applications and compare them with p4, PVM, and MPI applications.

1 Introduction

With the advances in processor technology and high-speed networking technologies such as Asynchronous Transfer Mode (ATM) [1], Myrinet [2], Fast Ethernet [3], and High Performance Parallel Interface (HIPPI) [4], workstation clusters interconnected by high-speed networks have evolved as a feasible and cost-effective environment for implementing large-scale high-performance distributed computing (HPDC) applications [5, 6, 7, 8, 9, 10]. One of the important characteristics of HPDC applications is that they have a wide range of communication requirements (e.g., latency and bandwidth requirement, flow control scheme, error control scheme, multicasting scheme) and even one single application has multiple communication requirements during the course of its execution (e.g., interactive multimedia applications). Furthermore, the distributed execution environment is usually built on top of the heterogeneity of computing resources/underlying networks, and changes dynamically. In order to satisfy these needs, the communication systems for this environment should be highly portable and provide adaptive communications and protocol functions so that users can dynamically select appropriate functions at runtime to meet the requirements of a given application.

In this paper we present an architecture, implementation, and performance evaluation of an adaptive message-passing system for a heterogeneous wide-area ATM cluster that we call the adaptive communication system (ACS). ACS uses multithreading to provide efficient techniques for overlapping computation and communication in wide-area computing. By separating control and data activities, ACS eliminates unnecessary control transfers. This optimizes the data path and improves the performance. ACS supports several different flow control, error control, and multicasting algorithms and allows programmers to select at runtime the suitable communication schemes per-connection basis. ACS provides three application communication interfaces such as socket communication interface (SCI), ATM communication interface (ACI), and high-performance interface (HPI) to support various classes of applications.

We present the performance of ACS applications and compare them with those of other message-passing tools such as p4 [13], PVM [14], and MPI [15]. The benchmark results show that ACS outperforms other message-passing tools.

The rest of the paper is organized as follows. Section 2 presents the overview of ACS architecture. Section 3 discusses the software architecture of ACS. Section 4 analyzes and compares the performance of ACS applications. Section 5 contains the summary and conclusion.

2 Overview of ACS Architecture

Figure 1 shows the general architecture of ACS. An ACS application consists of multiple *Compute_Threads* that include programs to perform the computations of the application. ACS supports both the *host-node* programming model and the Single Program Multi-

ple Data (SPMD) programming model. In both models processes are created at each node by using the *hostfile* that specifies the initial configurations of machines to run ACS applications. After each process is spawned, it creates multiple *Compute_Threads* according to the computation requirements of the application. The advantage of using a thread-based programming paradigm is that it reduces the cost of context switching, provides efficient support for fine-grained applications, and allows the overlapping of computation and communication.

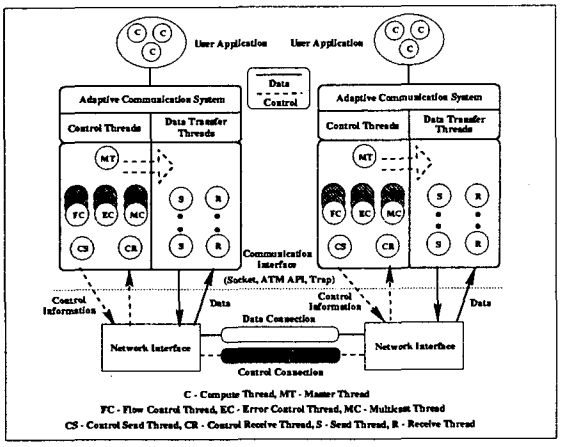

Figure 1: ACS General Architecture

ACS separates control and data functions by providing two planes (see Figure 1): a *control plane* and a *data plane*. The control plane consists of several threads that implement important control functions (e.g., connection management, flow control, error control) in an independent manner. These threads include *Master_Thread*, *Flow_Control_Thread*, *Error_Control_Thread*, *Multicast_Thread*, *Control_Send_Thread*, and *Control_Receive_Thread* (we call them *control threads*). The *data transfer threads* (*Send_Thread* and *Receive_Thread*) in the data plane are spawned on a per-connection basis by the *Master_Thread* to perform only the data transfers associated with a specific connection. Furthermore, the control and data information from the two planes are transmitted on separate connections. All control information (e.g., flow control, error control, configuration information) is transferred over the control connections, while the data connections are used only for the data transfer functions. The separation of control and data functions eliminates the process of demultiplexing control and data packets within a single connection and allows the concurrent processing of control and data functions. This allows applications to utilize all available bandwidth for the data transfer functions and thus improves the performance.

ACS supports multiple flow-control (e.g., window-based, credit-based, or rate-based), error-control (e.g., go-back N or selective repeat), and multicasting algorithms (e.g., repetitive send/receive or a multicast spanning tree) within the control plane to meet the Quality of Service (QoS) requirements of a wide range of HPDC applications. Each algorithm is implemented as a thread and programmers activate the appropriate thread when establishing a connection to meet the requirements of a given connection.

ACS provides three application communication interfaces such as *socket* communication interface (SCI), ATM communication interface (ACI), and high-performance interface (HPI) in order to support HPDC applications with different communication requirements. The SCI is provided mainly for applications that must be portable to many different computing platforms. The ACI provides the services that are compatible with ATM connection-oriented services where each connection can be configured to meet the QoS requirements of that connection. The HPI supports applications that demand low-latency and high-throughput communication services.

Additional details about ACS architecture can be found in [11, 12].

3 ACS Software Architecture

The software architecture of ACS can be described in terms of three main modules (see Figure 2): *ACS Application Programming Interface* (ACS_API), *ACS MultiThread Subsystem* (ACS_MTS) and *ACS Message Passing Subsystem* (ACS_MPS).

The ACS_API provides a common programming interface to support multithreading and communication services for parallel/distributed applications. The ACS_MTS provides thread-related services such as thread management and thread synchronization functions. The ACS_MPS provides point-to-point communication, group communication, and synchronization services required by parallel/distributed applications. Each subsystem is clearly divided into system-dependent part and system-independent part. All performance-critical parts are implemented in the system-dependent part using machine or operating

system specific features. This modular architecture allows ACS to provide high portability for the upper software layers and to achieve good performance by benefiting from machine-specific features.

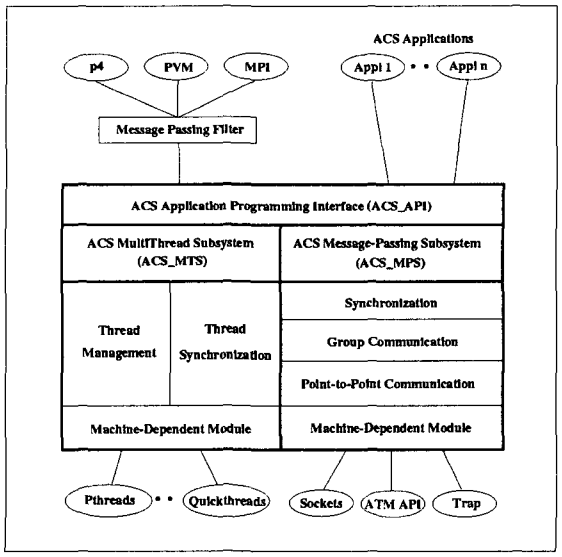

Figure 2: ACS Software Architecture

The support for existing applications written using existing message-passing systems (e.g., p4, PVM, MPI) is achieved by *Message Passing Filter* (MPF), which will be developed as a middle-ware between message-passing systems and ACS. The MPF provides the portability and heterogeneity support for existing applications and message-passing systems.

4 Experimental Results

In this section we evaluate the performance of four message-passing tools (ACS, p4, PVM, and MPI) by comparing the execution time of four applications (i.e., Fast Fourier Transform (FFT), Joint Photographic Experts Group (JPEG) Compression/Decompression, Parallel Sorting with Regular Sampling (PSRS), Back-Propagation Neural Network (BPNN) Learning) that are commonly used in parallel and distributed systems.

All experiments have been conducted over two different computing platforms (six SUN-4 workstations running SunOS 5.5 and four IBM/RS6000 workstations running AIX 4.1) interconnected by an ATM network. In all measurements, we used the ACS version implemented over *socket* interface with tree-based multicasting primitives. For the PVM (Version 3.3.11) benchmarking, we used the PVM Direct mode, where the direct TCP connection is made between two endpoints. The MPICH [16] (Version 1.0.13) and p4 (Version 1.4) were used to evaluate the performance of MPI and p4 applications.

Figure 3 shows the performance comparison of each message-passing tool, using four applications running over eight heterogeneous workstations (e.g., four SUN-4 workstations and four IBM RS6000 workstations) interconnected by an ATM network.

As we can see from Figure 3, ACS applications outperformed other implementations. For the applications that require many communications with small messages (e.g., FFT), the performance improvement is modest, while for the applications with a large amount of data exchange, the performance improvement is bigger (e.g, JPEG, PSRS). Furthermore, for the applications where a lot of broadcasting with a large amount of data is performed (e.g., BPNN), ACS shows outstanding performance. We believe that most of the improvements of ACS in this case are due to the overlapping of communications and computations and the tree-based broadcasting primitive.

While benchmarking those four applications, we learned several lessons. The first is in regard to the effects of the connection management scheme. In p4, if the message should be transmitted over a TCP/IP network, it first checks the *socket* connection to the destination process. If the connection is already open, it is used. Otherwise, a *socket* connection is first established at runtime before transmitting the message. For applications where the communication pattern is simple and very few communications take place during the execution of the applications, the time for connection set-up may affect the whole application performance if much time is required to set up the connection. For example, the performance of the p4 implementation in the JPEG application is ineffective in IBM/RS6000 platform, although the p4 has a good performance for both point-to-point communication and group communication primitives. The micro-benchmarking using the JPEG application over an IBM/RS6000 platform revealed that the p4 spent most of its time sending the first message to the destination process. This means that the performance of p4 applications may be degraded due to the time for a connection set-up in a particular application like JPEG.

The second lesson was that multithreaded message-passing tools such as ACS do not perform well for

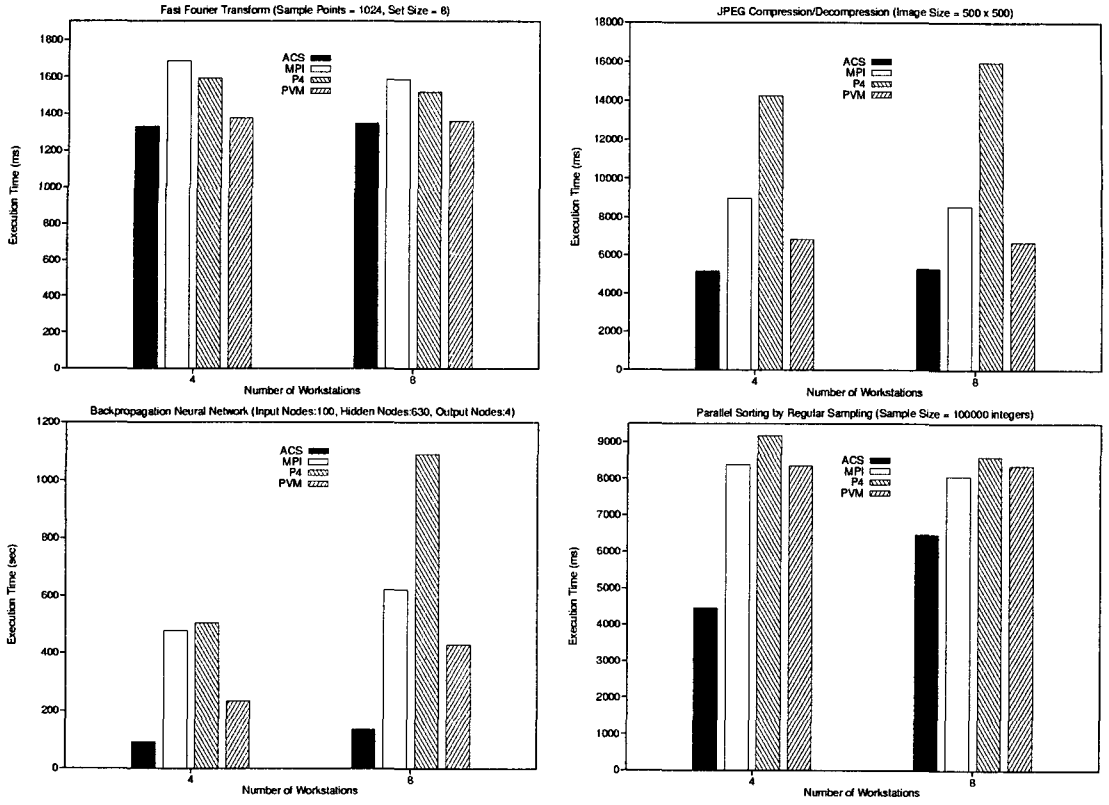

Figure 3: Application Performance over ATM Using Eight Heterogeneous Workstations

some application domains. For highly synchronous and SPMD-style applications with regular communication patterns, most of the processes participating in the computations are blocked for receiving the results from the other processes. By using multithreading techniques we can modify the structure of the application program such that some threads continue their computations while one thread is waiting for receiving data, which reduce the total execution time. However, if the data size is small, the overhead of using multithreading can be greater than the benefits obtained by overlapping computations with communications, and thus the multithreaded message-passing tools degrade the performance of such applications. This means that we cannot benefit from the use of threads for the highly synchronous parallel/distributed applications with small data exchanges.

5 Conclusion

In this paper we have presented an adaptive message-passing system for an ATM-based wide-area HPDC environment that can meet the QoS requirements of a wide range of HPDC applications. ACS utilizes the thread-based programming model to overlap computation and communication. ACS provides multiple communication algorithms (e.g., flow control, error control, and multicasting algorithms) and communication interfaces (SCI, ACI, HPI), and allows programmers to select appropriate communication algorithms and interfaces at runtime. This flexible and adaptive environment enables users to efficiently develop a wide variety of HPDC applications with different QoS requirements.

We have evaluated the performance of ACS applications. The benchmarking results showed that ACS outperformed other message-passing systems.

References

[1] J. Y. Le Boudec, "The Asynchronous Transfer Mode: a Tutorial," Computer Networks and ISDN Systems, Vol. 24, No. 4, pp. 279–309, 1992.

[2] N. J. Moden, D. Cohen, R. E. Felderman, A. E. Kulawik, C. L. Seitz, J. N. Seizovic, and W. Su, "Myrinet: A Gigabit-per-Second Local Area Net-

work," IEEE Micro, Vol. 15, No. 1, pp. 29–36, February 1995.

[3] IEEE Std 802.3u, "Local and Metropolitan Area Networks: Media Access Control (MAC) Parameters, Physical Layer, Medium Attachment Units, and Repeater for 100 Mb/s Operation, Type 100BASE-T," 1995.

[4] D. Tolmie, and J. Renwick, "HIPPI: Simplicity Yields Success," IEEE Network, pp. 28–32, January 1993.

[5] T. E. Anderson, D. Culler, and D. A. Patterson, "A Case for NOW (Network of Workstations)", IEEE Micro, pp. 54–64, February 1995.

[6] T. Sterling, D. Becker, D. F. Savarese, et al., "BEOWULF: A Parallel Workstation for Scientific Computation", Proc. of International Conference on Parallel Processing, 1995.

[7] E. Arnold, F. Bitz, E. Cooper, H. T. Kung, R. Sansom, and P. Steenkiste, "The Design of Nectar: A Network Backplane for Heterogeneous Multicomputers," Proc. of the 3rd International Conference on Architectural Support for Programming Languages and Operating Systems, pp. 205–216, April 1989.

[8] E. W. Felton, R. D. Alpert, A. Bilas, M. A. Blumrich, D. W. Clark, S. N. Damianakis, C. Dubnicki, L. Iftode, and K. Li, "Early Experience with Message-Passing on the SHRIMP Multicomputer,", Proc. of the 23rd International Symposium on Computer Architecture, pp. 296–307, May 1996.

[9] T. M. Warschko, J. M. Blum, and W. F. Tichy, "The ParaStation Project: Using Workstations as Building Blocks for Parallel Computing," Proc. of the International Conference on Parallel and Distributed Processing, Techniques and Applications (PDPTA'96), Vol. pp. 375–386, August 1996.

[10] J. V. Lawton, J. J. Bronsnan, M. P. Doyle, S. D. O Riordain, and T. G. Reddin, "Building a High-performance Message-passing System for MEMORY CHANNEL Clusters," Digital Technical Journal, Vol. 8, No. 2, pp. 96–116, 1996.

[11] S. Y. Park and S. Hariri, "High Performance Message Passing System for Network of Workstations" Journal of Supercomputing, Vol. 11, No. 2, pp. 159–179, 1997.

[12] S. Y. Park and S. Hariri, "ACS: An Adaptive Communication System for Heterogeneous Wide-area ATM Clusters", To appear in Cluster Computing Journal, 1999.

[13] R. Butler and E. Lusk, "Monitors, message, and clusters: The p4 parallel programming system," Parallel Computing, Vol. 20, pp. 547–564, April 1994.

[14] V. S. Sunderam, "PVM: A Framework for Parallel Distributed Computing," Concurrency: Practice and Experience, Vol. 2, No. 4, pp. 315–340, December 1990.

[15] MPI Forum, "MPI: A Message Passing Interface," Proc. of Supercomputing '93, pp. 878–883, November 1993.

[16] B. Gropp, R. Lusk, T. Skjellum, and N. Doss, "Portable MPI Model Implementation," Argonne National Laboratory, July 1994.