Design and Implementation of Application-Level Multicasting Services over ATM Networks*

Sung-Yong Park[1], Jihoon Yang[1], and Yoonhee Kim[2]

[1] Department of Computer Science and Engineering
Sogang University, Seoul Korea
{parksy,jhyang}@ccs.sogang.ac.kr,
[2] Department of Computer Science
Sookmyung Women's University, Seoul Korea
yulan@cs.sookmyung.ac.kr

Abstract. The ACS (Adaptive Communication System) is a multithreaded message-passing system that provides application programmers with multithreading and flexible communication services. This paper outlines the general software architecture of ACS and describes how the ACS architecture is applied to implement its flexible application-level group communication services. We provide the performance results of ACS multicasting services and compare them with those of p4, PVM, and MPI.

1 Introduction

The Adaptive Communication System (ACS) [1] is a multithreaded message-passing system that provides users with multithreading and dynamic communication services (e.g., point-to-point and group communication services). The ACS 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 scalable message-passing environment that can support multiple communication algorithms (e.g., error control, flow control, multicasting algorithms) and interfaces at runtime. This paper primarily focuses on the *flexible*, *scalable*, and *application-level* group communication services provided by ACS.

The group communication services provided by current message-passing systems have several drawbacks. First of all, some message-passing systems (e.g., PVM) implement group communication operations (e.g., collective communication) by repeatedly calling send routines for portability, which is computationally expensive and not scalable for groups with large members. Although the tree-based multicasting operations can be implemented at the source level, this process is cumbersome and prone to errors. Second, their communication primitives are static, which means that they are not able to adapt to rapidly changing

* The publication of this paper was supported in part by Institute for Applied Science and Technology of Sogang University

M.H. Shafazand and A M. Tjoa (Eds.): EurAsia-ICT 2002, LNCS 2510, pp. 479–486, 2002.

network dynamics. The ability to adapt to varying network conditions is one of the important features that need to be supported in the communication systems, especially for wide-area computing. Third, in traditional message-passing systems, the transfer of control and data are usually tightly coupled. For a large number of small groups, this results in generating a large amount of control traffic associated with group operations and potentially decreases the performance of applications. There have been several distributed computing software systems specially designed to support group communication services such as Horus [5], Totem [6] and Transis [7]. However, most of them are designed to support special functionalities (e.g., fault tolerance, message ordering, virtual synchrony, group partition) rather than to provide high throughput or scalable group communication services. They fail to address the issues mentioned above.

The group communication services in ACS are based on the dynamic grouping. Each ACS process can dynamically create, join or leave a group during the lifetime of the process. The multicasting operation in ACS is implemented by using a spanning tree (e.g., binary tree) and this is more efficient than repetitive techniques for large group size. The multicasting tree is virtually created at the application level upon unicast connections using the application specific performance metric (e.g., topology, latency, or bandwidth). The ACS architecture which separates the data and control transfer allows the multicasting operations to be implemented efficiently by utilizing the control connections when transferring status information (e.g., membership change, acknowledgment to maintain reliability etc.). This separation optimizes the data path and thus improves the performance of ACS applications.

The rest of the paper is organized as follows. We begin by providing an overview of the ACS architecture in Section 2. Section 3 presents an implementation approach to the ACS group communication services. Section 4 compares the multicasting performance of ACS with those of other message-passing systems such as p4 [2], PVM [3], and MPI [4]. Section 5 contains the summary and conclusion.

2 Overview of ACS Architecture

ACS is a multithreaded message-passing system that provides multithreading (e.g., thread synchronization, thread management) and communication services (e.g., point-to-point communication, group communication) for High Performance Distributed Computing (HPDC) applications with different Quality of Service (QoS) requirements (See Figure 1). ACS uses multiple *Compute_Thread*s to implement the computations of HPDC applications. These threads use the ACS primitives to communicate and synchronize with other *Compute_Thread*s. This allows ACS to provide efficient support for fine-grained applications, and to reduce the propagation delay impact on HPDC applications especially in Wide Area Network (WAN)-based distributed computing by overlapping computation and communication.

ACS decouples the control and data paths by creating different threads for both control and data functions. Moreover, the control and data information from the two paths are transmitted on separate connections. The *control thread*s implement important control functions such as connection management, flow control, error control, and configuration management in an independent manner. The *data transfer thread*s are spawned based on a per-connection basis to perform only the data transfers associated with a specific connection. 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.

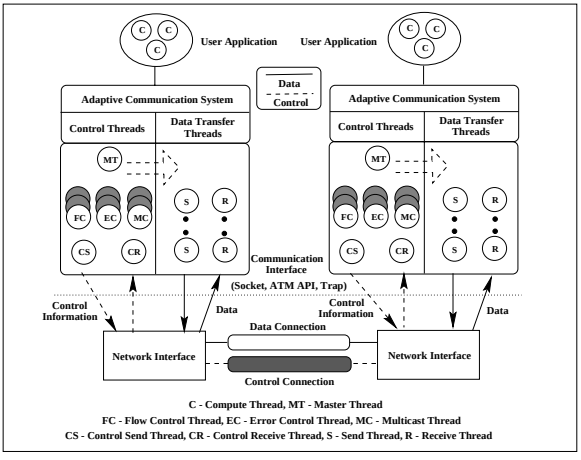

Fig. 1. ACS General Architecture

In ACS, 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) are provided as *control thread*s and programmers activate the appropriate thread when establishing a connection to meet the QoS requirements of a given connection. This allows the HPDC programmers to select for a given HPDC application the appropriate flow control, error control, and multicasting algorithms per-connection basis at runtime.

ACS is designed to support these classes of applications by offering three application communication interfaces: 1) Socket Communication Interface (SCI); 2) ATM Communication Interface (ACI); 3) High Performance Interface (HPI). The SCI is used mainly for providing high portability over a network of computers (e.g., workstations, PCs, parallel computers). The ACI is the application communication interface that allows programmers to access the inherent features of ATM network. The HPI is built to achieve high-throughput and low-latency inter-process communications.

3 ACS Group Communication Services

ACS group communication services support *dynamic* groups as shown in Figure 2. At program startup, a default ACS group, called *ACS_GRP*, is created and each ACS process in the *hostfile* joins this group automatically. The first process specified in the *hostfile* becomes a *Master Group Server* (MGS). Each process that creates a new group with a unique name becomes a *Local Group Server* (LGS) of that group by default. The MGS represents the whole LGSs and coordinates the group communication operations between these servers. The LGS is responsible for multicasting operations within the local group and maintains the membership information of the local group only. A *Global Multicasting Tree* (GMT) is built to connect all the LGSs rooted at the MGS. All the group members within the same group are connected by a *Local Multicasting Tree* (LMT) rooted at the LGS of that group.

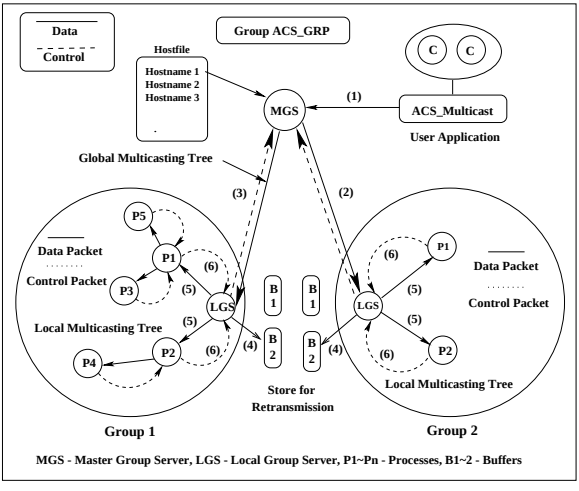

Fig. 2. Multicasting in the ACS Environment

The MGS and LGSs periodically exchange the status information of each group. By having a distributed group server in each group and making it to manage the local group only, the status traffic can be minimized and the information is managed with reliability. Each process that joins a particular group is identified by the *rank* within that group. Since ACS group communication primitives allow the overlapping of different groups, all ACS processes can join multiple groups at the same time.

When a group is created or destroyed, the MGS updates the GMT based on the application specific performance metric (e.g., topology, latency or bandwidth) and broadcasts the information to all the LGSs over the control connections. Each LGS updates the group information it is maintaining such as routing

information (e.g., left child, right child) and group server information (e.g., identifier of each group server) after receiving the information from the MGS. Each LGS in turn broadcasts the group server information to all the members of its group over the control connections. On the other hand, when a group member joins or leaves a particular group, the LGS of that group also updates the LMT and broadcasts the information to all the members of the same group only. Consequently, the information of the MGS and LGSs are visible to all the members within the global group *ACS_GRP* and the information of the group members are known only to the members within the same group. Therefore, the GMT and LMT are built before the multicasting operation is invoked. This reduces the setup time to perform multicasting operations and thus improves the performance. The ACS architecture that separates the control and data path allows us to implement this scheme efficiently by using the control path to transfer the membership changes without interfering with the data traffic.

3.1 Multicasting Protocol

ACS group communication primitives support *open* groups by providing three classes of multicasting operations: (1) *Global Broadcast*; (2) *Local Broadcast*; and (3) *Global Multicast*.

The *Global Broadcast* is used to transmit messages to the entire groups defined in the global group *ACS_GRP*. The *Local Broadcast* is used to transmit messages to the all members within the same group. The *Global Multicast* is used to transmit messages to the part of the entire groups. For all three operations, the destination end-point is not the members but the group server. The configuration information of the destination group (e.g., How many members are in the group, topology of the group) need not be visible to the multicasting process and the destination group server takes care of broadcasting the message to the its members. Keeping the state information for each member of all different groups is not efficient and generates a lot of broadcasting traffic whenever the membership status of a process is changed. In the *dynamic* group where a lot of membership changes are expected, the performance of the applications can be improved by reducing the traffic associated with the transfer of the status messages.

Since the three classes of multicasting operations are implemented using similar schemes, we will provide the algorithm for the *Global Broadcasting* only. The multicasting algorithm for the *Global Broadcasting* consists of six steps as shown in Figure 2 :

1. When the *Compute_Thread* of a process invokes the *ACS_mcast()* primitive, the *Multicast_Thread* of that process activates the corresponding *Send_Thread* to transmit an actual message to the MGS.
2. The MGS transmits the received message to the other LGSs using its GMT.
3. If the *ACS_mcast()* is invoked with reliable mode, each LGS that received the message sends an acknowledgment back to the MGS. The acknowledgment is merged along the GMT and the MGS should guarantee that all the LGSs receive the message.

4. A LGS maintains two buffers. The first buffer is used to assemble the messages, which are then transferred to the second buffer. The second buffer is used to retain the messages that have not been correctly received by group members.
5. Each LGS locally multicasts the message to its group members using its LMT.
6. If the *ACS_mcast()* is invoked with reliable mode, each member that received the message sends an acknowledgment back to the LGS. Again, the acknowledgment is merged along the LMT and the LGS should guarantee that all the members receive the message. If there is any group member which has not received a message within the timeout period, the LGS of the group retransmits the message again. This reduces the retransmission traffic from the source process.

4 Benchmarking Results

In this section, we analyze and compare the performance of ACS with those of other message-passing systems such as p4, PVM, and MPI in two levels: primitive performance level and application performance level using Back-Propagation Neural Network (BPNN) learning algorithm.

4.1 Primitive Performance

Figure 3 shows the performance of broadcasting primitives (e.g., *ACS_mcast()*, *pvm_mcast()*, *p4_broadcast()*, and *MPI_Bcast()*) of four message-passing systems over an ATM network for message sizes from 1 byte to 32 Kbytes. The group size varies from two to ten.

As we can see from Figure 3, ACS primitive (*ACS_mcast()*) shows the best performance for various message sizes and group sizes. Furthermore, *ACS_mcast()* primitive shows almost similar performance for large group sizes (over six members) as we increase the message size (over 4 Kbytes). In the *ACS_mcast()* primitive where most of the information for performing group communications (e.g., setup binary tree, setup routing information) is set up in advance by using the separate connections, the start-up time for the broadcasting operations is very small. Also, the tree-based broadcasting scheme improves the performance as the group size gets bigger.

The performance of PVM primitive (*pvm_mcast()*) is not so good for small message sizes but as the message size and group size increase, it shows better performance. In the *pvm_mcast()* where the broadcasting operation is implemented by repeatedly invoking a send primitive, the performance is expected to increase linearly as we increase the group size. Moreover, *pvm_mcast()* constructs a multicasting group internally for every invocation of the primitive, which results in the high start-up time when transmitting small messages as shown in Figure 3 (message size 1 byte).

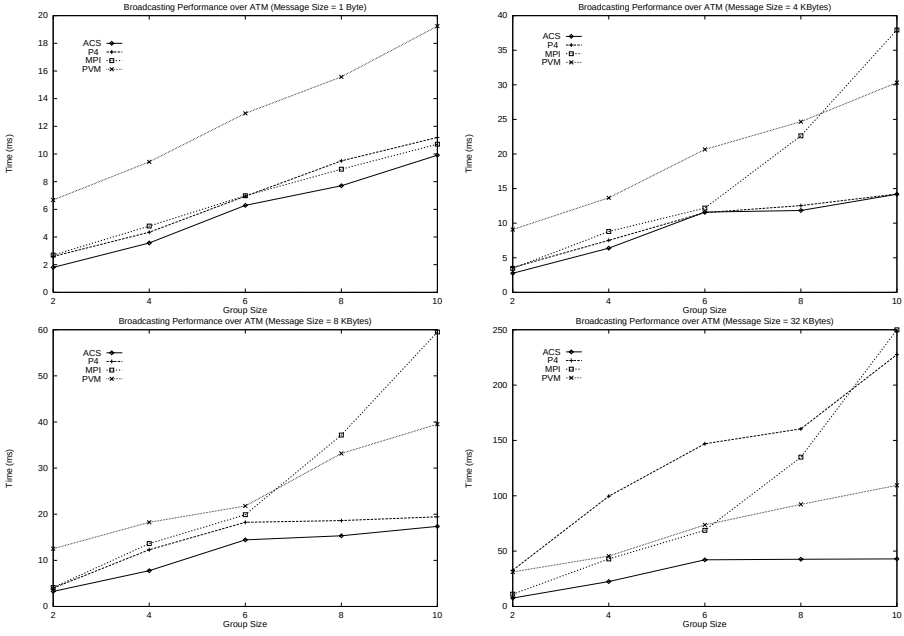

Fig. 3. Comparison of Broadcasting Performance over ATM

The p4 primitive (*p4_broadcast()*) and MPI primitive (*MPI_Bcast()*) shows comparable performance to ACS for relatively small message sizes and small group sizes but it is getting worse drastically when it is running for large message sizes and large group sizes.

4.2 BPNN Learning Algorithm

Training BPNN for character recognition is one of the problems in the Artificial Intelligence (AI) area which require highly intensive computation. We used master/slave programming model to parallelize this application. This application intensively uses the broadcasting primitives when distributing the weight vectors to all the *slave* processes. The BPNN used in this experiment has 100 input nodes, 630 hidden nodes, and 4 output nodes to train 16 input vectors which represent the hexadecimal digits from 0x01 to 0x0F.

Figure 4 shows the performance comparison of each message-passing system running over four homogeneous workstations (e.g., four SUN-4 workstations running SunOS 5.5 or four IBM/RS6000 workstations running AIX 4.1) and 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 4, the ACS implementation outperforms other implementations regardless of the platform used. In the BPNN application where large messages are broadcasted repeatedly, the performance improvement is noticeable and it is widening as we increase the group size. We believe that most

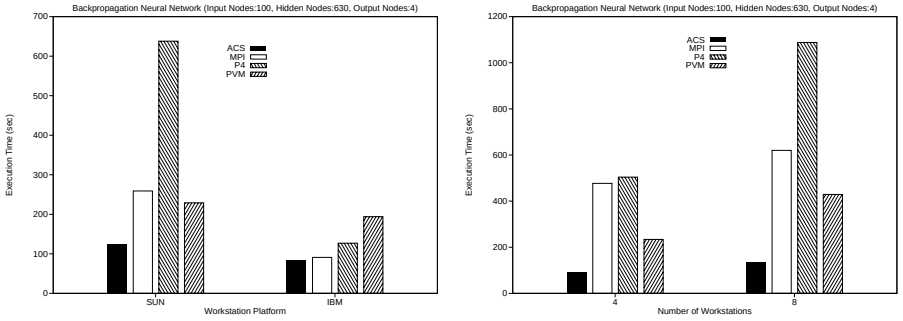

Fig. 4. Comparison of Application Performance

of the improvements of ACS are due to the overlapping of communication and computation and the tree-based broadcasting primitive.

5 Conclusion

In this paper, we have outlined the software architecture of a multithreaded message-passing system, ACS, and presented how ACS architecture can be applied to provide flexible and application-level group communication services. We have evaluated the performance of ACS group communication services and showed that ACS outperforms other message-passing systems. It is clear that the ACS novel architecture, which separates the data and control transfer and tree-based multicasting scheme played an important role in improving the performance of the communication primitives and the ACS applications.

References

1. S. Park and S. Hariri, "ACS: An Adaptive Communication System for Heterogeneous Wide-Area ATM Clusters", *Cluster Computing Journal*, pp. 229–246, 1999.
2. R. Butler and E. Lusk, "Monitors, message, and clusters: The p4 parallel programming system", *Parallel Computing*, Vol. 20, pp. 547–564, April 1994.
3. V. S. Sunderam, "PVM: A Framework for Parallel Distributed Computing", *Concurrency: Practice and Experience*, Vol. 2, No. 4, pp. 315–340, December 1990.
4. MPI Forum, "MPI: A Message Passing Interface", *Proc. of Supercomputing '93*, pp. 878–883, November 1993.
5. R. Renesse, T. Hickey, and K. Birman, "Design and performance of Horus: A lightweight group communications system", Technical Report TR94-1442, Cornell University, 1994.
6. L. E. Moser, P. M. Melliar-Smith, D. A. Agarwal, R. K. Budhia and C. A. Lingley-Papadopoulos, "Totem: A Fault-Tolerant Multicast Group Communication System", *Communications of the ACM*, Vol. 39, No. 4, pp. 54-63, 1996.
7. D. Dolev and D. Malki, "The Transis Approach to High Availability Cluster Communication", *Communications of the ACM*, Vol. 39, No. 4, pp. 64–70, 1996.