Scheduling Multiple Multicast for Heterogeneous Network of Workstations With Non-blocking Message-Passing

Jan-Jan Wu
Shih-Hsien Yeh
Institute of Information Science
Academia Sinica
Taipei 11529 Taiwan R.O.C.
{tristan,wuj}@iis.sinica.edu.tw

Pangfeng Liu
Dept. CSIE
National Chung Cheng Univ.
Chiayi, Taiwan, R.O.C.
pangfeng@cs.ccu.edu.tw

Abstract

This paper proposes efficient algorithms for implementing multicast in heterogeneous workstation/PC clusters. Multicast is an important operation in many scientific and industrial applications. Its efficient implementation on distributed-memory machines plays a critical role in the performance of distributed-memory parallel computing applications.

Our work distinguishs itself between others in two aspects: (1) In contrast to the blocking *communication model used by existing works, we model communication on heterogeneous clusters more accurately by a* non-blocking *model, and design multicast algorithms that can fully exploit the advantage of non-blocking communication, (2) While existing works only solve single-node multicast problem, we propose efficient algorithms for implementing general multi-node multicast (in which single-node multicast is a special case). Our simulation results demonstrate performance improvement of multicast by 20% to 160% compared to existing algorithms.*

1 Introduction

Due to the commodity nature of workstations and networking equipments, cluster environments are gradually becoming heterogeneous. This trend is forcing network of workstations/PCs to be redefined as Heterogeneous Network of Workstations (HNOW).

Many research projects are currently in progress to provide efficient communication for workstation/PC cluster systems [1, 4, 7, 9, 8, 5, 16, 6, 11, 10, 15, 17]. However, most of these research projects focus on homogeneous clusters. Communication algorithms designed for homogeneous clusters have been shown to be very inefficient for heterogeneous clusters [2].

In this paper we study the multicast problem in HNOW systems. Multicast is an important operation in many scientific, industrial and commercial applications. In a single-node multicast, a source node sends the same message to a subset of nodes in the system. Multi-node multicast is a general case in that multiple source nodes issue multicast communication simultaneously. Multi-node multicast operations are frequently used in sparse matrix computation, scientific simulation and Internet applications.

Multicast can be implemented at different levels: hardware-supported, network interface firmware-supported, and software implementation based on point-to-point messages. We focus on software implementation of multicast because it does not require modification of hardware/firmware and thus is portable to different cluster environments.

Software-based multicast in heterogeneous systems has not been investigated until very recently. In software-based approach, a multicast is implemented as p sequences of send/receive tasks, where p is the number of processors involved in the multicast. The fact that finding optimal sequences of tasks for multicast on heterogeneus systems is NP-complete has led to a number of research works on de-

vising heuristic algorithms [12, 2, 13].

The Efficient Collective Operation (ECO) package [12] was developed for networks of heterogeneous workstations. ECO uses heuristic algorithms to partition the workstations participating in a collective communication into subnetworks based on pair-wise round-trip latencies between workstations. It then decomposes the collective communication into two phases: inter-subnetwork and intra-subnetwork. ECO automatically chooses a suitable tree algorithm for each of these phases.

Banikazemi et al. [2] proposed a Fastest-Node-First (FNF) algorithm employs a heuristic that in each iteration of the algorithm, the fastest node which has not received the message is added to the tree. The simulation results show that the FNF algorithm achieves near optimal solution for multicast communication on HNOW systems of up to 10 nodes [2].

Bhat et al. [13] proposed a communication framework that characterizes heterogeneity of both processing nodes and networks. A cost function is constructed for each pair of nodes, which represents the communication cost between the two nodes. Based on the communication framework, the authors designed a number of heuristic algorithms (Fastest Edge First (FEF), Earliest Completing Edge First (ECF)) for multicast and broadcast on distributed networks. The experimental results show that FEF and ECF outperform the FNF approach significantly on distributed heterogeneous networks. The ECF heuristic also represents the best known result in this area.

The above works have two restrictions however. First, they all focus on single-node multicast and hence their algorithms cannot handle general cases of multi-node multicast. Second, they all assume a *blocking* communication model; that is, a node cannot send a message until the previously sent message has been received by the destination. In fact, many networks and operating systems support *non-blocking* communication; that is, after an initial start-up time, the sender can send the next message. The previously sent messages can be completed by the network without intervention of the sender. Thus, a node can send out several messages before the first message is received by the destination. To optimize performance of multicast communication, it is extremely important to take the factor of non-blocking communication into concern, which is not possible under a *blocking* communication model.

In this paper, we design two efficient algorithms, namely, *Work-Racing* and *Work-Racing-Preemptive*, for the general multi-node multicast communication on heterogeneous systems. The *Work-Racing* algorithm is a greedy algorithm based on the notion of "virtual time" that measures the time that a destination node of a multicast has spent in receiving messages. WR outperforms the best known result, the *Earliest-Completion-First* algorithm by up to 20% in our simulation result.

The Work-Racing algorithm does not fully exploit the advantage of non-blocking send operation. The *Work-Racing-Preemptive* algorithm further reduces the completion time of multicast by actively filling the receiving node's idle time frames with useful send operations. With this optimization, the WRP algorithm outperforms the Earliest-Completion-First algorithm by up to 160% in our simulation result.

2 Communication Model for Heterogeneous Clusters

Our model measures the cost of a point-to-point message transfer between the sender P_i and the receiver P_j using three parameters (1) the send overhead $S(i, m)$, which represents the message initialization cost on sender P_i for sending a message of length m, (2) the network link transmission rate $X(i, j)$, which accounts for the unit transmission time between P_i and P_j, and (3) receive overhead $R(j, m)$, which represents the software overhead on receiver P_j for receiving and copying the message from the network buffer to the user space. Based on these three parameters, the latency for transmitting a message of length m between the two nodes is given in Equation 1. Each parameter in the equation can be measured using the ping-pong scheme described in [3].

$$T(i, j, m) = S(i, m) + X(i, j) * m + R(j, m) \quad (1)$$

Note that in heterogeneous systems these parameters for each pair of nodes may vary dramatically due to the heterogeneity in system architecture, operating system, network protocol, network interface, etc. Furthermore, we make the following assumptions.

- A send operation is *non-blocking*. In other words, after an initial *start-up* time (i.e. the send overhead), the sender can execute its next send operation.

- A receive operation is *blocking*. That is, after issuing a receive operation the receiving node can continue to execute its next operation only when the received message arrives and is removed from the local network buffer to the local memory of the receiving node. We consider this assumption reasonable as in most applications, the computation following a receive operation is very likely to require data in the received message and hence the computation cannot start until the required data is ready in the local memory.

3 Multi-node Multicast Problem

A multi-node multicast has multiple source nodes multicasting their messages to their destination nodes simultaneously. Consider a heterogeneous cluster consisting of N nodes and let $\mathbf{N} = \{\mathbf{P_1}, \mathbf{P_2}, \ldots, \mathbf{P_N}\}$ be the set of all nodes. Let $\mathbf{S}$ denote the set of the multicast source nodes, and $\mathbf{D_i} \subseteq \mathbf{N} - \{\mathbf{P_i}\}$ be the set of destination nodes for source node $P_i \in \mathbf{S}$. After the multicast communication, each node in $\mathbf{D_i}$ has a copy of the message from source node P_i, denoted by m_i.

A multicast is accomplished by a series of *communication tasks*. A communication task (op, i, j, k) is a send or receive operation that P_i is scheduled to execute. For $op = send$, P_i is scheduled to send m_k to P_j, where $P_i \in \mathbf{D_k} + \{\mathbf{P_k}\}$ and $P_j \in \mathbf{D_k}$. For $op = recv$, P_i is scheduled to receive m_k from P_j where $P_i \in \mathbf{D_k}$ and $P_j \in \mathbf{D_k} + \{\mathbf{P_k}\}$. The *task schedule* of P_i, denoted by $\mathbf{TaskList_i}$, is an ordered list of communication tasks (op, i, j, k). The tasks in the task schedule are executed in the order they appear.

The multicast scheduling problem is to determine a task schedule for each participating node so that the time to deliver all the messages is as short as possible. Finding the optimal schedule is NP-complete. We have designed several heuristic algorithms to solve this problem.

3.1. A Lower Bound

We first derive a lower bound on the time to solve a multiple multicast problem. Since it is too computationally expensive to determine the optimal completion time of a multiple multicast, we idealize our model and deduce an *idealized optimal completion time* as a lower bound. In this model we assume that a node is allowed to receive a message and, meanwhile, send multiple messages in parallel. That is, a receive operation will not be delayed by any send operation, and the messages from one node to different destinations can be processed in parallel.

Since the completion time of a multiple multicast is determined by the maximum of the *task completion time* of all the destination nodes, we can compute the lower bound on the task completion time for each destination node, and select the maximum as the lower bound on the overall completion time. First we compute the shortest path for each pair of source and destination in the idealized model. Let $L[P_j, P_i]$ denote the cost of the shortest path from source P_j to its destination P_i, i.e., $L[P_j, P_i]$ represents the *earliest-reach-time* at which the message from P_j can reach P_i. Thus, the lower bound on the task completion time of a destination node can be defined as follows.

Definition 1 *Assume that a node P_i needs to receive messages from n_i source nodes. Let $I(k)$ denote the kth source node from which P_i receives the message. Suppose that P_i receives messages in the increasing order of the earliest reach times of the messages, i.e., $L[I(k), P_i] \geq L[I(k-1), P_i]$ for $1 < k \leq n_i$, then we call this receiving order the* earliest-reach-first *order.*

Let l_k be the message size of the kth source node $I(k)$. The earliest receiving time for P_i to complete receiving the message from the kth source node $I(k)$, denoted by $T_r(i,k)$, can be derived recursively as follows. Note that Equation 2 uses the fact that the receive overhead from different source nodes cannot overlap, and the earliest-reach-first order can minimize $T_r(i,k)$.

$$T_r(i,k) = \begin{cases} L[I(k),i] & k = 1 \\ \max\{T_r(i,k-1) + R(i,l_k),\ L[I(k),i]\} & 1 < k \leq n_i \end{cases} \quad (2)$$

The lower bound on the completion time of the multiple multicast is the maximum of all $T_r(i,n_i)$, for all i.

4 Scheduling Algorithms for Multiple Multicast

We first give several definitions in order to describe our scheduling algorithms.

Available time. The *available time* of P_i, denoted as $Avail_i$, is the earliest time at which P_i can execute a new task. Initially, the available time of the participating nodes is zero.

Arrival time. Assume that P_i is scheduled to send m_k to P_j. Let l_k be the size of m_k, then m_k will arrive at the network buffer of P_j at time $ArrivalTime(i,j,k)$.

$$ArrivalTime(i,j,k) = Avail_i + S(i,m) + X(i,j) * l_k \quad (3)$$

Completion time. The completion time, $CompleteTime(i,j,k)$ of a task $(send, i, j, k)$ is defined as follows. If the destination node P_j is not available when m_k arrives at its network buffer, it will not be processed until P_j becomes available.

$$CompleteTime(i,j,k) = \max(ArrivalTime(i,j,k), Avail_j) + R(j,l_k) \quad (4)$$

4.1. Fastest-Edge-First Algorithm

We extend the Fastest Edge First (FEF) heuristic algorithm in [14] to solve the multi-node multicast problem. Similar to the FEF algorithm, we keep a sender set A_k and a receiver set B_k for each source node P_k of the multi-node

multicast. Initially $A_k = \{P_k\}$ and B_k contains all the destination nodes of P_k.

In each iteration we select the smallest weight edge $(i,\ j,\ k)$ where node P_i belongs to A_k and node P_j belongs to B_k, and then move P_j from B_k to A_k. The same steps repeat until all B_k become empty. The *weight* of an edge is defined as the point-to-point latency between the sender and the receiver.

Similar to [13], a sorted edge list and a sorted sender list according to their edge weights is maintained, and the complexity of this algorithm is $O(N^3 \log N)$ where N is the number of nodes.

4.2. Earliest-Completion-First Algorithm

This algorithm is based on the *Earliest-Completion-First* algorithm in [14] Similar to the FEF algorithm, for each source node P_k of the multi-node multicast we keep a *sender set* A_k and a *receiver set* B_k. In each iteration, the algorithm selects the earliest completing task for each source node which has not yet completed its multicast operation. Then, among these earliest-completing tasks, it selects the task with the minimum completion time as the next task to be scheduled. The receiver of the selected task is then moved from the receiver set to the sender set. The same steps repeat until all B_k become empty.

In each iteration, it requires $O(N^2)$ steps to compute the available times of the senders and receivers. The algorithm iterates N^2 times. Thus the total time for the ECF algorithm is $O(N^4)$.

4.3. Work Racing Algorithm

In our communication model, a receive operation is *blocking*, that is, if multiple messages arrive at the receiving node, they will be queued in the buffer until the receiving node has finished receiving previous message. The key idea in the *Work Racing* (WR) heuristic is that, if the chance of message built-up at the destination nodes is reduced, then the messages can be delivered as early as possible, resulting in earlier completion of the multicast. One way to implement this strategy is to allow faster destination node to receive messages more often than slower nodes. However, the scheduling should also be fair so that slower nodes will not be starved forever.

We define the concept of *virtual time* of a destination node to be the time this node has spent in receiving messages. WR selects the destination node with the earliest virtual time as the receiver of the next message. Then WR selects a message and a sender for this message based on the earliest-completion-first principle. The new send/recv task is then appended to the task schedule of the sender/receiver. After that, the *virtual time* of the receiver is increased by the amount of work it has just accomplished. This mechanism ensures that the faster destination node will be served more often, and each destination will be served fairly according to their communication capability. In the following we elaborate on this algorithm.

- For each destination P_i, assuming P_i has received messages from k sources so far, the Work Racing algorithm keeps the record of the *virtual time*, denoted by $W_{i,k}$, which indicates the services P_i has received from sources. Initially, $W_{i,0}$ is set to $zero$.

- While scheduling a new task, the Work Racing algorithm selects the destination which has the earliest virtual time. If there are multiple possibilities, it chooses the fastest one.

- Each destination node P_i keeps a set of source nodes SR_i of the multiple multicast in which P_i is a destination. Each source node P_k keeps a set of sender nodes A_k which contains the nodes that have received message m_k. Nodes in A_k may relay m_k to other nodes. When a destination P_i is selected, the Work Racing algorithm chooses a source P_m from SR_i and a sender P_j from A_m such that the completion time of the task $(send, j, i, m)$ is the minimum.

- The destination P_i will be scheduled a receive task $(receive, i, j, k)$. Assume that this is the kth receive task that P_i has been scheduled (and we will call the source node of this message the kth source node of P_i), the virtual time $W_{i,k}$ of P_i is updated as follows:

$$W_{i,k} = \begin{cases} A_r[k,i] + R(i,l_k) & k = 1 \\ \max\{W_{i,k-1},\ A_r[k,i]\} + R(i,l_k) & k > 1 \end{cases} \quad (5)$$

  where l_k is the message length of this receive task and $A_r[k,i]$ is the time at which the message from the kth source node arrives at P_i. Let the source node of the message be the nth source in sender P_j. Then, $A_r[k,i]$ is calculated as follows:

$$A_r(k,i) = \begin{cases} W_{s,n} + S(s,l_k) + X(s,i) * l_k & \text{if } P_s \text{ is not the source node} \\ S(s,l_k) + X(s,i) * l_k & \text{otherwise} \end{cases} \quad (6)$$

- Finally, the source node P_j is removed from the source set SR_i of the destination P_i, and the destination P_i is added to the sender set A_k of the source P_k of the multicast.

In general, the completed work of a faster node advances more slowly than that of a slower node. Thus a faster destination nodes can be scheduled earlier than a slower one. Consequently, a faster node has more chances to relay messages for the sources to the slower nodes.

Work-Racing Scheduling Algorithm

Let SR_i be the set of source nodes for a receiving node P_i. Initially, SR_i contains all the source nodes of the multiple multicast which has P_i in their destination sets. Similar to the FEF and the ECF algorithms, we define a sender set A_k for each source P_k of the multiple multicast.

Step 1: Let P_i be the node with the minimum completed work whose source set SR_i is not empty. If there are multiple possibilities, choose the fastest one.

Step 2: Choose $P_k \in SR_i$ and $P_j \in A_k$ such that the completion time of the task $(send, j, i, k)$ is the minimum. Assume the source of this task (i.e. P_k) is the mth source in P_i.

$$
\begin{aligned}
&A_k \leftarrow A_k + \{P_i\} \\
&SR_i \leftarrow SR_i - \{P_k\} \\
&Compute\ W_{i,m} \\
&AppendTask(\mathbf{TaskList_j}, (\mathbf{send, j, i, k})) \\
&Avail_j \leftarrow Avail_j + S(j, l_k) \\
&AppendTask(\mathbf{TaskList_i}, (\mathbf{recv, i, j, k})) \\
&Avail_i \leftarrow CompleteTime(j, i, k)
\end{aligned}
$$

Step 3: Repeat Step 1 and Step 2 until all SR_i become empty.

For a selected receiver, it takes $O(N)$ steps to select the source and the sender. The algorithm iterates at most N^2 times for multiple multicast. Therefore, the overall complexity of the WR algorithm is $O(N^3)$.

4.4. Work-Racing-Preemptive Algorithm

Our communication model assumes that send operations are non-blocking and receive operations are blocking. After issueing a non-blocking send a sender only has to wait until the message goes into the network before starting its next communication. In contrast, a node performing a blocking receive cannot issues another communication until the incoming message is received completely (Equation 1). We illustrate this phenomenon by an example. In the Work-Racing algorithm we always append the new task to the end of the task schedule. This can incur much longer waiting time than necessary. In Figure 1(a), a receive operation $recv[k]$ is issued at time $t1$. However, the message will not arrive until time $t4$, making the receiver idle waiting from $t1$ to $t4$.

To overcome this problem, we propose an optimization to preempt blocking receive with non-blocking send, under the assumption that it will not invalid the original multicast schedule. As illustrated in Figure 1(b), if we preempt the receive task, $recv[k]$, with the send tasks $send[2]$ and $send[3]$, and issue $send[k]$ at $t3$ instead, the waiting time is reduced from $(t4 - t1)$ to $(t4 - t3)$. The idea is that if a new task can preempt a prescheduled receive task safely, we can reduce the waiting time by delaying the preempted receive task.

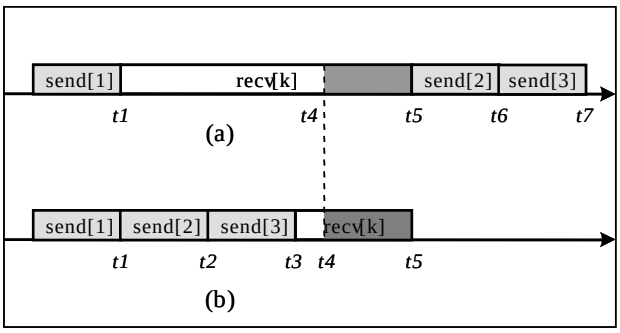

Figure 1. Preemptive Scheduling of Send Tasks

To preempt a receiving task without interfering or destroying the original schedule, we need to observe the following rules.

- The send tasks must be executed in the order as they appear in their schedule list. As in Figure 2(a), the task $send[k]$ must appear before $send[k+1]$.
- If the sender of a task is not the source node of the message, the new send task must be scheduled *after* the sender has received the message. This is to ensure that a sender will not rely a message that it has not yet received. Figure 2(b) shows that only after the task $recv[h+u]$ in which the sender of $send[k+1]$ receives the message will it relay to the receiver of $send[k+1]$.
- The preempting send tasks cannot delay the completion time of the preempted receiving task. To avoid interfering the execution of the preempted receive $task[k]$ in Figure 2(c), the completion time of the preempting send task must be earlier than the arrival time of the message that $task[k]$ is waiting for.

To facilitate the design of the algorithm, we classify the available time of a node as *receive available time* and *send available time*. The receive available time of P_i is the earliest time that P_i can issue a receive, which is defined as the maximum between the completion time of the last send task and the last receive task. The send available time of

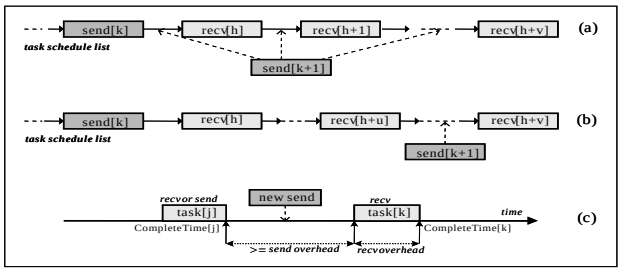

Figure 2. Examples of Preemption Rules

P_i *for* P_k is defined as the earliest time that P_i can send out m_k that satisfies all three properties listed above. The send available time should be later than both the completion time of the last send by P_i and the completion time of the receive in which P_i receives m_k. In addition, based on the new definitions of the available time, the completion time of a communication task defined in Equation 4 should adopt the new definitions of the available time.

We improve the work racing algorithm by preempting the receive operations with send. By the preemptive property, the new completed work definition reflects more accurately the amount of work a node has performed as a receiver. For each iteration the node with the minimum completed work is selected as the destination of the new task. Then we select a source node and a sender node so that the task completion time is minimized, under the assumption that a send can preempt a receive operation.

Work-Racing-Preemptive Algorithm

Step 1: Let P_i be the node with the minimum completed work whose SR_i is not empty. If there are multiple possibilities, choose the fastest one.

Step 2: Chose $P_k \in SR_i$ and $P_j \in A_k$ such that the completion time of the $(send, j, i, k)$ is the minimum. Assume P_k is the mth source for node P_i.

$$
\begin{aligned}
&A_k \leftarrow A_k + \{P_i\} \\
&SR_i \leftarrow SR_i - \{P_k\} \\
&Compute\ W_{i,m} \\
&UpdateReceiverState(i, j, k, CompleteTime(j, i, k)) \\
&UpdateSenderState(j, i, k) Avail_j \leftarrow Avail_j + S(j, l_k) \\
&Avail_i \leftarrow CompleteTime(j, i, k)
\end{aligned}
$$

Step 3: Repeat Step 1 and Step 2 until all SR_i become empty.

The procedure $UpdateSenderState$ deals with the preempting process of a new send task in the send node. The procedure $UpdateReceiverState$ deals with the process of appending the new receive task to the last task currently in the task schedule of the receive node.

The complexity of the WRP algorithm is also $O(N^3)$ since it follows the same structure of the WR algorithm and the preempting process requires constant steps.

5 Experimental Results

To evaluate the performance of the scheduling algorithms we have developed a software simulator to calculate the communication completion time and computation overheads. The simulator is modeled by the number of nodes, the network capability, and the multicast patterns. We classify the nodes into several classes according to their send and receive overheads. The constant parts are in the range of $80\mu s$ to $400\mu s$, and the length dependent parts are chosen in the range of $0.0001\mu s/byte$ to $0.01\mu s/byte$. As for the the network capability, two link transmission rates, $155Mbps$ and $1Gbps$, are considered in our simulations. We consider three message length in our experiments – small messages ($\leq 1kbytes$), large messages ($1Mbytes$ and $1.5Mbytes$), and a hybrid of small and large messages. We present and discuss the results from three cases – a single broadcast, all-to-all broadcast and general multiple multicast.

5.1. Single Broadcast

The broadcast time on a N-node system is measured as follows. We repeat the broadcast for N times and each time a different processor is chosen as the source. The time is measured as the average of the completion time from the N broadcasts.

The completion time of the FEF algorithm is much longer than those of the others. Since the ECF, WR and WRP generate the same task schedules for a single broadcast, the completion time of these three heuristics are almost identical.

5.2. Multiple Multicast

For multiple multicast we consider a 64-node HNOW system, with three different combinations of message length and two classes of networks described earlier. Given the set of source nodes, the simulator picks the message sizes and randomly chooses the destination nodes for each source node. The completion time is taken from the average of 1000 runs with random configuration of source and destination nodes.

Figure 3 and Figure 4 show the completion times in a fast network and a slow network respectively. The WRP algorithm performs best among all four algorithms, especially for large messages. It can be observed that the completion times of the WRP algorithm are within 2.5 times of the lower bounds on large systems.

6 Conclusion and Future Work

In this paper we have presented four algorithms for multiple multicast in heterogeneous NOW systems. We extend the Fastest-Edge-First and the Earliest-Completion-First heuristics in [14, 2] to solve the multiple multicast problem in a non-blocking communication environment. We have also designed two new algorithms, Work-Racing and Work-Racing-Preemptive that were inspired by a flow control mechanism for packet-switched networks.

These algorithms have been evaluated using a software simulator. A lower bound for the completion time of multiple multicast is also derived. The simulation results demonstrate the performance advantage of the two new algorithms on systems of up to 64 nodes.

One of the main challenges in designing multicast algorithms is the handling of dynamic multicast patterns. For dynamic patterns, the algorithm must computes the schedule very quickly, without sacrificing the schedule quality. Although the Work-Racing and the Work-Racing-Preemptive algorithms that we have proposed can determine efficient schedules with negligible overhead for multicast of long messages, there is room for improvement for short messages. We are considering an incremental optimization approach for dynamic multiple multicast. That is, given a multiple multicast pattern P and a good schedule S for it, our goal is to derive a good schedule for a similar pattern P' from S instead of recompute the schedule from scratch.

We are also investigating the possibility of extending this work to handling collective communication over Wide-Area-Networks (WAN). The first step toward WAN communication is to enhance our communication model with the ability to predict the behavior of communication in WAN. In order to do so, we need to statistically analyze the effect of the cross-traffics from other sessions and the traffic pattern of a communication in order to measure the network transmission time more accurately.

References

[1] J. Bala, Bruck, R. Cypher, P. Elustando, A. Ho, C.-T. Ho, S. Kipnis, and M. Snir. CCL: A portable and tunable collective communication library for scalable parallel computers. *Journal of Parallel and Distributed Computing*, 6(2):154–164, February 1995.

[2] M. Banikazemi, V. Moorthy, and D. K. Panda. Effi cient collective communication on heterogeneous networks of workstations. In *Proceedings of International Conference on Parallel Processing*, pages 460–467, 1998.

[3] M. Banilazemi, J. Sampathkumar, S. Prabhu, D. K. Panda, , and P. Sadayappan. Communication modeling of heterogeneous networks of workstations for performance characterization of collective operations. In *Proceedings of the Heterogenenous Computing Workshop*, 1999.

[4] J. Bruck, R. Cypher, P. Elustando, A. Ho, C.-T. Ho, V. Bala, S. Kipnis, and M. Snir. Effi cient message passing interface (MPI) for parallel computing on clusters of workstations. *Journal of Parallel and Distributed Computing*, pages 19–34, January 1997.

[5] A. Chien, S. Pakin, M. Lauria, M. .Badanan, K. Hane, and L. Giannini. High performance virtual machines (HPVM): clusters with supercomputing and performance. In *Proceedings of 8th SIAM Conference on Parallel Processing for Scientifi c Computing*, 1996.

[6] D. C. et al. Parallel computing on the berkeley now. In *9th Joint Symposium on Parallel Processing*, 1997.

[7] K.-P. Fan and C.-T. King. Effi cient multicast on wormhole switch-based irregular networks of workstations and processor clusters. In *Proceedings of Parallel and Distributed Computing Symposium (PDCS)*, 1997.

[8] W. Gropp and E. Lusk. User's guide for mpich, a portable implementation of mpi. Technical Report ANL/MCS-TM-ANL-96/6, Mathematics and Computer Science Division, Argonne National Laboratory, 1996.

[9] C. Huang, Y. Huang, and P. K. Mckinley. A thread-based interface for collective communication on ATM networks. In *Proceedings of ICDCS*, pages 254–261, 1995.

[10] Y. Hwang and P. K. Mckinley. Effi cient collective operations with atm network interface support. In *Proceedings of ICPP*, 1996.

[11] M. Lauria. High performance MPI implementation on a network of workstations. Technical Report Master Thesis, Department of Computer Science, University of Illinois at Urbana-Champaign, 1996.

[12] B. Lowekamp and A. Beguelin. ECO: Effi cient collective operations for communication on heterogeneous networks. In *Proceedings of International Parallel Processing Symposium*, pages 399–405, 1996.

[13] C. S. R. V. K. P. P. B. Bhat. Effi cient collective communication in distributed heterogeneous systems. In *Proceedings of Intnl. Conf. Distributed Computing Systems (ICDCS)*, 1999.

[14] V. K. P. P. B. Bhat and C. S. Raghavendra. Adaptive communication algorithms for distributed heterogeneous systems. In *Proceedings of the 7th IEEE Intnl. Symposium on High Performance Distributed Computing (HPDC)*, 1998.

[15] J. park, H. A. Choi, N. Nupairoj, and L. M. Ni. Construction of optimal multicast trees based on the parameterized communication model. In *Proceedings of the International Conference on Parallel Processing*, 1996.

[16] H. Tezuka, A. Hori, Y. Ishikawa, and M. Sato. PM : An operating system coordinated high-performance communication library. In *Proceedings of High Performance Computing and Networking (LNCS vol. 1225)*, 1997.

[17] K. Verstoep, K. Langendoen, and H. Bal. Effi cient reliable multicast on myrinet. In *Proc. Intl. Conf. Parallel Processing*, volume III, pages 156–165, August 1996.

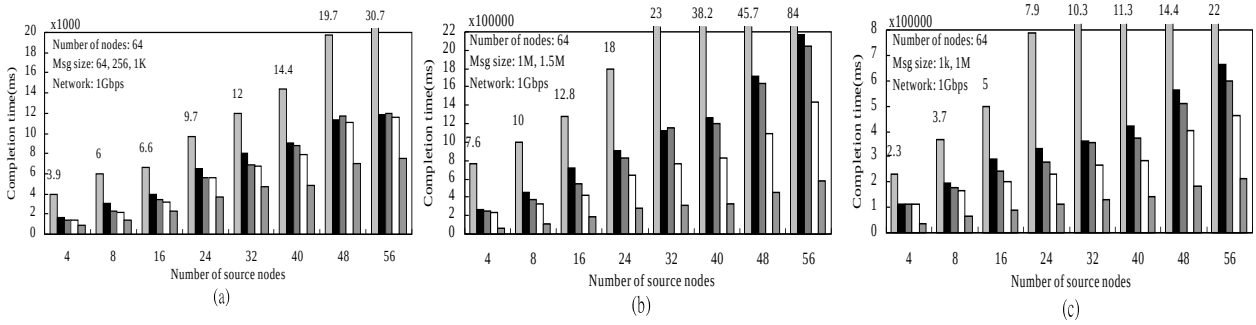

Figure 3. Completion time of multiple multicast in a fast network: from left to right: FEF, ECF, WR, WRP and the lower bound.

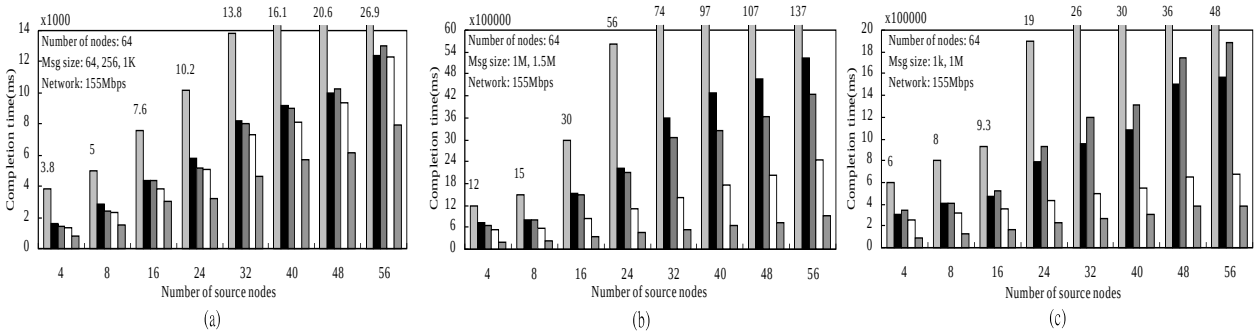

Figure 4. Completion time of multiple multicast in a slow network: from left to right: FEF, ECF, WR, WRP and the lower bound.