Adaptive Communication Algorithms for Distributed Heterogeneous Systems

Prashanth B. Bhat * and Viktor K. Prasanna *
Department of EE-Systems, EEB 200C
University of Southern California
Los Angeles, CA 90089-2562
{prabhat, prasanna}@halcyon.usc.edu

C.S. Raghavendra
The Aerospace Corporation
P. O. Box 29257
Los Angeles, CA 90009
raghu@aero.org

Abstract

Heterogeneous network-based systems are emerging as attractive computing platforms for HPC applications. This paper discusses fundamental research issues that must be addressed to enable network-aware communication at the application level. We present a uniform framework for developing adaptive communication schedules for various collective communication patterns. Schedules are developed at run-time, based on network performance information obtained from a directory service. We illustrate our framework by developing communication schedules for total exchange. Our first algorithm develops a schedule by computing a series of matchings in a bipartite graph. We also present a $O(P^3)$ heuristic algorithm, whose completion time is within twice the optimal. This algorithm is based on the open shop scheduling problem. Simulation results show performance improvements of a factor of 5 over well known homogeneous scheduling techniques.

1. Introduction

With recent advances in high-speed networks, *metacomputing* has emerged as a viable and attractive computational paradigm. A metacomputing system [23] consists of geographically distributed supercomputers and visualization devices. These are interconnected by a heterogeneous collection of local and wide-area networks. High performance applications can be executed over such a *networked virtual supercomputer,* wherein the distributed computational resources are used in a coordinated way, very much as though they were part of a single computer system.

The potential of metacomputing has been demonstrated by the Global Information Infrastructure (GII) testbed at SC '95 [16]. The testbed linked dozens of high performance computers and visualization machines with existing high-bandwidth networks and telephone systems. Some of the leading metacomputing research projects are Globus, Legion, VDCE, and MSHN, to name a few. These will be discussed further in Section 2.

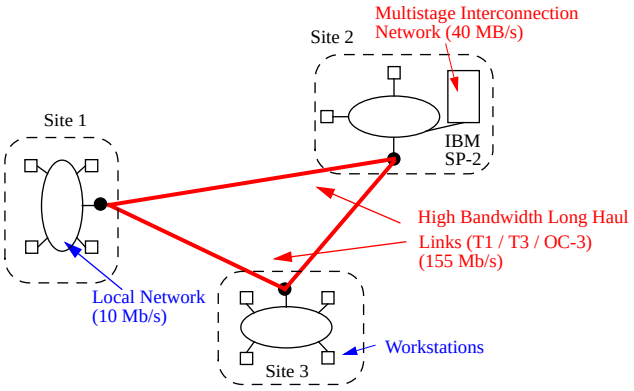

Figure 1. A typical metacomputing system.

Figure 1 shows an example of a small-scale metacomputing system. The compute nodes in the system are located at three different sites. Some of the nodes are high-end supercomputing systems, while others are workstations. The example shown in this figure has three kinds of interconnection networks: (i) the multi-stage interconnection network within the IBM SP-2, (ii) local networks at each site, and (iii) high bandwidth long haul ATM or T3 links between the sites.

Although network-based computing platforms offer significant advantages for high performance computing, effective use of their resources is still a major challenge. Computational and communication resources are typically shared among different applications. Computational tasks may be preempted by higher priority processes. Network conditions change continuously, and run-time loads cannot be determined apriori. Applications must therefore be capable of

*Supported by the DARPA/ITO Quorum Program through the Naval Postgraduate School under subcontract number N62271-97-M-0931.

adapting to changing system conditions.

In this paper, we develop communication techniques that enable applications to adapt to variations in network conditions. We focus on collective communication patterns among application processes executing over a heterogeneous network. Our goal is to develop efficient application-level implementations of these communication routines. We assume the availability of end-to-end send and receive communication routines, which can be invoked between any pair of nodes. The details of network topology, routing, and flow control policies are therefore hidden from the application.

Efficient algorithms for various collective communication patterns have been developed for tightly coupled parallel architectures with homogeneous networks [2, 19]. These algorithms have been incorporated into communication libraries and implementations of MPI. However, these techniques can perform poorly in metacomputing systems due to the heterogeneity among network bandwidths. Further, these are static algorithms, with no provision for adaptivity to network conditions.

In Section 3, we introduce our approach for developing network-aware communication techniques. The key components of our approach are: (i) a directory service which provides information on current network performance, (ii) a communication model which estimates the time for individual communication events, (iii) timing diagrams which abstractly represent both the communication pattern and the network performance, and (iv) scheduling algorithms which reduce overall communication time by appropriately positioning the communication events in the timing diagram.

Our approach is a general one, and can be used for different collective communication patterns and a variety of network-based architectures. In Section 4, we use our scheduling framework for the all-to-all personalized communication pattern in a typical metacomputing environment. We develop three different scheduling algorithms for this problem. Our first algorithm is a matching-based algorithm. It first constructs a bipartite graph, with edge weights equal to communication costs between processor pairs. A series of maximum weight complete matchings in this graph are then computed. The communication schedule is derived from these matchings. Our second algorithm is a greedy approximation to this matching-based algorithm. The third algorithm is a heuristic that has been used for the open shop scheduling problem. We evaluate the performance of our algorithms by simulation, and compare it with a well-known scheduling technique used in homogeneous scenarios. Our results show excellent improvements in performance.

This paper represents one of the early efforts to formalize research problems related to network-based computing. In Section 6, we discuss other fundamental research issues that are motivated by the need for network-aware applications. We consider enhancements to our communication model and techniques to reduce the complexity of the scheduling algorithm. We also discuss communication scheduling in the presence of QoS constraints.

The rest of the paper is organized as follows: Section 2 discusses related research projects in metacomputing. Section 3 introduces our approach for deriving efficient communication techniques, and describes each of the components in detail. Section 4 formulates the all-to-all heterogeneous data communication problem, and presents our scheduling algorithms for this communication pattern. Section 5 presents simulation results of our algorithms. Section 6 discusses future research directions for network-aware communication scheduling. Section 7 concludes the paper.

2. Related Work

Several research projects are developing software infrastructure and defining API functionality for network-based computing systems. We believe that our work will complement these software development efforts. Our scheduling techniques from Section 4 can be incorporated into these software systems and tool-kits. A few projects are also investigating performance related issues.

The Globus project [9, 10] at ANL and USC-ISI is developing a set of low level core services, called the Globus tool-kit. This includes modules for resource location and allocation, communications, authentication, process creation, and data access. Higher level systems software and applications then build upon the functionality provided by the tool-kit. Nexus is the communications library component of the Globus tool-kit. Globus incorporates a directory service, called the Metacomputing Directory Service (MDS). Applications can query MDS for information on current loads on the processing nodes, as well as end-to-end network performance between node pairs.

The Legion project at the University of Virginia uses an object oriented approach to metacomputing system design [12, 18]. The philosophy is to hide the complexity of resource scheduling, load balancing, etc. from the application developer. The object oriented properties of encapsulation and inheritance, as well as software reuse, fault containment, and reduction in complexity are used to achieve this goal.

The Virtual Distributed Computing Environment (VDCE) at Syracuse University [25, 26] aims to develop a complete framework for application development, configuration, and execution. A GUI allows library routines or user developed routines to be combined into an application task graph. The task graph is then interpreted and configured to execute on currently available resources.

At Carnegie Mellon, the ReMoS (Resource Monitoring System) project [7] is developing a portable and system-independent API that allows applications to obtain informa-

tion about network status and capabilities. Most architectures generate information about the network hardware and software in a system-specific format. ReMoS provides a standard interface format that is independent of the details of any particular type of network. ReMoS explicitly accounts for resource sharing between applications.

The Management System for Heterogeneous Networks (MSHN) [21] project at Naval Postgraduate School, USC, and Purdue University is designing and implementing a Resource Management System (RMS) for distributed heterogeneous and shared environments. MSHN assumes heterogeneity in resources, processes, and QoS requirements. Processes may have different priorities, deadlines, and compute characteristics. The goal is to assign resources to individual applications so that their QoS requirements are satisfied. MSHN also addresses uncertainty due to unpredictable loads in the operating environment. Various task mapping and scheduling algorithms are being developed [1, 20]. Our research is a part of the MSHN effort.

Communication performance in the presence of multiple heterogeneous networks has been investigated in [14, 15]. Experiments are performed on a local cluster of workstations, interconnected with ATM, Ethernet, and FibreChannel networks. The performance characteristics of each of the networks are first evaluated by measuring the time for sending messages of various sizes over the particular network. These characteristics are used to choose a suitable technique for data communication. The *Performance Based Path Selection (PBPS)* technique selects one of the networks to be used for a communication event, depending on the size of the message. The *Aggregation* technique uses multiple networks at the same time, by breaking up the message into multiple parts and sending these parts over different networks. However, this research only considered point-to-point communication between a pair of nodes in the system. Collective communication patterns such as all-to-all or all-to-some were not studied. Such collective communication patterns typically occur in most parallel applications.

Distributed heterogeneous computing has important military applications as well. The BADD (Battlefield Awareness and Data Dissemination) [6] program at DARPA aims to develop an operational distributed data communication system. The goal is to deliver to warfighters an accurate, timely, and consistent picture of the battlefield, as well as to provide access to key transmission mechanisms and worldwide data repositories. [24] considers an important data staging problem that arises in such heterogeneous networking environments, where data items must be moved from their initial locations to requester nodes. Each data request also has a time-deadline and priority associated with it. In [24], a heuristic based on the multiple-source shortest-path algorithm is used to find a communication schedule for this data staging problem. In Section 6, we mention some related problems.

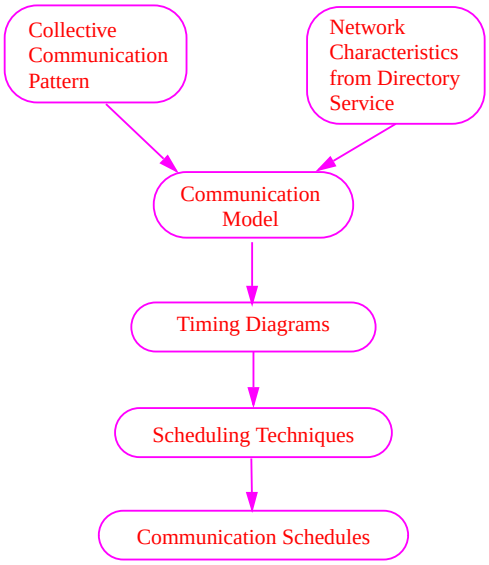

Figure 2. Our communication scheduling approach.

3. Our Approach: A Uniform Framework for Communication Scheduling

Figure 2 shows our approach for developing adaptive communication techniques, which are essential for network-aware applications. We use a communication scheduling framework consisting of four key components: (i) A directory service, (ii) An analytical communication model, (iii) Timing diagrams, and (iv) Scheduling algorithms. The directory service provides information on current network performance. Based on this information and the application's communication pattern, the communication model is used to compute the time for each node-to-node communication event. This is then represented using a timing diagram. A scheduling algorithm uses a timing diagram as input, and appropriately schedules the events to reduce the overall communication time. We discuss each of these components in further detail.

3.1. Directory Service

Since network load in shared environments varies with time, a directory service which provides information on current network performance is essential. A suitable directory infrastructure is therefore a key component of our framework for developing adaptive communication techniques.

	AMES	ANL	IND	USC-ISI	NCSA
AMES		34.5	89.5	12	42
ANL	34.5		20	26.5	4.5
IND	89.5	20		42.5	21.5
USC-ISI	12	26.5	42.5		29.5
NCSA	42	4.5	21.5	29.5	

Table 1. Latency (ms) between 5 GUSTO sites.

	AMES	ANL	IND	USC-ISI	NCSA
AMES		512	246	2044	391
ANL	512		491	693	2402
IND	246	491		311	448
USC-ISI	2044	693	311		4976
NCSA	391	2402	448	4976	

Table 2. Bandwidth (kbits/s) between 5 GUSTO sites.

The information provided by the directory makes it possible to develop communication schedules which are adaptive to changes in network performance. At run-time, applications can query the directory service through an Application Programming Interface. For example, the Metacomputing Directory Service (MDS) in Globus [8] provides current information on start-up costs and end-to-end bandwidths between every pair of processors. The ReMoS API, developed at CMU [7], is an example of an API that is independent of the details of network hardware.

Table 1 and 2 are examples of information provided by the directory service in GUSTO, which is a testbed of Globus. The directory provides current values of end-to-end network latency and bandwidth between any pair of computing sites. The tables show five of the GUSTO sites: NASA AMES, Argonne National Lab, University of Indiana, USC-ISI, and NCSA.

The directory service takes into account the current network load, including the load imposed by the application. If the paths between two distinct node pairs share a common link, the bandwidth of the common link is divided among these communicating pairs.

3.2. Communication Model

We use a communication model to analytically represent the network performance. Using information about the application's communication pattern and the performance parameters provided by the directory service, the communication model can estimate the time for individual node-to-node communication events.

Consider a typical metacomputing system, such as shown in Figure 1. A path between compute nodes typically includes links from multiple networks of different bandwidths. For example, in Figure 1, a message from a node in Site 1 to a node in Site 2 would pass through the local network at both sites and the long haul link which interconnects these geographically distributed sites.

Our communication model represents the network performance between any processor pair (P_i, P_j) using two parameters: a start-up cost T_{ij} and a data transmission rate B_{ij}. The time for sending a m byte message between these nodes is then given by $T_{ij} + \frac{m}{B_{ij}}$. The two parameters abstractly represent the total time for traversing all the links on the path between P_i and P_j. The model ignores the negligible delays incurred by contention at intermediate links and nodes on the path between P_i and P_j.

Our model focuses on the effective network performance at the application layer. We assume the availability of end-to-end send and receive communication routines, which can be invoked between any pair of processor nodes. Since the details of network topology, routing, and flow control policies are not visible to the application, our model does not incorporate these parameters.

A similar communication model has been widely used for tightly-coupled distributed memory systems with good results [27]. In metacomputing systems, typical values for the start-up cost could be in the range of 10 to 50 ms, while typical values for the bandwidth could be in the range of kb/s to hundreds of Mb/s.

The model assumes that a node is allowed to simultaneously participate in at most one send and one receive operation. When a node has multiple messages to send, it performs these send operations one after another. Current hardware and software do not easily enable multiple messages to be transmitted simultaneously. Software support for non-blocking and multithreaded communication sometimes allow applications to initiate multiple send and receive operations. However, all these operations are eventually serialized by the single hardware port to the network. Our model accurately represents this phenomenon.

If multiple nodes simultaneously send to any node P_j, we say that node contention occurs at P_j. The model assumes that these messages are received one after the other at P_j. The validity of this assumption can be seen by examining the events involved in a message transmission from P_i to P_j. A control message is first transmitted by P_i. The actual data is sent only after this control message is acknowledged by P_j. If P_j is busy receiving from a different node, it sends the acknowledgement to P_i only after completing the previous receive operation.

3.3. Timing Diagrams

We use timing diagrams to represent communication schedules for given network characteristics and a communication pattern. Examples of timing diagrams for all-to-all personalized communication with 5 processors are shown in Figures 4 and 7. The diagram consists of P columns, one per processor. The vertical axis represents time. The communication events in column i represent the messages sent from processor P_i. The rectangle labeled j in column i represents the message sent from P_i to P_j [1]. The height of the rectangle denotes the time for the communication event [2]. Once the message sizes and the values of T_{ij} and B_{ij} between all processor pairs are known, the heights of all the rectangles can be determined. Thus, the timing diagram inherently absorbs the heterogeneity in network parameters and message lengths.

3.4. Scheduling Algorithms

Our communication scheduling algorithms determine the positions of the individual events in the timing diagram so that the completion time is minimized. A valid communication schedule must satisfy the following conditions – since a node cannot send multiple message simultaneously, none of the rectangles in a column can overlap in time. Similarly, since multiple simultaneous receive events are not permitted at a processor, all the rectangles with the same label j must have mutually disjoint time intervals.

We do not consider "indirect" schedules where messages from different sources are combined at intermediate nodes and then forwarded to common destinations. This is because such combine-and-forward schemes increase the volume of traffic to be communicated. Since data in metacomputing applications is often extremely voluminous, this can lead to large communication costs.

We also do not allow messages to be partitioned. Since the start-up overhead is incurred for each message transmission, such a partitioning would increase the start-up overheads.

The next section presents our scheduling algorithms for the all-to-all personalized communication pattern.

4. Scheduling Algorithms for Total Exchange

In this section, we develop communication scheduling algorithms for total exchange, or all-to-all personalized communication. We briefly describe a well known communication algorithm for this problem. Section 5 shows the performance improvements achieved by our new algorithms over this algorithm.

[1] A receive schedule can be similarly constructed, where the communication events in column i represent messages received by processor P_i.

[2] The width of the rectangle does not have any significance.

4.1. Communication Pattern and Scheduling Complexity

All-to-all personalized communication occurs very frequently in HPC applications. For example, consider a two-dimensional matrix which is initially distributed by rows among the processors. If the matrix must be transposed so that the final distribution has columns on each processor, the resulting communication pattern is an all-to-all personalized communication. Here, each compute node has a distinct message for every other node in the system. For a P processor system, this communication pattern consists of $O(P^2)$ communication events. The message sizes between all pairs of nodes are not necessarily the same. When the network is heterogeneous, the individual communication events in the timing diagram will have different lengths. These communication events in the timing diagram must be efficiently scheduled. The goal is to reduce the *completion time* t_{max} of the communication schedule, *i.e.* the time at which the last communication event is completed. Observe that the completion time of the schedule cannot be less than the summation of send times or receive times at any processor, whichever is larger. This is therefore a *lower bound* t_{lb} on the completion time. To analyze the complexity of this communication scheduling problem, we first state it as a decision problem.

TOT_EXCH: Given a distributed heterogeneous system with P processors (P_0, ..., P_{P-1}), a deadline τ, and a $P \times P$ communication matrix $\mathbf{C}$, where $\mathbf{C}_{i,j}$ is the time for the communication event from P_j to P_i, $0 \leq i, j < P$ is there a communication schedule with completion time less than or equal to τ?

Theorem 1: *TOT_EXCH is NP-Complete for* $P > 2$.

Proof: The theorem can be proved by transformation from the open shop scheduling problem. The problem [5, 11] consists of m machines and n jobs. Each machine i performs task $t_{j,i}$ of job j. The execution time of all tasks are given in an $n \times m$ matrix. There are no dependences among the tasks of a job. Hence there are no restrictions on the sequence in which these tasks are to be executed. However, any machine can work on only one job at a time and any job can be processed by only one machine at a time. The goal is to schedule the tasks on the machines so as to minimize the finish time. The problem is known to be *NP*-Complete for $m > 2$ [11]. Details of our proof can be found in [3]. □

4.2. Baseline Algorithm

Since the total exchange communication scheduling problem is *NP*-Complete, we have developed heuristic algo-

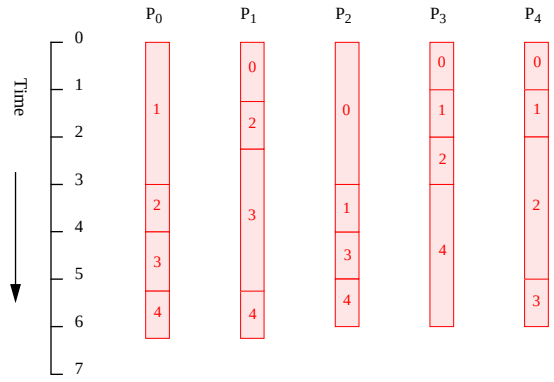

Figure 3. Example problem.

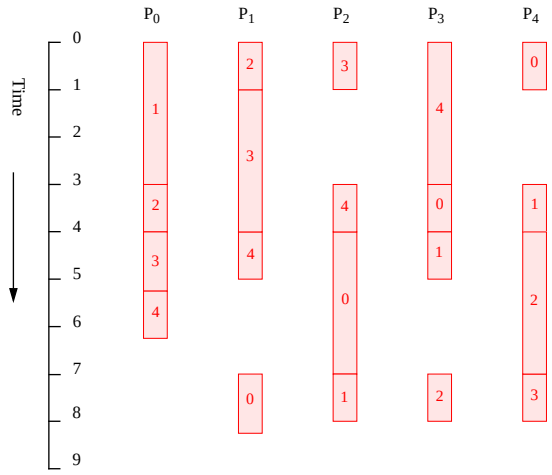

Figure 4. Schedule generated by baseline algorithm.

rithms. As a *baseline algorithm* for performance comparison with our heuristic algorithms, we shall use the caterpillar algorithm, which is widely used in tightly coupled homogeneous systems. This generates a schedule with P steps. In step j, $(0 \leq j < P)$, each compute node $P_i (0 \leq i < P)$ sends a message to $P_{(i+j) mod P}$ [13]. Such a schedule does not incur any node contention in a homogeneous system, when the message sizes and network bandwidths are uniform. This is because all the communication events have the same duration, *i.e.* all the rectangles in the timing diagram have the same height. An important disadvantage of the baseline algorithm is that it derives a fixed schedule, which is not adaptive to variations in message lengths or network performance.

We illustrate our scheduling techniques with a running example. Figure 3 shows an example communication problem, represented in the timing diagram formalism. The unscheduled communication events originating from each processor are shown in increasing order of destination processor number.

In our examples, we assume that the diagonal entries in **C** are zeroes. This is valid, since the time for a local memory copy operation is negligible in comparison with the time for sending messages over the heterogeneous network.

Using the baseline algorithm, the schedule shown in Figure 4 is derived. Observe that the longer communication events in the earlier steps cause the later communication steps to be delayed.

Theorem 2: Performance Bound for the Baseline Algorithm.

1. *The completion time t_{max} of the baseline schedule is always within $\frac{P}{2}$ times the lower bound t_{lb}.*

2. *The above bound is tight, i.e. there exist instances where the baseline schedule takes $\frac{P}{2}$ times the lower bound.*

Proof: We first introduce the notion of a dependence graph **DG** for a given schedule. This is a directed graph with P^2 nodes, one for each communication event. A directed edge is present from node i to node j if there exists a sequential dependency between the corresponding communication events in the schedule.

Figure 5 shows the dependence graph for the baseline schedule with 5 processors. Column i contains all the communication events sent from Processor P_i, in the order that they appear in the schedule. Observe that the edges in **DG** are of two kinds: (i) vertical edges between adjacent nodes in the same column, and (ii) diagonal edges between nodes in adjacent columns. A correspondence exists between the graph **DG** and the communication matrix **C**. Each node in **DG** corresponds to an entry in **C**. If a vertical edge exists between two nodes, then these correspond to entries in the same column of **C**. If a diagonal edge is present, then the nodes correspond to entries in the same row of **C**.

Each path in the **DG** for the baseline schedule contains P nodes and $P-1$ edges. The completion time is equal to the weight of the longest path in the graph. Let $t_1, t_2, \ldots, t_P$ be the nodes in the longest path. Then,

$$t_{max} = t_1 + t_2 + \ldots + t_P \quad (1)$$

Since adjacent nodes in any path belong to the same row or column of **C**, and from the definition of t_{lb},

$$t_{lb} \geq max\{(t_1 + t_2), (t_3 + t_4), \ldots, (t_{P-1} + t_P)\} \quad (2)$$

We can now rewrite Eq (1) as

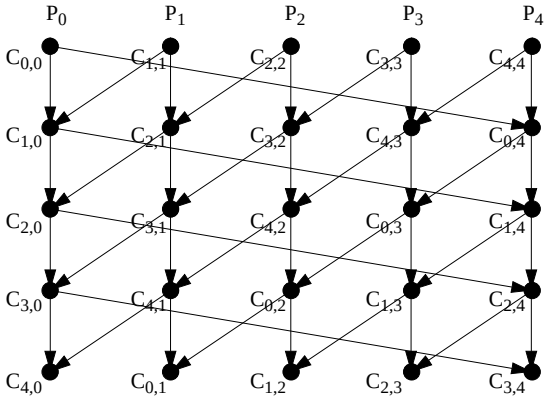

Figure 5. Dependence graph for the baseline schedule.

$$t_{max} = (t_1 + t_2) + (t_3 + t_4) + \ldots + (t_{P-1} + t_P)$$
$$t_{max} \leq \frac{P}{2} \times max\{(t_1 + t_2), \ldots, (t_{P-1} + t_P)\} \quad (3)$$

From Eq (2) and Eq (3)

$$t_{max} \leq \frac{P}{2} \times t_{lb} \quad (4)$$

To prove the tightness of the bound, consider the following communication matrix:

$$\mathbf{C} = \begin{bmatrix} \epsilon & \epsilon & \epsilon & \epsilon \\ \epsilon & 1 & \epsilon & \epsilon \\ 1 & 1 & \epsilon & \epsilon \\ 1 & \epsilon & \epsilon & \epsilon \end{bmatrix}$$

Each dependence path consists of 4 elements. The first element is always on the diagonal. Adjacent elements in the path are either in the same row or the same column. In the former case, the i^{th} element in the path is to the immediate left of the $i-1^{th}$ element. In the latter case, the i^{th} element is immediately below the $i-1^{th}$ element. For this example, the critical path contains all the unit-time entries, and takes 4 units of time. The lower bound is $2 + 2\epsilon$ [3].

Therefore,

$$\frac{t_{max}}{t_{lb}} = \frac{4}{2 + 2\epsilon} \approx 2 \quad (5)$$

□

[3] ϵ is an arbitrarily small numer.

4.3. Matching-Based Scheduling Techniques

We present two matching-based scheduling techniques for the total exchange problem. The first technique finds a series of maximum weight matchings in a bipartite graph. We also consider the variation wherein minimum matchings are found.

Our algorithm partitions the $P \times P$ communication events into P independent steps using graph matching algorithms. For a P node system, we construct a bipartite graph with P vertices on each side. The edge from v_i on the left side to v_j on the right side is assigned a weight equal to the time for the communication event from P_i to P_j. Thus, there are $O(P^2)$ edges in the bipartite graph. A complete matching in such a graph consists of P edges, and corresponds to a permutation of $(P_0, \ldots, P_{P-1})$. Such a matching can therefore represent a valid communication step, without contention at any processor. Well known algorithms exist for finding a maximum weight complete matching [17]. This is identical to the linear assignment problem. The complexity of this algorithm is $O(P^3)$. Our algorithm therefore consists of finding a maximum weight complete matching in the graph, deleting the edges of the matching from the graph, and then repeating the process until P such matchings have been found. Thus, the total complexity is $O(P^4)$. Although the schedule finds the communication events step by step, the communication phase does not impose a synchronization among the processors after each step. A communication event will begin whenever the sending and receiving processors are both ready.

In theory, the completion time of the matching based techniques can be $\frac{P}{2}$ times the lower bound. We can prove a result similar to part (i) of Theorem 2. In practice, the performance is significantly better, and the bound is therefore not tight. Unlike the fixed schedule derived by the baseline algorithm, our matching based schedule is adaptive. When the lengths of the communication events change with variations in network performance, the algorithm finds a different schedule with a low completion time. Section 5 presents simulation results.

For the example of Figure 3, our adaptive maximum matching algorithm derives the schedule shown in Figure 6. The matching technique groups together communication events with similar length, thereby reducing the idle cycles. Figure 6 is an optimal schedule for this example, since there exists a processor (P_1 or P_2) which is busy during the entire schedule.

4.4. Greedy Technique

The greedy technique is an approximation to the matching technique, with a lower computational complexity. Initially, the communication events within each processor are

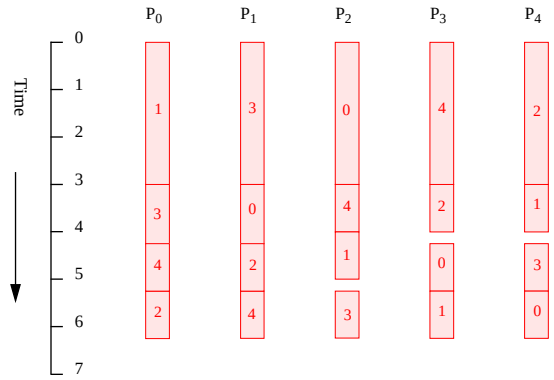

Figure 6. Schedule generated by a series of maximum matchings.

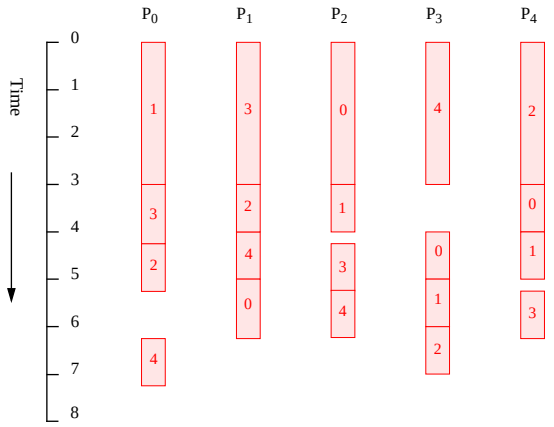

Figure 7. Schedule generated by the greedy algorithm.

rank ordered in decreasing order of communication time. A series of communication steps is then composed. In composing each step, we traverse the rank ordered list of every processor, with the goal of finding a destination processor. If a destination processor is found, it will be the first processor in its list that has not been selected by this processor in a previous step, and that is not the destination of another processor in the same step. If the end of the list is reached without finding a destination, the processor idles during this step, and we proceed to the next processor. Due to such incomplete steps, the total number of steps could be larger than P. To ensure fairness, a processor which was idle in any step will be the first to pick the destination processor in the next step. If there was no idle processor in a step, the last processor in any step will be the first in the next step. The greedy algorithm has a computational complexity of $O(P^3)$. From the description above, it is clear that the greedy algorithm is adaptive to the lengths of the communication events. For the previous example, the communication schedule derived by the greedy algorithm is shown in Figure 7.

4.5. Open Shop Technique

Since our communication problem has similarities to the open shop scheduling problem, we have developed a scheduling algorithm based on a heuristic derived for the open shop problem [22]. Other approximate algorithms for the open shop problem are given in [4].

Each processor is considered as two independent entities, a sender and a receiver. The following data structures are maintained by the algorithm:

- For each sender $i, 0 \leq i < P$, a set R_i of receivers is maintained. Initially, this consists of all receivers to which i must send a message. When a communication event is scheduled, the appropriate receiver is deleted from the receiver set.
- The P-element arrays *sendavail* and *recvavail* contain information about the availability of the corresponding senders and receivers. For example, the i^{th} element of *sendavail* specifies the earliest time at which sender i can participate in future send operations. All elements of both these arrays are initialized to 0.

The algorithm proceeds as follows:

- Whenever a sender i becomes available at time *sendavail[i]*, its receiver set R_i is scanned, and the earliest available receiver j is selected. The communication event from i to j is scheduled to begin at time *t=max(sendavail[i], recvavail[j])*. *sendavail[i]* and *recvavail[j]* are assigned the value $t + \mathbf{C}[j, i]$, since the sender i and receiver j will be busy until this time. Further, j is deleted from R_i.
- If multiple senders become available at the same time (for example, at time 0), they are processed in an arbitrary order. However, all senders that become available at time t are processed before any senders that become available at a later time. The algorithm maintains a list of senders in increasing order of their time of availability.
- Whenever a sender is finished with all its operations, it is deleted from this list. The algorithm terminates when all the senders are thus deleted.

The total number of communication events to be scheduled is $O(P^2)$. The scheduling of each event takes $O(P)$

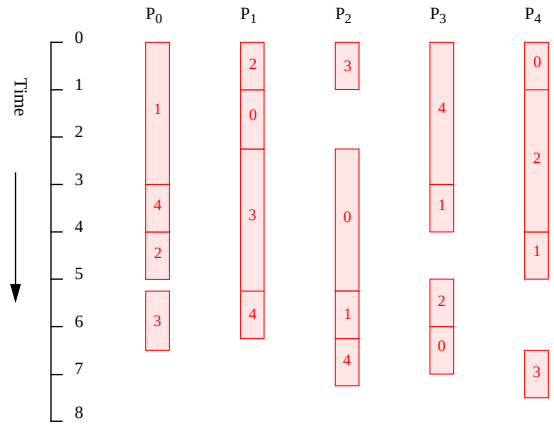

Figure 8. Schedule generated by the open shop algorithm.

time, since the elements of the corresponding receiver set must be scanned. The algorithm therefore runs in time $O(P^3)$.

Observe that the algorithm is a greedy one. At any time a sender is free, the heuristic assigns a communication event to any of the elements in its receiver set. Idle cycles are inserted in a sender's schedule only if none of its potential receivers are available. For our running example, the schedule derived by this heuristic is shown in Figure 8.

Theorem 3: *The open shop heuristic algorithm is guaranteed to find a communication schedule whose completion time is within twice the lower bound.*

Proof: Assume, without loss of generality, that the last sender to finish all its transmissions is i. Let j be the receiver that i sends its last message to. It can be deduced that during the idle cycles in sender i's schedule, receiver j must have been always busy. If this were not the case, the algorithm would have scheduled the communication event from i to j at this time. Thus, we can conclude that the sum of the idle cycles in sender i's schedule is bounded by the total time for communication events having j as the receiver, *i.e.*, the sum of elements in row j of **C**. The completion time is the sum of the total time for send events from sender i, *i.e.*, the sum of elements in column i of **C**, and the idle cycles in sender i. Thus, the completion time is at most the sum of a row and a column in the communication matrix **C**, and is hence within twice the lower bound. □

5. Experimental Results

We have developed a software simulator that executes the scheduling algorithms discussed in Section 4, and calculates the completion time for each of them. The simulator accepts processor count and communication times as input, and generates the schedules based on these techniques. We have used this simulation tool to evaluate the baseline, maximum matching, minimum matching, greedy, and open shop scheduling techniques.

The simulator generates random performance characteristics for pairwise network performance, using information from the GUSTO directory service as a guideline. The communication matrix **C** can then be generated for any fixed message size. We have selected message sizes of 1kB, 1MB, and a random mix of these two sizes. In our experiments, we assume that the diagonal entries in **C** are zeroes. This is valid, since the time for a local memory copy operation is negligible in comparison with the time for sending messages over the heterogeneous network. The scheduling techniques are then applied to this communication matrix. Results for the different message sizes are shown in Figures 9, 10, and 11. Systems with up to 50 processors were considered.

Figure 12 considers a scenario when some of the processors are designated as servers. The message sizes from the servers to the other (client) processors are assumed to be large. The message sizes between the servers themselves and also between the client processors are small. This is typical in multimedia applications, where images and video clips reside on servers, and are accessed by other processors. In our experiment, 20% of the processors are assumed to be servers. Data is also assumed to be partitioned over the servers, so that the load on the servers is balanced. It can be seen that the baseline algorithm performs very poorly in such scenarios. Our algorithms perform 2 to 5 times faster than the baseline in these examples.

The graphs clearly show the performance improvements that can be achieved by our communication scheduling techniques. The open shop algorithm finds schedules that are very close to the lower bound, often within 2%, and always within 10 %. The maximum and minimum matching based techniques find schedules with comparable completion times. These are within 15% of the lower bound. The schedules generated by the greedy algorithm are within 25% of the lower bound. The schedules generated by the baseline algorithm sometimes take upto 6 times longer than the lower bound. Based on our results, the open shop algorithm achieves the best performance.

6. Enhancements and Future Research

In the previous sections, we presented our approach for developing communication scheduling techniques that are adaptive to network performance variations. To the best of our knowledge, this is one of the early efforts in formalizing communication problems relevant to network-based computing. Several exciting research issues remain to be explored. In this section, we discuss some future research di-

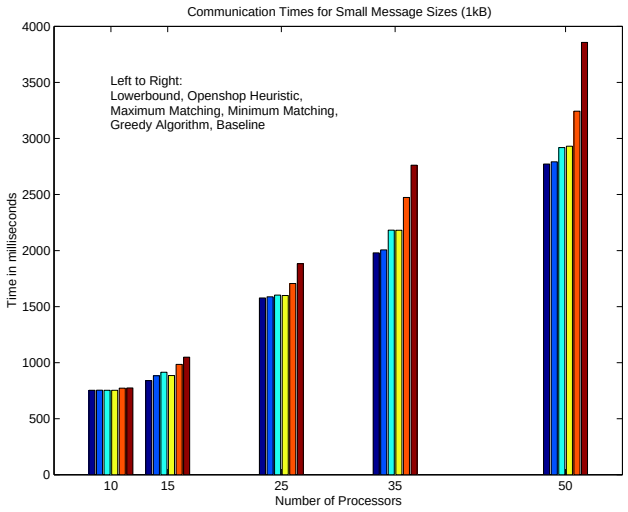

Figure 9. Simulator results for all-to-all personalized communication with small message sizes.

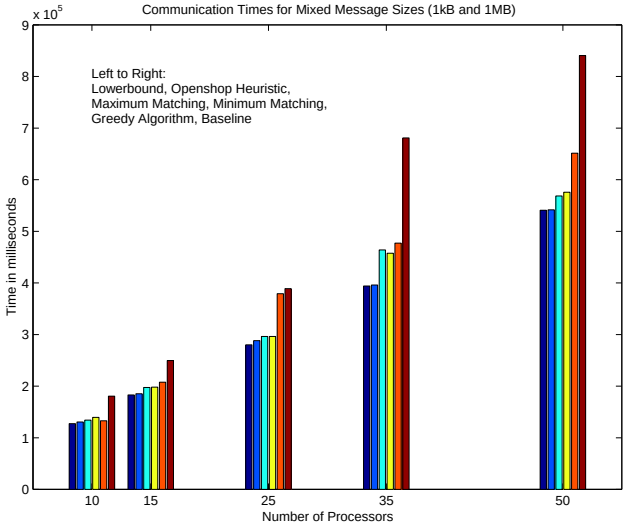

Figure 11. Simulator results for all-to-all personalized communication with mixed message sizes.

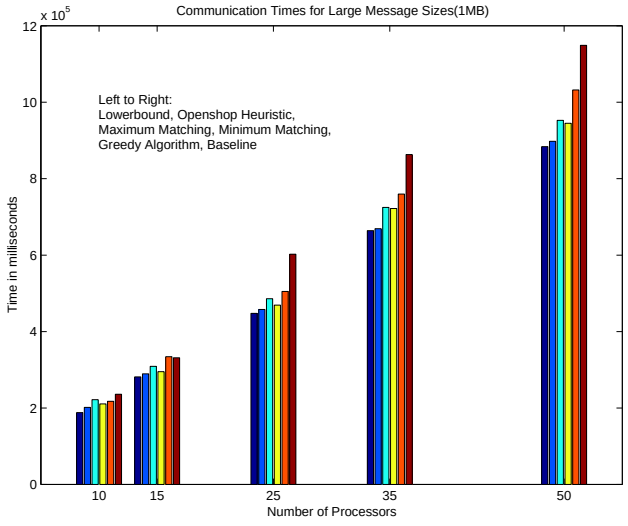

Figure 10. Simulator results for all-to-all personalized communication with large message sizes.

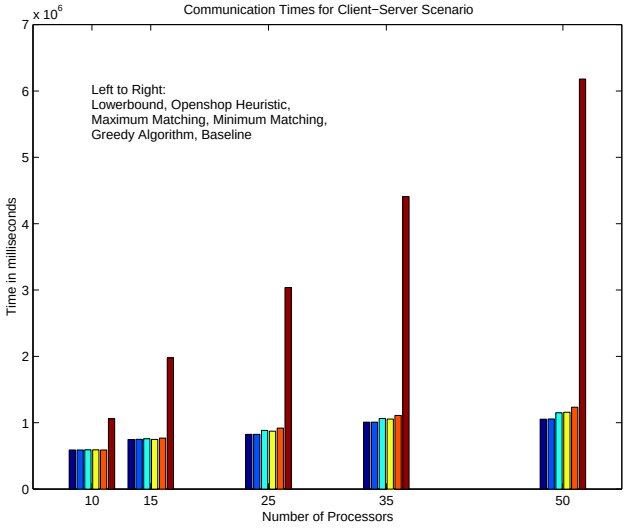

Figure 12. Simulator results for all-to-all personalized communication when 20% of the processors are servers. Servers send large messages to their clients.

rections that are motivated by the need for network-aware communication scheduling.

6.1. Enhancing the Model

Our scheduling algorithms were developed using a simple yet effective communication model. In Section 3, we mentioned the assumptions made by our model, and the validity of these assumptions. Enhanced versions of the model can be formulated by relaxing some of these assumptions. For example, one restriction is that a processor can send and receive only one message at a time. This restriction can be relaxed in two ways.

When multiple messages arrive at a node, we can assume that the messages are received in an interleaved fashion. For example, the use of multithreading allows multiple simultaneous communication events in Nexus. An additional parameter α can be introduced for the overhead incurred in context switching between the multiple receiving threads. Thus, if t_1 and t_2 are the times for individually receiving two messages, the total time for receiving them simultaneously would be $(1 + \alpha)(t_1 + t_2)$.

It could also be assumed that a finite buffer space is available at nodes to receive messages. When multiple messages arrive at a node, one of the messages is received by the application, while the others are queued in the buffer. The sending nodes do not wait until the receive operation is complete, but only until the message is stored in the buffer. If the buffer is full, the sender must wait until adequate free space is created in the buffer.

6.2. Incremental Dynamic Scheduling

The communication schedules presented in Section 4 are computed at run-time based on information obtained from the directory service. In many sensor-based applications, a series of continuously arriving data sets are processed in an identical manner. In such cases, the overhead for repeatedly calculating the communication schedule at run-time can be expensive, especially when the number of processors is large. It is therefore necessary to develop scheduling techniques which have significantly lower computational costs.

An incremental approach would be one way to reduce the complexity of deriving dynamic communication schedules. Here, a communication schedule is computed once, either at compile time or during the first run-time occurrence. At each subsequent invocation, the incremental algorithm must refine this communication schedule to find a new communication schedule. The algorithm would query the directory service regarding changes in the bandwidths. In this context, the research problem is that of developing fast algorithms for refining an existing communication schedule.

6.3. Enhancing Adaptivity of the Schedules

In some scenarios, the lengths of all communication events may not be known even when the communication is started. This could happen because variations in network performance are so rapid that significant changes could occur within the duration of the communication schedule. In such cases, an initial communication schedule can be derived using estimates of the communication times. The schedule can then be modified at intermediate *checkpoints*. At these checkpoints, processors decide whether the difference between the estimated time and actual time is large enough to require rescheduling. The checkpoints could be defined in different ways: after each communication event is complete ($O(P)$ checkpoints), or after half the remaining communication events are complete ($O(\log P)$ checkpoints), and so on.

6.4. Scheduling with Critical Resources or QoS Constraints

We have discussed communication schedules where the goal is to minimize the completion time. In many scenarios, other cost measures are also important. For example, one of the processors in the heterogeneous system could be a critical resource (*e.g.,* an expensive supercomputer). The schedule should complete the communication events of this processor as early as possible, even if it delays the other processors.

Quality of Service (QoS) requirements in some applications can introduce other variations in the problem formulation. For example, data forwarding and data staging problems arise in the BADD project [6]. The QoS parameters associated with each message are deadlines and priorities. The communication schedule must ensure that data items reach their destinations by the specified real-time deadlines. When multiple communication events contend for a communication link, the scheduling algorithm must sequence them based on their respective deadlines and priorities.

7. Conclusion

In this paper, we have developed a uniform framework for communication scheduling in heterogeneous network-based systems. The framework consists of a directory service, a communication model, timing diagrams, and scheduling algorithms. We discussed our approach for the design of adaptive communication techniques, and applied it to the problem of all-to-all personalized communication. Although this problem has been thoroughly researched for homogeneous systems, we showed that well known algorithms perform poorly in the presence of network heterogeneity. We developed algorithms based on bipartite graph

matching, and a heuristic algorithms based on Open shop scheduling. We showed that our algorithms performed significantly better than a well known homogeneous communication scheduling algorithm. Our algorithms are adaptive and execute at run-time, based on network performance information obtained from the directory service. To the best of our knowledge, this is one of the early efforts in formalizing communication problems in a distributed heterogeneous computing environment. Our paper also discusses several new research problems related to communication scheduling in network-based systems, that arise due to the unique features of such environments. These include communication scheduling with QoS constraints and techniques to reduce the complexity of the scheduling algorithm.

Acknowledgments

We thank Roy Jonker of MagicLogic Optimization, Inc. for his public domain program to solve the Linear Assignment Problem. This routine was used to find the matchings in our algorithms. We also thank Craig Lee and Paul Stelling of The Aerospace Corporation and the members of the MSHN project for helpful technical suggestions.

References

[1] R. Armstrong, D. Hensgen, and T. Kidd. The relative performance of various mapping algorithms is independent of sizable variances in runtime predictions. In *Proc. Heterogeneous Computing Workshop*, pages 79–87, March 1998.

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

[3] P. B. Bhat, V. K. Prasanna, and C. S. Raghavendra. Adaptive communication algorithms for distributed heterogeneous systems. *Manuscript*, Dept. of EE-Systems, University of Southern California, May 1998.

[4] H. Brasel, T. Tautenhahn, and F. Werner. Constructive heuristic algorithms for the open shop problem. *Computing*, 51:95–110, 1993.

[5] P. Brucker. *Scheduling Algorithms*. Springer, 1995.

[6] DARPA ISO Web Page for the BADD Program. *https://www.iso.darpa.mil/ WD@27000.cgi?get+iso::Office+Information_System +WDI_i_home_frames*.

[7] T. DeWitt, T. Gross, B. Lowekamp, N. Miller, P. Steenkiste, J. Subhlok, and D. Sutherland. ReMoS: A resource monitoring system for network-aware applications. Technical Report CMU-CS-97-194, School of Computer Science, Carnegie Mellon University, Dec 1997.

[8] S. Fitzgerald, I. Foster, C. Kesselman, G. von Laszewskiand, W. Smith, and S. Tuecke. A directory service for configuring high-performance distributed computations. In *Proc. 6th IEEE Intnl. Symp. on High Performance Distributed Computing*, pages 365–375, 1997.

[9] I. Foster and C. Kesselman. Globus: A metacomputing infrastructure toolkit. *Intl. Journal of Supercomputer Applications*, 11(2):115–128, 1997.

[10] Globus Web Page. *http://www.globus.org*.

[11] T. Gonzalez and S. Sahni. Open shop scheduling to minimize finish time. *Journal of the ACM*, 23(4):665–679, Oct. 1976.

[12] A. S. Grimshaw and W. A. Wulf. Legion – a view from 50,000 feet. In *Proc. Fifth IEEE Intl. Symp. on High Performance Distributed Computing*, August 1996.

[13] L. H. Jamieson, P. T. Mueller, and H. J. Siegel. FFT algorithms for SIMD parallel processing systems. *Journal of Parallel and Distributed Computing*, 3(1):48–71, March 1986.

[14] J. Kim and D. J. Lilja. Exploiting multiple heterogeneous networks to reduce communication costs in parallel programs. In *Proc. Heterogeneous Computing Workshop*, pages 83–95, April 1997.

[15] J. Kim and D. J. Lilja. Utilizing heterogeneous networks in distributed parallel computing systems. In *Proc. Sixth IEEE Intl. Symp. High Performance Distributed Computing*, 1997.

[16] H. Korab and M. D. Brown, *Eds*. *Virtual Environments and Distributed Computing at SC '95: GII Testbed and HPC Challenge Applications on the I-WAY*. ACM/IEEE Supercomputing '95, 1995.

[17] E. Lawler. *Combinatorial Optimization: Networks and Matroids*. Holt, Rinehart and Winston, 1976.

[18] Legion Web Page. *http://legion.virginia.edu*.

[19] Y. W. Lim, P. B. Bhat, and V. K. Prasanna. Efficient algorithms for block-cyclic redistribution of arrays. *Algorithmica*, to appear.

[20] M. Maheswaran and H. J. Siegel. A dynamic matching and scheduling algorithm for heterogeneous computing systems. In *Proc. Heterogeneous Computing Workshop*, pages 57–69, March 1998.

[21] MSHN Web Page. *http://www.cs.nps.navy.mil/mshn*.

[22] D. B. Shmoys, C. Stein, and J. Wein. Improved approximation algorithms for shop scheduling problems. *SIAM J. Comput.*, 23(3):617–632, June 1994.

[23] L. Smarr and C. E. Catlett. Metacomputing. *Commns. of the ACM*, 35(6):45–52, June 1992.

[24] M. Tan, M. D. Theys, H. J. Siegel, N. B. Beck, and M. Jurczyk. A mathematical model, heuristic, and simulation study for a basic data staging problem in a heterogeneous networking environment. In *Proc. Heterogeneous Computing Workshop*, pages 115–129, March 1998.

[25] 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. In *Proc. Sixth IEEE Intl. Symp. on High Performance Distributed Computing*, 1997.

[26] VDCE Web Page. *http://www.atm.syr.edu/projects/vm/index.html*.

[27] C.-L. Wang, P. B. Bhat, and V. K. Prasanna. High-performance computing for vision. *Proceedings of the IEEE*, 84(7):931–946, July 1996.