Configurable Communication Middleware for Clusters with Multiple Interconnections

Nader MOHAMED†(a), Jameela AL-JAROODI†, and Hong JIANG†, Nonmembers

SUMMARY High performance scientific and engineering applications running on clusters have different communication requirements. Current cluster configurations typically provide multiple network interfaces per node and multiple interconnections among nodes. However, transport protocols such as TCP do not utilize existing multiple network interfaces to enhance communication performance. This paper introduces a new configurable communication model utilizing multiple interconnections. The model adds mechanisms to manage and enhance the overall communication performance of clusters. These configurations include the use of parallel message transfers, the separation of the transfer channels between small messages and large messages, and load balancing among the channels. The main advantages of the model are: (1) providing a flexible, enhanced network infrastructure, (2) hiding the technical details of the heterogeneous network resources from the applications, and (3) providing an easy and flexible way to extend the network capacities for specific nodes. To illustrate the advantages and performance enhancements of the model, a prototype was implemented to experimentally evaluate the cluster network performance, which showed considerable gains.

key words: communication middleware, cluster computing, multiple networks, configurable networking

1. Introduction

One of the main advantages of clusters [3] and cluster computing is the extensible infrastructure where system resources such as processors, memory, storage, and network interfaces can easily be augmented or changed. In addition, cluster nodes can be either homogeneous or heterogeneous in terms of their resource capabilities, platforms, operating system, and function designations (e.g., storage, computation, or visualization nodes).

From the network viewpoint, several cluster performance properties stand to benefit significantly from expansions and enhancements. One such property is message transfer time; another is overall throughput applications achieve collectively during their life spans. In addition, the latency of transferring a small message is an important property that can also be enhanced. However, applications benefit differently from performance enhancements, depending on their reliance on different properties in varying situations. For example, bulk data transfer for parallel applications such as image processing and data mining requires high bandwidth, while multi-node graphical visualization requires low latency and high throughput among participating nodes. However, most cluster communication software and hardware infrastructures do not support easy network properties or quality-of-service (QoS) reconfigurations to accommodate the different performance requirements.

Currently, there is growing interest in utilizing multiple networks to increase bandwidth and throughput and enhance latency and reliability among cluster nodes. For example, channel bonding [2] binds multiple Ethernet network interfaces to appear as a single network interface with high throughput. However, it generally does not provide flexible mechanisms to adapt to varying cluster configurations and applications requirements. In this paper, we develop a generic, flexible and transparent cluster networking model (MuniCluster) to enhance network properties and to alleviate the limitations of the existing approaches. MuniCluster presents a flexible communication infrastructure that allows applications to transparently utilize multiple network resources such as network interface cards and network switches in various arrangements based on their communication requirements. These configurations include parallel message transfer, separate channels for small messages and large messages, and load balancing among multiple network channels. Furthermore, MuniCluster enables heterogeneous clusters with varying capacities and numbers of network interface cards to fully utilize all of their network resources. One salient feature of the model is the enhancement of the quality of service (QoS) for small messages, which in this context refers to the average transfer time of small messages over networks with varying workload.

In this paper, we outline the needs for flexible cluster communication, in addition to discussing some related work in Sect. 2. Section 3 describes the architecture, functions, load balancing techniques, and operation modes of our model. Then, in Sect. 4, we discuss different possible configurations for the model and measure their performance. Section 5 concludes the paper.

2. Backgrounds and Related Work

In a homogeneous cluster, all nodes have similar resources and processing powers and many applications can benefit from this uniformity to execute efficiently. When applications require uniform and balanced access to all nodes, it becomes important to enhance the network properties of the whole cluster. On the other hand, non-uniformity in a cluster can result from non-uniform distribution or assign-
ment of processing power, functionalities, and network or storage capacities among the nodes. However, most of the non-uniformity in current clusters is dictated by the nature of the applications using them. For example, some clusters have a number of nodes specifically assigned for high performance applications, while the rest of the nodes are shared by different users and multiple applications, or designated for file/storage servers. To accommodate such differences in characteristics and network requirements, it may be feasible to utilize a varying number of network interfaces on each node. For example, consider a cluster with 16 nodes connected through a 24-port Fast Ethernet switch and share a fast RAID file system (e.g., 400 Mbps peak read/write throughput) via the same switch. Assume that the applications running on the cluster may not need more than 100 Mbps for communication among the nodes. However, the high throughput of the RAID will never be reached for the file system due to the limitation of the fast Ethernet Interface card. One simple but expensive solution is to upgrade the network to high performance network such as Gigabit Ethernet or Myrinet. Another possible solution is to add three more fast Ethernet cards to the RAID node and connect them to the same cluster switch. This provides an inexpensive solution; however, it requires management services to hide the multiple cards and provide uniform access mechanisms.

Another problem in Cluster communication is QoS for small messages. Consider a cluster with a Gigabit Ethernet network connecting all nodes. Executing high performance applications on the cluster while running management services such as rsh and FTP will negatively affect the performance of the applications. One possible solution is to install another low-cost network such as Fast Ethernet to be used by all non-performance-critical applications and services, while the high performance applications utilize the Gigabit network. This is a static enhancement; however, having multithreaded high performance applications on the cluster, there may be cases where one thread starts to send a large message while at the same time a second thread sends a small message. The transfer time of the small message will be high since the large message will fill the network queues, although at the same time the additional fast Ethernet interface may be free. Another example is a cluster connected by two Fast Ethernet networks and running software-based Distributed Shared Memory (DSM). In DSM, small control messages such as invalidations are exchanged frequently for the consistency and coherence protocols. In addition, larger messages are exchanged among nodes for data replications and information exchange. In general, to have a good DSM performance the control messages need higher priority than any other messages. However, most of the current communication software used for DSM in clusters cannot effectively support this functionality. For some networks, sending large messages can severely slow down small messages sharing the same network card. This is caused by many factors such as transport layer multiplexing rate, system resource capacities, network bandwidth, capacity of switch resources, and switch scheduling policy.

Most of existing solutions for multiple networks in clusters do not provide configurable communication. They require uniform interconnections and work on homogeneous systems and networks. In addition, most of these solutions do not provide any QoS for small messages. Our previous work, Munisocket [8]-[10], works on heterogeneous systems and networks. However, it does not provide configurable, flexible, and optimized network enhancements for clusters and does not support QoS for small messages. Table 1 is a brief comparison of the characteristics of these techniques in terms of the different configurations of possible property enhancements.

### Table 1 Summary of network properties enhancements solution approaches.

<table>
<thead>
<tr>
<th>Enhancement Technique</th>
<th>Throughput</th>
<th>Short Msg. QoS</th>
<th>Uniform</th>
<th>Non-Uniform</th>
<th>System Heterogeneity</th>
<th>Network Heterogeneity</th>
</tr>
</thead>
<tbody>
<tr>
<td>Channel bonding [2]</td>
<td>✓</td>
<td>✗</td>
<td>✗</td>
<td>X</td>
<td>X</td>
<td>✗</td>
</tr>
<tr>
<td>FNMP [15]</td>
<td>✗</td>
<td>✗</td>
<td>✗</td>
<td>✗</td>
<td>✗</td>
<td>✗</td>
</tr>
<tr>
<td>Heterogeneous Networks [6]</td>
<td>✓</td>
<td>✗</td>
<td>✗</td>
<td>✗</td>
<td>N/A</td>
<td>✗</td>
</tr>
<tr>
<td>Multitail [4]</td>
<td>✓</td>
<td>✗</td>
<td>✗</td>
<td>✗</td>
<td>N/A</td>
<td>✗</td>
</tr>
<tr>
<td>Munisocket [8][9][10]</td>
<td>✓</td>
<td>✗</td>
<td>✗</td>
<td>✗</td>
<td>✗</td>
<td>✗</td>
</tr>
</tbody>
</table>

*: only outbound

3. The Proposed Model

To have flexible enhancements for cluster network properties, we propose a new configurable communication software model that extends the functions of the Socket, while providing the same interfaces. The Socket API [14] provides basic primitives such as send and receive for distributed applications to communicate. Many higher-level mechanisms such as remote procedure call (RPC) [11], remote method invocation (RMI) [13], and CORBA [12] are built on top of sockets to provide more abstracted APIs for distributed applications.

Extending the socket model, while providing the same interfaces for the applications, provides a transparent way to enhance the communication software. While a socket is usually attached to a specific network interface card connected to a specific network, our model attaches a socket to multiple network interfaces. In addition, the socket transfer bandwidth is limited by the bandwidth of the card and the network it is bonded to; however, the connections in the proposed model can be configured in different forms for different purposes.

3.1 The Model Architecture

The proposed model is a lightweight software layer between the network resources and the applications (see Fig. 1). The model provides configurable network resource management based on user-defined settings. Although the model can be designed and implemented at different network layers, the middleware layer is selected. Middleware provides system-
and network-independent solutions utilizing existing and optimized socket APIs for advanced networks, thus it is highly portable. Instead of dealing with technical optimizations of different network types, the model utilizes the already optimized reliable transport sockets to deal with multiple network interfaces. The model utilizes channel multiplexing, error control, flow control, and congestion control functions of the reliable transport protocol. It also allows existing applications to use fixed ports and IP addresses while data transfer may actually be performed using other network interfaces or multiple interfaces based on the user-specified configurations. The main function of the model is routing, where the communication addresses and ports are considered as virtual addresses, while the actual communications is done using different addresses.

3.2 The Model Components and Configurations

The model configuration extends the configuration of the routing table. The main function of the routing table is to map the destination address of the sent message to the local NIC, which the message uses to get to the destination. While regular routing is performed at the network layer, the routing process in the proposed model is performed at the middleware level. The configuration of the new model maps the communication addresses and ports of the sent messages to a Multiple-Network-Interface-Card (MNIC) virtual network. MNIC is a virtual network that connects cluster nodes through a single NIC or multiple NICs with multiple networks. For each node connected to a MNIC, some information, including a set of local NIC addresses and the type of transfer to be supported, is defined. There is no shared NIC between multiple MNICs defined on the same machine. Transfers can be sequential, parallel, or a combination based on a threshold value. In the sequential transfer mode, all messages are transferred sequentially through one of the NICs in the source machine to a single NIC at the destination where both cards are defined in the MNIC. The parallel transfer mode allows a message to be fragmented and transferred in parallel through multiple NICs. The combination mode allows messages to be transferred in parallel if their sizes are greater than the threshold value; otherwise, they are transferred sequentially. Each MNIC has a name, which identifies it when multiple machines are connected to it. There are two types of MNICs: (1) Uniform MNIC in which any node has a NIC defined in the MNIC and can communicate with a single NIC on each node connected to the same MNIC. (2) Non-uniform MNIC in which each NIC in any node connected through an MNIC can communicate with all interfaces in other nodes connected with the same MNIC. The first MNIC type must have the same number of NICs in each connected node, while in the second type nodes can be connected to the MNIC through different numbers of NICs. The definition of NICs connected to a MNIC is determined on a node-basis.

The routing table for the model consists of multiple entries each of which consists of message and destination conditions and routing information. The message destination conditions include destination address, destination port ranges, address mask, and a maximum message size value, while routing information includes the MNIC where the message will be transferred through. Host or subnet address can be defined on the destination address, while destination port range and maximum value may or may not be specified. The destination port range contains the range of destination ports and the maximum size value and may contain the maximum size of the message as a condition for considering the routing information for the message.

The user can define different situations. For example, routing can be defined for messages going to a specific host or subnet and to a specific port. Routing can also be defined based on the message size. For example, we can configure the model such that all small messages are transferred through low-latency networks while other messages are transferred through high-bandwidth networks. Another example is that control messages going to specific ports are transferred through a dedicated low latency network. In addition, all large messages are transferred in parallel through a specific MNIC. Figure 2 shows two nodes in a cluster with two defined MNICs. The first MNIC has a single NIC in each node while the second has two NICs per node. The model is responsible for mapping and routing the application logical connections (IP addresses and ports) to the actual physical connects between both ends.

3.3 Multiple Channels

Multiple channels may exist between two nodes through a predefined MNIC in a switched cluster. We will abstract the MNIC network resource between any two nodes by a set of channels. A channel \((k, m)\) exists for a MNIC between nodes \(N_i\) and \(N_j\) in a cluster if there is a physical path between NIC \(k\) in node \(N_i\) and NIC \(m\) in node \(N_j\) in which both interfaces \(k\)
and $m$ are defined in the same MNIC. A MNIC with $n$ independent networks, where each network consists of independent links and switches connecting all nodes, has $n$ channels between any two nodes. For each channel, there are two physical links. The first link is represented by the network interface of the first node and the link from the interface to the switch. The second is represented by the network interface of the second node and the link from the interface to the switch. The channels in a uniform MNIC are not shared, i.e., there is no shared physical link between channels. In addition, the number of channels on a MNIC is equal to the number of NICs on the node defined in the MNIC.

In a non-uniform MNIC, nodes have different number of links connected to the same MNIC, the number of channels between two nodes is equal to or greater than the number of NICs on each node. If nodes $N_i$ and $N_j$ are connected through the same MNIC by the same switch, there are $n_i + n_j$ channels between them, where $N_i$ has $n_i$ NICs and $N_j$ has $n_j$ NICs defined for the MNIC. Network switches are designed to deliver high-aggregate bandwidth between connected interfaces thus they are not considered as shared resources. However, the channels between any two nodes in a non-uniform MNIC contain shared links. The maximum number of unshared channels between nodes $N_i$ and $N_j$ is $\min(n_i, n_j)$. The maximum bandwidth available between $N_i$ and $N_j$ when there is no other load coming from the other nodes in the cluster is $B \cdot \min(n_i, n_j)$ where $B$ is the bandwidth of a single link, assuming that all links are of equal bandwidth. However, when there is load from other nodes, the available bandwidth is shared among the nodes. Standard sockets can be used to open channels, where both local and destination network interface cards can be specified during the binding operation. In addition, to ensure the sent data flows through the specified interfaces, socket option $SO_{DONTROUTE}$ can be used to bypass the normal routing routines of the underlying protocols.

While opening a client socket establishes a connection with a server socket and closing a socket terminates the connection between both ends in the standard socket, in our model multiple channels are established between the client and server sockets. If a client socket opens a connection to a server socket in a machine with one of the interfaces with IP 192.168.0.1 through port 8254, our model will search the model routing table to find all MNICs associated with the destination IP address and port. Information of the associated MNICs and of the available local ports that can be used between the nodes is sent to the destination machine, which replies with information about its MNICs and its free ports for the connections. Multiple channels will be opened in these MNICs using the multiple-channel concepts explained earlier. Although multiple channels are established during socket opening time, which generates some overhead, these channels can be easily reused via socket reuse techniques. Thus, a pool of reusable channels can be maintained where a number of channels are removed or added whenever a socket is opened or closed. For example, consider a scenario in Fig. 2, where the routing table of the model indicates that all small messages should flow through the first MNIC while all other messages should flow in parallel through the second MNIC. If the first machine opens a client socket with a server socket on the second machine, three channels will be established during socket opening since there are three independent networks between both machines.

### 3.4 Small Messages vs. Large Messages

Messages of different sizes are exchanged among processes on a cluster. Some of them are as small as 32 bytes, while others are as large as 1 MB. In the context of this paper, we define a message to be small if the transfer time of the message on a single channel is less than the parallel transfer time of fragmenting the message over multiple channels. This distinction is important since it determines when it is profitable to fragment a message for parallel transfer and when it is not.

### 3.5 Enhancing Large-Message Transfers via Load-Balancing

This section develops an efficient scheme for coordinating and sending large messages over multiple channels. Coordinating multiple sends from multiple source nodes to a single destination node can be achieved by a dynamic allocation scheme as in [4] for uniform cluster networks. However, the scheme in [4] is based on reserving both communication endpoints before sending any large message. The reservation helps avoid resource contention and involves sending different control messages between the NICs of the sender and receiver nodes, which generates some overhead on message transfers as well as the network resources. We propose an efficient scheme to utilize the available resources for large messages without reservation or coordination protocols. Unlike in [4], where the solution requires uniform cluster networks, our solution works on uniform and non-uniform networks.

The maximum utilization (and Bandwidth) for network resources in a MNIC can be achieved if the load balancing among the available network resource is optimal. Optimal load balancing is achieved by having equal load on all network resources, i.e., sender node’s NICs participating in the MNIC and their connections to the cluster switch(es) and receiver node’s NICs participating in the MNIC and their connections to the cluster switch(es). Efficient load balancing among networks can be achieved if we use striping for each large message. In striping, a large message is divided into small fragments of $f$ bytes each then sent through all available channels, while maintaining balanced load on all links. Load balancing can be achieved using, respectively, the sender NICs and the receiver NICs in sequence (e.g. round-robin). Consider a sender with 2 NICs, labeled 0 and 1, a receiver with 3 NICs, labeled 0, 1, and 2, and a 6-fragement message. To achieve load balancing at the sender, we send the fragments in the following NIC sequence: 0, 1, 0, 1, 0, 1. At the same time, we need to re-
ceive them in the following NIC sequence: 0, 1, 2, 0, 1, 2, implying the use of channels pairs (0, 0), (1, 1), (0, 2), (1, 0), (0, 1), and (1, 2). Channel numbers \((k, m)\) for sending fragment number \(s\) between node \(N_i\) and \(N_j\) are determined as follows:

\[
k = s \mod n_i \quad \text{and} \quad m = s \mod n_j
\]

where \(n_i\) and \(n_j\) are numbers of network interface cards connected to a MNIC in nodes \(N_i\) and \(N_j\), respectively.

In some cases, we need to use fragment sizes smaller than \(f\) to achieve optimal load balancing, depending on the message size \(l\). If a message of size \(l\) is to be sent through \(C\) channels while we have \(l/C < f\), where \(f\) is the fragment size already used, then a fragment size \(l/C\) may be used. This technique can be used in both uniform and non-informal cases. Since this method maintains load balancing among all NICs for each node during striping, global load balancing can be achieved in cluster network resources and leads to better network utilization and performance.

If multiple nodes send messages to the same interfaces on the destination nodes, the messages will generate the same load on all destination NICs. Therefore, the bandwidth of the NICs and links will be shared evenly and the load will be balanced. Since each channel represents a reliable transport channel, multiple fragments from different sources can be efficiently multiplexed to the destination NICs. The same is true if two processes executing on the same node send two large messages. Both messages will be distributed equally among the node’s NICs. This technique ensures load balancing without the costly coordination, synchronization or reservation techniques.

Reliable transport, which is utilized for all channels, ensures reliability and FIFO order of the fragments. These properties can be utilized to predefine the order of the fragments. Both sender and receiver sockets after communicating the message size can use the above formulas to know from which channel the fragments will be sent and received. Therefore, there is no need to add any striping header or sequence number to the fragments. The receiver can easily find the sequence using the formulas. If the transport protocol used does not guarantee FIFO order, then sequence numbers must be added to the fragments. Most Sockets provides send and receive commands that deal with direct memories, which can be utilized for striping to avoid extra copying beyond that of the reliable transport protocol.

3.6 Enhancing Latency for Small Messages

Message striping cannot enhance the performance for small messages; however, it can help enhance the throughput. Short messages can be sent in round-robin manner among available channels. Small messages can be sent through sequential MNIC that contains single or multiple channels, or through parallel/sequential MNIC that contains multiple channels. In both cases, round-robin distribution can be used among channels to maintain load balancing on the interfaces. The round-robin distribution among available channels can be used in the same way as the algorithm in Sect. 3.5. Although coordination can be done for allocating the available channels among small messages coming from multiple threads executing on the same machines, this coordination, which involves some synchronization, can be very costly for sending short messages in high performance computing. Therefore, each open socket can distribute the load among its own channels, thus leading to uniform load distribution. Another configuration can be used in which a specific channel is dedicated for small messages. Using this method we could guarantee QoS for small messages such that none is delayed due to other outgoing large messages on the node. As a result the average transfer time of small messages is significantly reduced.

4. The Performance

To evaluate the performance gains of the model, a number of experiments were conducted. All experiments were executed on Sandhills, a 24-node Linux cluster with two AthlonMP 1.4 GHz processors and 1 GB RAM per node. The nodes are connected by a Fast Ethernet and a Gigabit Ethernet. In addition, some nodes are equipped with extra Fast and Gigabit Ethernet interface cards and connections. The experiments measure the round trip time (RTT) and/or the effective bandwidth for different possible configurations of the model, with the effective bandwidth being derived indirectly from RTT as follows:

\[
\text{Effective Bandwidth (Mbps)} = \frac{(8 \times \text{Message size}/10^6)/(\text{RTT}/2)}
\]

A standard TCP socket was used as a point-to-point communication mechanism for the model.

4.1 Cut-Off Points

Experiments were conducted to find the cut-off points among different network configurations and different types of transfers. The cut-off point distinguishes a small message from a large one and therefore is important in determining when a message must be sent in parallel. Finding cut-off points is also useful for determining the performance of sending small messages on different types of networks. The communication performance in terms of RTT for TCP over fast Ethernet (TCP 100), parallel TCP transfer over two and four independent fast Ethernets (TCP 2*100) and (TCP 4*100), TCP transfer over gigabit Ethernet (TCP 1000), and parallel transfer over two gigabit Ethernets (TCP 2*1000) were measured (Fig. 3). The results show high RTT values for TCP 2*100 and TCP 4*100, relative to TCP 100, for messages smaller than 64 Bytes and 256 Bytes respectively. In addition, the results show high RTT values for TCP 2*1000 compared to TCP 1000, for messages smaller than 512 Bytes.

An interesting observation from experiments on this cluster is that the performance of a Fast Ethernet network for
small messages (<128 Bytes) is better than that of a Gigabit Ethernet. This is arguably due to factors such as network drivers, card manufacturing, and switches. This property can be used to enhance the performance of communication by using Fast Ethernet for small messages and gigabit for large ones.

4.2 Separating Small Messages from Large Ones

Experiments were conducted to measure the negative impact on small message latency of transmitting large messages using different types of cluster networks including Fast and Gigabit Ethernet. The experiments involved two machines where two pairs of processes exchange messages. The experiments were designed to measure the impact when both messages are sent and received through the same network interfaces. The first pair of processes was located in different machines exchanges small messages of size 32 Bytes, while the second pair exchanges large messages of predefined sizes. The experiment measures the average RTT of transferring the 32 Byte messages at the same time when messages of other sizes were being exchanged. The results are shown in Fig. 4. While the average RTT for 32 Bytes message on fast Ethernet was 0.123 milliseconds (ms) when there is no load from the second pair, it increased to 1.055 ms with a load of 16 Kbytes from the second pair. In addition, the average RTT increased to 4.398 ms with the load of 1 Mbytes from the second pair. However, small messages do not have high impact on each other. The same performance patterns occurred in the Gigabit Ethernet experiments. The average RTT of 32 bytes increased from 0.171 ms with no load to 0.383 ms and 0.946 ms, with the load of 16 Kbytes and 1 Mbytes, respectively. While some coordination and resource allocation may be done for messages going from and to the same source and destination nodes at the same time, it becomes difficult when there are other messages sent by other nodes in the cluster to the same destination node.

Figure 5 shows the effect of using the model and dedicating a fast Ethernet network for small messages while using other networks for large ones. The results also show that using a dedicated low-cost fast Ethernet network for small messages with the existing networks enhances the performance of applications like DSM that depend on fast exchanges of small messages.

Other experiments were conducted to show the impact of high network traffic on the transfer time of small messages. Two experiments were conducted using 10 and 15 client machines respectively to generate high traffic of small messages of size 32 Bytes. All these machines exchange messages with a single machine with a multithreaded server through a Fast Ethernet network. As a result, there will be high network traffic on the server interface. The average RTT were 0.190 and 0.265 ms respectively. These figures remain small compared to the average RTT of small messages when it competes with a single large message. The average RTT of small messages is 2.490 and 4.398 ms when sent with a large message of sizes 128 Kbyte and 1 MB, respectively. Similar tests were done using the Gigabit Ethernet and the results were also similar to those of the Fast Ethernet, where the average RTT for multiple small messages is much smaller than that of a small message competing with a large message.

4.3 Parallel Transfer

Experiments on parallel transfer for large messages over both multiple fast and gigabit Ethernets were conducted.
Fragments of size at most 1 KB and 8 KB were used for fast and gigabit networks, respectively. The performance in terms of RTT for TCP over fast Ethernet (TCP 100), parallel TCP transfer over two independent fast Ethernets (TCP 2 * 100), parallel TCP transfer over four independent fast Ethernets (TCP 4 * 100), TCP transfer over gigabit Ethernet (TCP 1000), and parallel transfer over two gigabit Ethernets (TCP 2 * 1000) were measured. The corresponding effective bandwidth is shown in Fig. 6. While a peak effective bandwidth of 93.703 Mbps was achieved with the transmission of 1 MB messages on a single fast Ethernet network, peak effective bandwidths of 187.037 Mbps and 370.358 Mbps were achieved on two and four Fast Ethernet networks, respectively. The efficiency is 99.8% for two networks and 98.8% for four. In addition, the effective bandwidths of Gigabit Ethernet are also enhanced, but with less efficiency. This is due to the limitations of the system and high overhead of the underlying reliable protocol TCP. Although the parallel TCP transfer is implemented without adding extra overhead (e.g., extra copying beyond what is performed by TCP or adding extra headers), using TCP for parallel transfer incurs some overhead on the CPU. This is due to high overhead of copying in TCP. However, using optimized and lightweight sockets, which can significantly reduce the overhead, is possible with our model.

4.4 Non-uniform Experiment

Another experiment was conducted to test the load balancing technique on non-uniform connections. In the experiments, one of the nodes (server node) was equipped with two Fast Ethernet cards that are both connected to the same cluster switch. Other nodes (client nodes) have single Fast Ethernet NIC each and all are connected to the same switch. 1 MB Messages were transferred simultaneously between the client nodes and the server node. The average RTT and effective bandwidth for transferring the messages for each client and the aggregated effective bandwidth for different number of clients are shown in Table 2. The average effective bandwidth per client represents the out going bandwidth each client sends to the server. Since the server can only handle a maximum of 200 Mbps, three or more clients will evenly share the load of sending a maximum of 200 Mbps. For example, when two threads were sending the total server bandwidth of 182 Mbps was divided among the two clients such that each sends an average of 91 Mbps. However, when three clients were available each had a smaller load of around 62 Mbps to send to the server, which collectively receives 187 Mbps.

4.5 Applications

Two applications were tested on the prototype implementation. The first is a parallel traveling salesman problem (TSP) algorithm based on the branch-and-bound method [5]. TSP exchanges a large number of small messages that contain updated minimum tour values to reduce the search space. Two experiments were conducted for a 20-city TSP problem. The problem was executed on 8 nodes of Sandhills using JOPI [1], [7], which uses TCP socket. A class was used to handle the search mechanism given the proper range in the problem and a boundary value (minimum current tour value). Another class for the result tour is also used by the different processes to report their results to the master process. The minimum tour value is broadcast among processes to allow them to update their minimum value to speedup their search. Each process will perform the search within its range, report the search result to the master process and request a new range for the next search. An advanced range is sent by the master process to other processes to overlap communication with the computation. In addition, asynchronous communications are used by processes to report their results to the master process. Two types of result messages are sent from the processes to the master: positive result when a minimum tour is found and negative results when none was found within the given range. Table 3 shows the different messages exchanged among processes for the used TSP problem.

The first experiment was conducted using Gigabit Ethernet. The second experiment was conducted using the proposed model that utilizes Gigabit Ethernet for large messages and Fast Ethernet for small messages. The experiments were conducted while there is other generated load on the networks that is maintained at the same level during all experiments. The average execution time of the first experiment was 47.451 seconds while it was 45.436 seconds for the second experiment. This is due to enhancing the communication performance of the messages for the minimum tour found, which enhances the process of reducing
the research space.

The second application is file transfer. As in TSP, two experiments were conducted to test file transfer on a single Fast Ethernet and on two Fast Ethernet networks. A 16 MB file was used. While the put time and get time of the file on a single network was 1.511 and 1.521 seconds respectively, the time was reduced to .797 and .811 seconds respectively using the proposed model. Both experiments show the advantages and performance gains of the proposed model.

5. Conclusion

Clusters can have multiple homogeneous or heterogeneous interconnects among some or all the nodes. However, current software tools and protocols cannot handle multiple networks simultaneously in an efficient way. In this paper, we introduced a flexible model to address this issue. The proposed model is a middleware solution that provides portable, high-level abstractions to the applications that require flexible and enhanced properties from the network. It utilizes reliable transport protocols, while allowing applications to transparently utilize multiple network interfaces and networks. The model provides parallel message transfer to increase bandwidth and throughput for large messages and uses channel partitioning to reduce the average transfer time for small messages. Moreover, it provides load-balancing mechanisms to optimize network utilization and increase achievable bandwidth for large messages.

The model allows applications to use the multiple available networks in different configurations. In addition, this model is system and network independent, thus, it can be interoperated into heterogeneous clusters with varying network interconnects. Furthermore, the flexible enhancements can accommodate changes in processing and communication load on the cluster. As discussed, the model provides flexible and transparent mechanisms to utilize multiple networks on clusters to enhance bandwidth, throughput, and QoS for small messages.

Acknowledgments

This project was partially supported by a National Science Foundation Grant (EPS-0091900) and an Academic Priority Grant of the University of Nebraska-Lincoln. We would also like to thank members of the research computing facility (RCF) at UNL for their continuous support. In addition, we would like to thank the reviewers for their valuable suggestions and expansion recommendations that helped improve the paper’s quality and presentation. A preliminary version of portions of this paper was presented at IEEE IPCCC 2004.

References


Nader Mohamed obtained his B.Sc. in Electrical Engineering from The University of Bahrain, Bahrain, in 1992 and M.Sc. in Computer Science from Western Michigan University, Kalamazoo, Michigan, USA, in 1998. He is currently a Ph.D. Candidate at the Department of Computer Science & Engineering, University of Nebraska-Lincoln in Nebraska, USA. His research interests include middleware, networking, cluster and grid computing, and parallel and distributed software. He has over 20 publications in these areas.
Jameela Al-Jaroodi obtained her B.Sc. in Computer Science from The University of Bahrain, Bahrain, in 1993 and M.Sc. in Computer Science from Western Michigan University, Kalamazoo, Michigan, USA, in 1998. She is currently a Ph.D. Candidate at the Department of Computer Science & Engineering, University of Nebraska-Lincoln in Nebraska, USA. Her research interests include distributed system middleware, heterogeneous systems, parallel and distributed programming models, programming languages, wireless networks, and computer security. She has over 20 publications in major journals and international conferences.

Hong Jiang received a B.Sc. in Computer Engineering from Huazhong University of Science and Technology, Wuhan, China; 1982, an M.A.Sc. in Computer Engineering from the University of Toronto, Toronto, Canada; 1987, and a Ph.D. in Computer Science from the Texas A&M University, College Station, Texas, USA, 1991. Since August 1991 he has been at the University of Nebraska-Lincoln, Lincoln, Nebraska, USA, where he is Associate Professor and Vice Chair in the Department of Computer Science and Engineering. His present research interests are computer architecture, parallel/distributed computing, computer storage systems and parallel I/O, performance evaluation, middleware, networking, and computational engineering. He has over 80 publications in major journals and international Conferences and his research has been supported by NSF, DoD and the State of Nebraska.