To the Graduate Council:

I am submitting herewith a thesis written by Priyanka Dasgupta entitled "Performance Evaluation of Fast Ethernet, ATM and Myrinet under PVM". I have accepted the final copy of this thesis for form and content and recommend that it be accepted in partial fulfillment of the requirements for the degree of Master of Science, with a major in Electrical Engineering.

J. D. Birdwell

Dr. J. D. Birdwell, Major Professor

We have read this thesis and

recommend its acceptance:

Accepted for the Council:

Interim Vice Provost and

Dean of The Graduate School

PERFORMANCE EVALUATION OF FAST ETHERNET, ATM AND MYRINET UNDER PVM

A Thesis

Presented for the

Master of Science

Degree

The University of Tennessee, Knoxville

Priyanka Dasgupta

May 2001

ACKNOWLEDGEMENTS

This work has been supported by the US Department of Justice, Federal Bureau of Investigation under contract J-FBI-98-083. The views and conclusions contained in this document are those of the author and should not be interpreted as necessarily representing the official policies, either expressed or implied, of the U.S. Government.

I would like to thank my advisor, Dr. J.D. Birdwell for his continued guidance and support during the preparation of this thesis. I also want to thank the other members of my thesis committee, Dr. T. W. Wang and Dr. H. Qi for their advice and help.

I acknowledge the support of the Innovative Computing Laboratory at University of Tennessee for allowing me to use the Tennessee Oak Ridge Cluster for the benchmark studies.

ABSTRACT

Congestion in network switches can limit the communication traffic between Parallel Virtual Machine (PVM) nodes in a parallel computation. The research introduces a new benchmark to evaluate the performance of PVM in various networking environments. The benchmark is used to achieve a better understanding of performance limitations in parallel computing that are imposed by the choice of the network.

The networks considered here are Fast Ethernet, Asynchronous Transfer Mode (ATM) OC-3c (155Mb/s) and Myrinet. Together, they represent an interesting range of alternatives for parallel cluster computing. A characterization of network delays and throughput and a comparison of the expected costs of the three environments are developed to provide a basis for an informed decision on the networking methods and topology for a parallel database that is being considered for FBI's National DNA Indexing System (NDIS)[17]. This network is used for communications among the nodes of the parallel machine; thus the security requirements defined for the FBI's Criminal Justice Information Services Division Wide Area Network (CJIS-WAN) [12] are not a concern.

TABLE OF CONTENTS

CHAPTER		PAGE
1.	**INTRODUCTION**	1
	PROBLEM STATEMENT	1
	MOTIVATION	1
	METHOD	3
	SUMMARY OF RESULTS	3
2.	**BACKGROUND**	5
	LITERATURE SURVEY	5
	SUPERCOMPUTERS	10
	CLUSTER COMPUTING	14
	BEOWULF CLUSTER	17
	PARALLEL COMPUTING	19
	MPI	25
	MPICH	26
	PVM	27
	COMPARISON OF PVM AND MPI	28
	CHOICE OF PVM OVER MPI	29

CHAPTER **PAGE**

PVM CONCEPTS....30

PVM COMPONENTS....31

MESSAGE PASSING IN PVM....32

COMMUNICATION PROTOCOLS IN PVM....33

BOTTLENECKS IN PVM APPLICATIONS....36

CONGESTION CONTROL IN TCP....38

NETWORKING TECHNOLOGIES....40

FAST ETHERNET....42

ATM....44

ATM ARCHITECTURE....46

LAN EMULATION (LANE)....47

MYRINET....48

3. EXPERIMENTAL DESIGN....52

ARCHITECTURE....52

TEST MACHINES....56

PERFORMANCE TESTS....58

XPVM....62

CHAPTER **PAGE**

4. RESULTS AND DISCUSSIONS.....67

TIMING SCENARIO I: EFFECT OF ENCODING.....70

TIMING SCENARIO II: EFFECT OF DIRECT ROUTING.....75

TIMING SCENARIO III: LATENCY STUDIES.....80

COMPARISON OF LATENCY ON FAST ETHERNET AND MYRINET ON TORC.....80

LATENCY MEASUREMENT ON ATM CLUSTER AT LIT.....83

COMPARISON OF LATENCY OF FAST ETHERNET ON TORC AND LIT CLUSTER.....83

TIMING SCENARIO IV: EFFECT OF MESSAGE SIZE ON TRANSFER RATES.....86

COMPARISON OF EFFECT OF MESSAGE SIZE ON TRANSFER RATES FOR FAST ETHERNET AND MYRINET ON TORC CLUSTER.....88

EFFECT OF MESSAGE SIZE ON TRANSFER RATES FOR ATM ON LIT CLUSTER.....92

COMPARISON OF EFFECT OF MESSAGE SIZE ON TRANSFER RATES FOR FAST ETHERNET ON TORC AND LIT CLUSTER.....96

CHAPTER **PAGE**

TIMING SCENARIO V: EFFECT OF NUMBER OF NODES ON TRANSFER RATES....96

COST ANALYSIS....98

SUMMARY OF RESULTS....106

5. RECOMMENDATIONS FOR FUTURE RESEARCH....108

BIBLIOGRAPHY....110

APPENDIX....115

VITA....124

LIST OF TABLES

TABLE **PAGE**

3.1 Specifications of the nodes in TORC cluster.....57

3.2 Specifications of the nodes in LIT cluster.....59

4.1 Comparison of delay times with encoding and using raw format.....71

4.2 Comparison of delay times with direct routing and normal routing.....76

4.3 Comparison of latency of Fast Ethernet and Myrinet on the TORC cluster.....81

4.4 ATM Latency with varying number of nodes.....84

4.5 Comparison of latency of Fast Ethernet on LIT and TORC clusters.....87

4.6 Summary of the Different Testing Scenarios.....107

LIST OF FIGURES

FIGURE **PAGE**

1.1 Flow chart showing different testing scenarios....4

2.1 Message Size Vs Bandwidth using MPICH 1.0.12 [6]....7

2.2 Message Size Vs Bandwidth using PVM 3.3.11 [7]....8

2.3 Performance of the computer systems for the last 5 decades compared to Moore's Law [23]....13

2.4 Shared-Device Model....16

2.5 Shared-Nothing Model....17

2.6 Comparison of Performances of CPU and DRAM....20

2.7 Perfect Parallelism in CODIS search engine....22

2.8 Functional and Data Parallelism....24

2.9 Communication Protocols in PVM....33

2.10 Communication between tasks in Normal mode....35

2.11 Communication between tasks in Direct mode....35

2.12 Congestion control in TCP....41

2.13 ATM cell....45

2.14 The ATM Network Architecture....46

2.15 LANE Components....49

FIGURE **PAGE**

3.1 Recommended implementation of CODIS parallel search server....53

3.2 Redundant nodes connected through redundant network switches....54

3.3 Architecture of each cluster....55

3.4 Overview of the benchmarking software....61

3.5 Network View....64

3.6 Space-Time View....65

3.7 Utilization View....66

4.1 Excel Pivot Table....68

4.2 Example Bubble Chart....69

4.3 Comparison of delays with encoding and using raw format....72

4.4 Comparison of transfer rates with encoding and using raw format....73

4.5 Improvement in performance using raw format over encoded format....74

4.6 Comparison of delays with direct routing and normal routing....77

4.7 Comparison of transfer rates with direct routing and

FIGURE **PAGE**

normal routing.....78

4.8 Improvement in performance with direct routing over normal routing.....79

4.9 Comparison of latency for Fast Ethernet and Myrinet on TORC cluster.....82

4.10 Improvement in performance of Myrinet over Fast Ethernet on TORC cluster.....84

4.11 ATM Latency.....85

4.12 Comparison of latency of Fast Ethernet on LIT and TORC cluster.....88

4.13 Percentage increase in latency for Fast Ethernet on LIT cluster over TORC cluster.....89

4.14 Transfer rates for Fast Ethernet and Myrinet with 2 nodes on the TORC cluster.....90

4.15 Transfer rates for Fast Ethernet and Myrinet with 4 nodes on the TORC cluster.....90

4.16 Transfer rates for Fast Ethernet and Myrinet with 6 nodes on the TORC cluster.....91

4.17 Transfer rates for Fast Ethernet and Myrinet with 8 nodes on the TORC cluster.....91

FIGURE **PAGE**

4.18 Plot of ratio of transfer rates for Myrinet to Fast Ethernet versus bytes transferred....93

4.19 Transfer rate for ATM with 2 nodes on the LIT cluster....94

4.20 Transfer rate for ATM with 4 nodes on the LIT cluster....95

4.21 Comparison of transfer rates for Fast Ethernet on TORC and LIT cluster with 2 nodes....97

4.22 Comparison of transfer rates for Fast Ethernet on TORC and LIT cluster with 4 nodes....97

4.23 Comparison of transfer rates for Fast Ethernet on TORC and LIT cluster with 6 nodes....98

4.24 Effect on transfer rate with increasing number of nodes (Bytes transferred 40 Mbytes)....99

4.25 Control Rack....100

4.26 PC Rack....102

4.27 Fully Configured System....103

A.1 Bubble chart for message size of 40 bytes and on 8 nodes of the TORC cluster....117

A.2 Bubble chart for message size of 400 bytes and on 8 nodes of the TORC cluster....118

A.3 Bubble chart for message size of 4 Kbytes and on 8 nodes

of the TORC cluster.....119

A.4 Bubble chart for message size of 40 Kbytes and on 8 nodes

of the TORC cluster.....120

A.5 Bubble chart for message size of 400 Kbytes and on 8 nodes

of the TORC cluster.....121

A.6 Bubble chart for message size of 4 Mbytes and on 8 nodes

of the TORC cluster.....122

A.7 Bubble chart for message size of 40 Mbytes and on 8 nodes

of the TORC cluster.....123

CHAPTER 1

INTRODUCTION

PROBLEM STATEMENT

The purpose of this research is to develop a computer network benchmark program for parallel computation. The benchmark is used to evaluate the effect of various network interconnection strategies on Parallel Virtual Machine (PVM) performance. Three network interconnects were evaluated: Fast Ethernet, Asynchronous Transfer Mode (ATM OC-3c) and Myrinet.

MOTIVATION

The implementation of the new search engine for the FBI CODIS DNA database is migrating from the original single process recursive implementation to a multithreaded implementation and finally to a parallel machine implementation [30]. The parallel implementation will execute Search Engine and Load Balancing objects on each host, probably with as many Search Engine objects as there are processors on a host [31]. A root host will be used to accept Client requests and create a Search Client object to handle each request. A critical component of the parallel implementation is the method used to balance the workload across the set of available hosts and processors. During the transfer of workload between nodes, the traffic routed through a switch

may exceed the switch's bandwidth, resulting in congestion and degraded performance.

Transport Control Protocol (TCP) uses a sliding-window flow control algorithm with "Go-Back-N" packet retransmission upon timeout [14]. The flow control mechanism of TCP avoids packet overflow at the receiver side; however, it cannot prevent buffer overflow in a Local Area Network (LAN) switch due to network congestion. When this occurs, the switch discards some packets. To control congestion, TCP slows down the transmission of packets until congestion is alleviated. Eventually the retransmission mechanisms of TCP on the sending hosts are triggered and start resending more packets than needed, increasing network traffic and therefore making the LAN even more congested.

This is very undesirable in our application, and so the timing and congestion studies are very important as guides to choose both the type of network switch used and the number of nodes in a cluster. The networks we discuss in this thesis are Switched Fast Ethernet, ATM and Myrinet. Fast Ethernet gives higher bandwidth at low cost. ATM was originally designed for applications in the telecommunications industry but it has also been extensively used for real-time, multimedia and parallel applications. Myrinet was designed especially for parallel computing applications.

METHOD

In order to evaluate the performance of various networking topologies, the benchmark tests were done on each of the three network interconnects. Chapter 2 provides background on the component technologies and a review of previous benchmark studies. Our benchmarking software attempts to stress the network switch to gain information on limitations imposed by the switch as well as the bandwidth of each connection. The software used to perform the timing studies and the test conditions are explained in Chapter 3. The testing scenarios consist of 2 broad categories – testing options available in PVM and testing the different networking topologies.

In the first category, we have observed the effect of encoding and the improvement in transfer rate due to direct routing. The second category involved testing of each of the network technologies for latency and transfer rates with varying number of nodes and varying message sizes. This can be illustrated as in the flow chart in Figure 1.1.

SUMMARY OF RESULTS

The results are discussed in detail in Chapter 4. From the timing studies it is observed that Myrinet shows a 30-50% increase in performance over Fast Ethernet. But the details discussed in Chapter 4 about the cost benefits make Fast Ethernet the best option for our application.

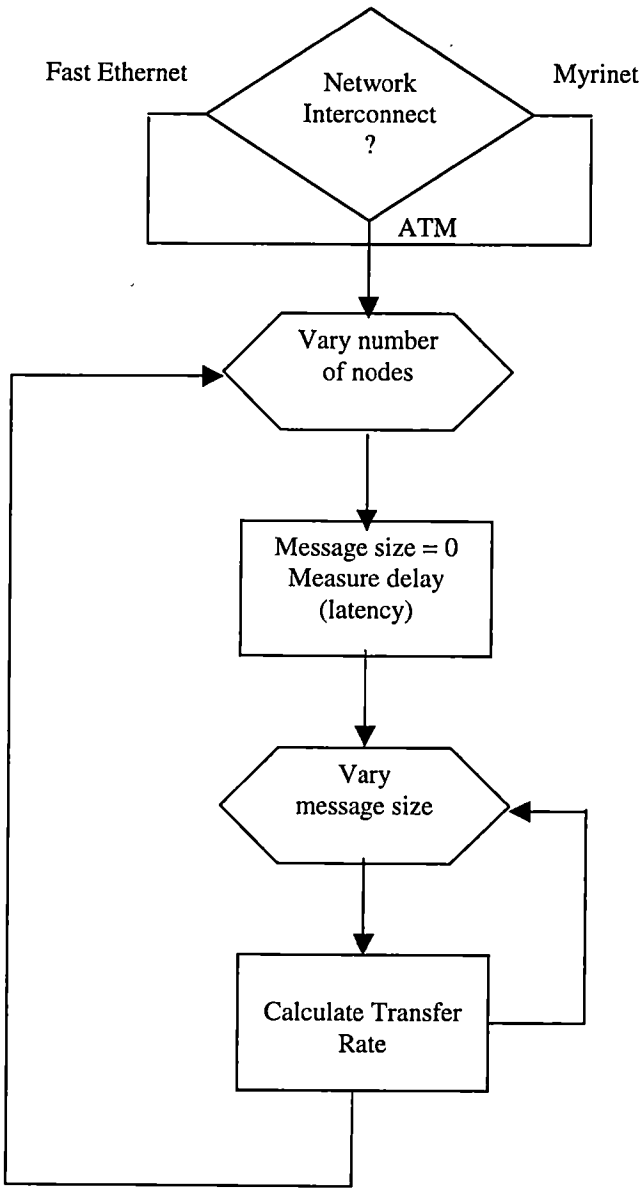


Figure 1.1: Flow chart showing different testing scenarios.

CHAPTER 2

BACKGROUND

LITERATURE SURVEY

Recent years have witnessed a growing trend towards parallel computing. Performance evaluation plays an important role during the architecture design, development and implementation stages of designing a parallel cluster for a particular application. There have been many benchmark studies performed on different network connections with different programming libraries.

In October 1993, Michael J. Lewis and Raynond E. Cline [1] of Sandia National Laboratory described the results of performance tests of Parallel Virtual Machine (PVM) in a switched Fiber Distributed Data Interface (FDDI) [2] heterogeneous distributed computing environment. The test environment, called the Heterogeneous Environment and Test bed (HEAT), was a group of 50 workstations consisting of 10 of each of the following: IBM/RS6000's, HP-9000 PA-RISC's, SGI IRIS Indigos, DEC Alphas, and Sun SPARCstation 10's. These workstations were networked via a 100 Mbps DEC FDDI Gigaswitch and several DEC Concentrator 500s. Simple ping-pong tests were carried out, and the results showed that pvmd, the PVM daemon, could become a bottleneck in communication intensive applications, especially those in which multiple tasks reside on the same machine. Also from the results, it was

concluded that PVM's efficiency needed to be improved in order to better exploit the speed of the network hardware. These results are important for our application, as we would need to take into account the bottlenecks due to PVM. These tests reported in [2] were done only on FDDI and so do not provide a direct comparison of performance with other interconnects.

In June 1996, latency and bandwidth-timing benchmarks were executed on various networks of the Lewis Advanced Cluster Environment (LACE) at the NASA Lewis Research Center [3]. LACE was a group of thirty-two networked IBM RS/6000 machines (lace01 - lace32) plus one 'control' node called lace. The tests bounced messages of various lengths back and forth between lace15 and lace16. Messages were bounced back and forth 100 times using Asynchronous Transfer Mode (ATM), Fiber Distributed Data Interface (FDDI) and Fiber Channel Standard (FCS) [4] networks, and the IBM ALLNODE switch [5], all in dedicated mode. The Message Passing Interface (MPI) and PVM libraries were used. The results showed small differences between the libraries on different networks for small message sizes. For large messages, PVM performance was steady and predictable, with ATM OC-3C performing best, then FCS, FDDI, and ALLNODE. All three MPI implementations tested performed best with FCS on large messages. ATM performance peaked on messages of around 200KB. The comparison of the different networks using MPI is shown in Figure 2.1 [6] and using PVM in Figure 2.2 [7]. These results were obtained using a ping-pong test between 2 nodes only. For our application we need to support

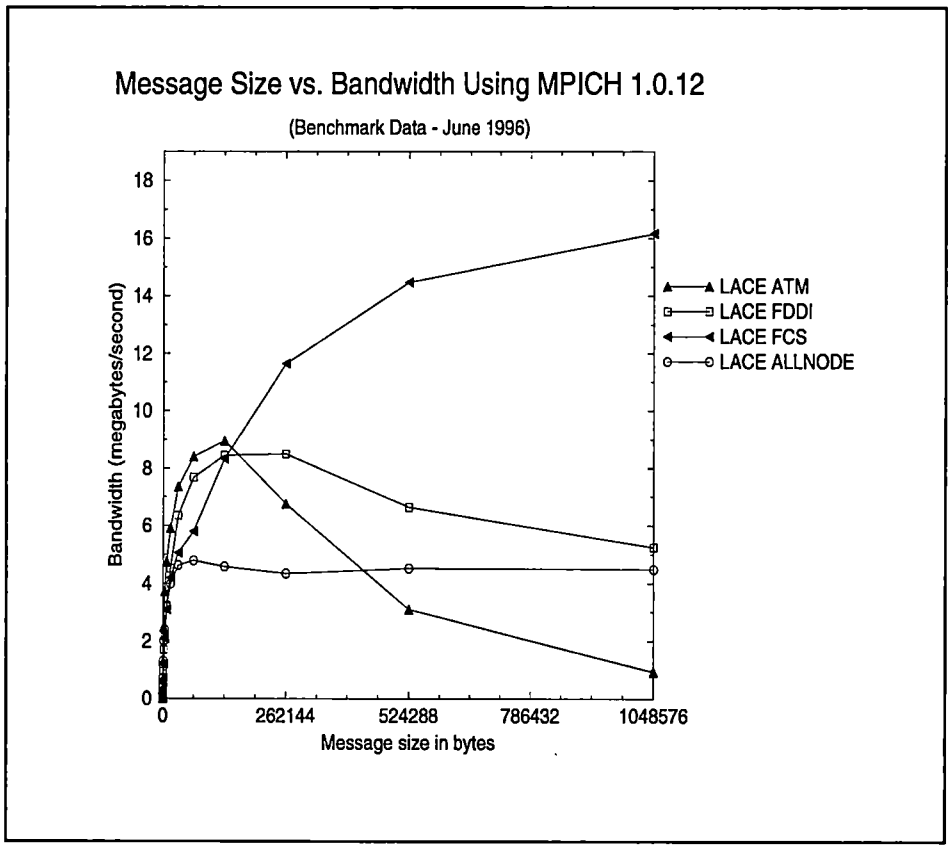


Figure 2.1: Message Size Vs Bandwidth using MPICH 1.0.12 [6].

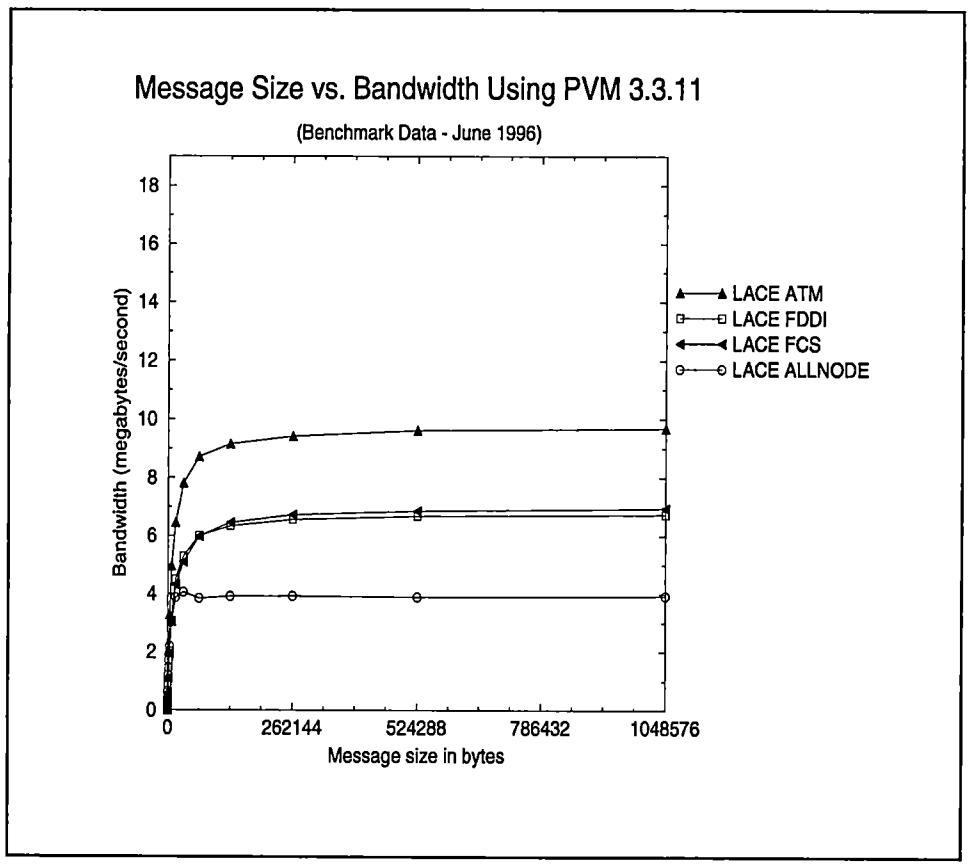


Figure 2.2: Message Size Vs Bandwidth using PVM 3.3.11 [7].

simultaneous communication among all nodes, so these results cannot be used as a basis for selection of our network topology.

Different physical networks and protocols have different performance characteristics. The Performance Based Path Selection (PBPS) strategy takes advantage of this heterogeneity, selecting a path among the available options to minimize the communication time. JunSeong Kim and David Lilja [8] of the University of Minnesota studied latency characteristics as a function of the message size for Ethernet, Fiber Channel and High Performance Parallel Interface (HiPPI) [9] networks in order to evaluate PBPS. The results showed clear tradeoffs between Ethernet and Fiber Channel: Ethernet produced higher transfer rates than Fiber Channel when sending small messages (a few hundreds of bytes), while Fiber Channel produced higher transfer rates for larger messages. These results are used as the basis to select the appropriate path; that is, small messages was sent on Ethernet and others on Fiber Channel. Inter-process communication libraries like PVM and MPI do not support multiple communication paths, so a separate library was developed. In our application, there is only one path type between any two hosts, so PBPS is not useful.

In April 1996, Michael Steed [10] developed the APACHE (Automated Pvm Application CHaracterization Environment) performance prediction system for PVM programs. This system represents performance with a set of equations. These equations were verified to predict very close performance to those observed by carrying out various tests such as a Jacobi algorithm, matrix multiplication, numerical

integration and a parallel sort. These equations could also be used to predict the performance for different networks by changing the factors accordingly. Steed compared the performance of Ethernet and ATM. The results showed the latency of PVM communications was slightly higher over ATM than over Ethernet. Thus, the performance advantage of ATM was more apparent for communications involving large messages and a large number of processes, when the performance of Ethernet began to suffer because of bandwidth restrictions and packet collisions. This performance prediction system does not verify the performance of the system using a ping-pong test, so the results have limited utility for our application.

We also found many studies, [36] which either evaluate performance for only one type of network comparison between network types, or examine only isolated performance factors such as the effect of the number of processors or the effect of message size. These tests are pair-wise and do not simulate the stress that will be developed in the network during a parallel data search of the magnitude we are considering. Hence, we needed to develop a new benchmarking method to make a well-rounded decision for our application.

SUPERCOMPUTERS

A supercomputer is a computer that performs at or near the currently highest operational rate for computers [32]. A supercomputer is typically used for scientific

and engineering applications that must handle very large databases or do a great amount of computation or both.

Supercomputer development began in the 1960s. The first big commercial supercomputer was the Control Data Corporation (CDC) 6600 in 1964. IBM developed the 360/90 in 1967 and many others in the 360 and 370 series during this period. By 1975, Seymour Cray had left CDC and formed Cray Computer, developing a separate line of supercomputers that dominated the supercomputer field for the next two decades. Starting with the Cray 1 in 1976, with speeds of 100 million floating-point operations per second (Mflops), reliability and performance grew to about 1.9 Gflops with Cray 2. As the Cray 2 arrived in 1985, other companies also came into the scene. CDC introduced the Cyber-205, while Hitachi, Fujitsu, and NEC each offered their own design.

Initially supercomputers were limited to research areas and for applications in national defense, weather modeling, aircraft and space research. With the growing popularity of supercomputers in academic fields and industries, research began focusing on building cheaper supercomputers, and in the 1990's, one of those ways turned out to be using a big array of small computers or computer chips.

According to the Top 500 Supercomputer List [19], the most powerful supercomputer is the Department of Energy's Accelerated Strategic Computing Initiative (ASCI) White [20]. It has been developed by IBM and Lawrence Livermore National

Laboratory. It has 8192 processors and a peak performance of 12.3 teraflops (Tflops). IBM's most ambitious supercomputer, the Blue Gene [21] is planned to have 1 million processors in an array that will yield the first petaflop computer.

Despite all these changes, the evolution of performance on a large scale is a very steady and continuous process. The trend seems to obey Moore's law. If we plot the peak performance of various computers of the last 5 decades in Figure 2.3 [23], which could have been called the 'supercomputers' of their time, we see how well this law holds for almost the complete lifespan of modern computing. On average, we see an increased performance of two orders of magnitude every decade.

These days there is another class of system that has supercomputer capability and has become popular – cluster computing. One example of this class is the Beowulf cluster [22] - multiple Linux computers tied together with fast communications links. There are three primary limits to performance at the supercomputer level: individual processor speed, the overhead involved in making large numbers of processors work together on a single task, and the input/output speed between processors and between processors and memory [18]. In the next section we discuss cluster computing.

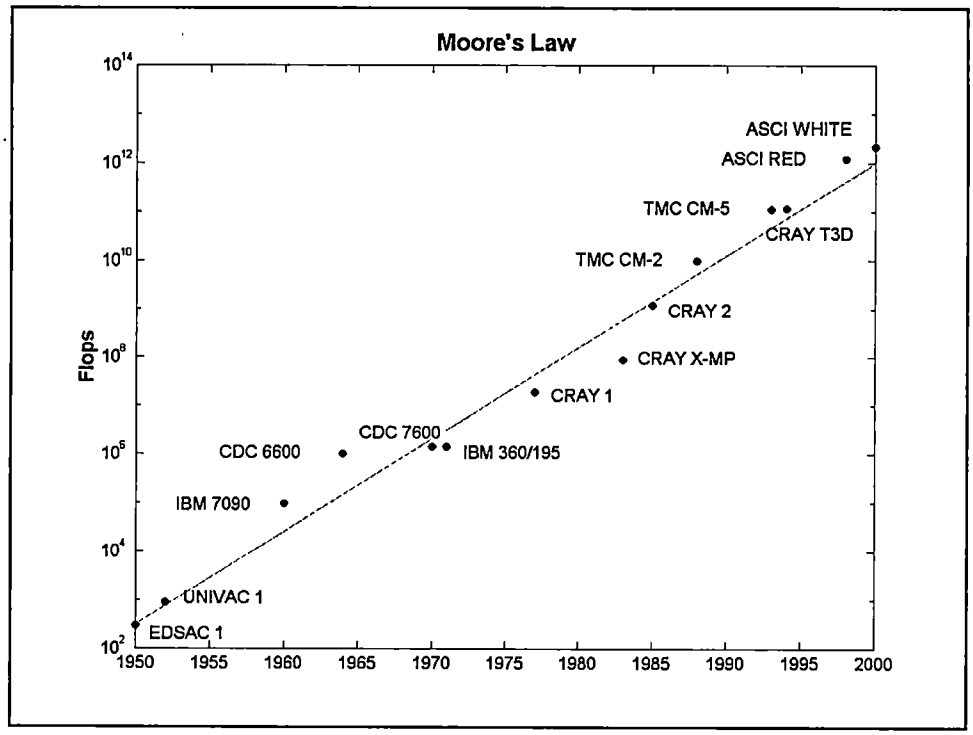


Figure 2.3: Performance of the computer systems for the last 5 decades compared to Moore's Law [23].

CLUSTER COMPUTING

Cluster computing interconnects a number of computers and uses their total computational power to solve a single problem. Clusters can be either a network of workstations (NOW) or a collection of specialized processor nodes. There is no restriction to the architecture of the nodes used in a cluster. It can range from Shared Memory Processors (SMPs) to Vector Processors or to a cluster of independent machines. Each of these architectures needs a different programming technique in order to achieve the best performance. Clever programming could help in exploiting this heterogeneity and improving the performance further, but it would make the programming more complex.

Cluster computing is gaining popularity because of the low price to performance ratio associated with it. With the falling prices of hardware such as the network interface cards and switches needed for building a cluster, clusters are becoming cheaper. A well-designed cluster can be scaled to very large systems without much trouble. This is useful since the clusters can be expanded if the complexity of the problem increases. The cluster can also be upgraded over the years by changing individual nodes or network components to take advantage of the latest developments in technology.

In a cluster, a faulty node can be replaced without too many hassles. This may reduce the performance of the cluster but the computation may still continue. This aspect of cluster computing is very important in applications such as ours where continuous operation is desirable.

When using a cluster, a programmer has to be careful of the data formats being used by the individual nodes. Sometimes data formats are not compatible and the data sent cannot be interpreted at the receiving end. This is taken care of by adopting a standard encoding technique during communication.

Another important issue associated with efficient cluster computing is load balancing. In a cluster some nodes may be faster than others and will finish their work faster and then remain idle. Random effects can also cause imbalance and allow some processors to become idle. This lowers the overall efficiency of the cluster. In order to make use of the wasted time cycles, work from the busy nodes needs to be transferred to the idle nodes. The implementation of load balancing needs to be based on the nature of application and the communication and computation times. In the final implementation of the search engine for the FBI CODIS DNA database, load balancing will be an important feature to improve the search timing. The results of this thesis assist in the final implementation of load balancing.

Sometimes a common problem in clusters is the network load imposed by other network users. This will not be a problem for us because in our final design we are using a network dedicated to the parallel machine.

There are two common implementation models for a cluster – the shared-device model and the shared-nothing model. In the shared-device model, software running on any computer in the cluster can gain access to any hardware resource connected to any

computer in the cluster. For example in Figure 2.4, both nodes A and B can access data from the SCSI drive. In such models, synchronization is an important issue.

In the shared-nothing model, every node of the cluster has access to a subset of the hardware resources available. For example as shown in Figure 2.5, node B cannot read or write to the SCSI drive and so if it needs to retrieve data from the drive, it is routed through node A that has the permission to do so. Shared-nothing models provide more scalability. Also since a certain piece of hardware can be accessed by only one node at a time, synchronization problems are reduced.

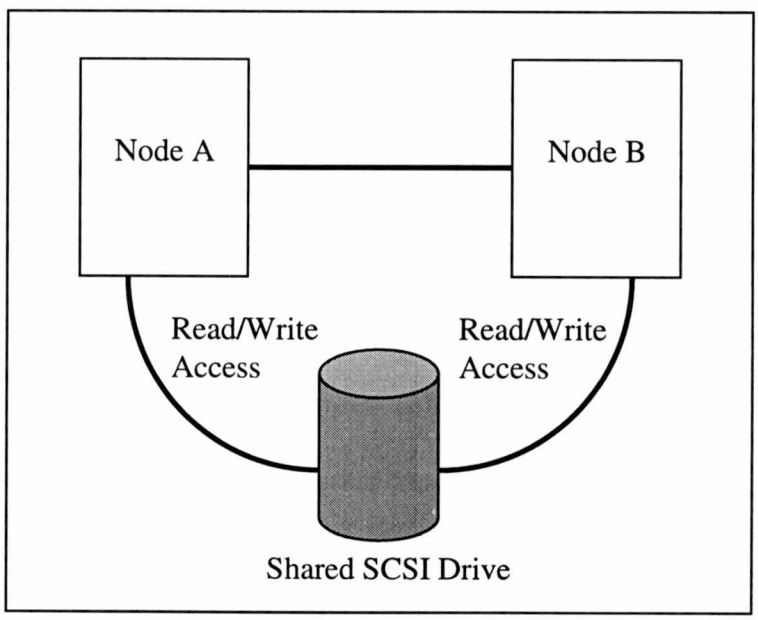


Figure 2.4: Shared-Device Model.

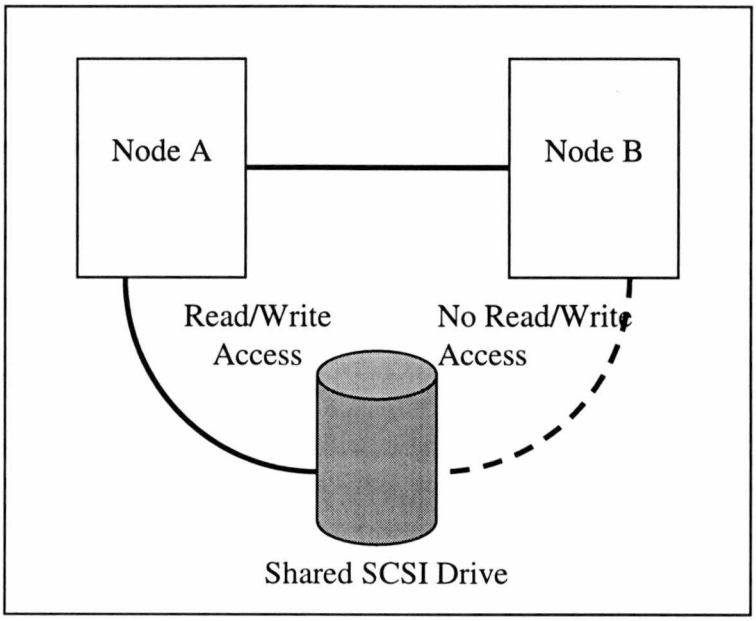


Figure 2.5: Shared-Nothing Model.

BEOWULF CLUSTER

Beowulf is a method of supercomputer construction using a cluster of commodity off-the-shelf (COTS) computers, interconnected with a technology such as Ethernet, and running programs written for parallel processing. The original Beowulf cluster was developed in 1994 at the Center of Excellence in Space Data and Information Sciences (CESDIS) [22], under a contract from the US National Aeronautics and Space Administration (NASA) at the Goddard Space Flight Center in Greenbelt, Maryland. Thomas Sterling and Don Becker built a cluster computer that consisted of 16 Intel DX4 processors connected by 10 Mbps Ethernet. Their success led to the Beowulf Project, which resulted in the development of similar COTS clusters. A number of clusters have been developed in universities and research groups, ranging from the

original 16-processor Beowulf to Avalon, a cluster of 140 Alpha processors built by the Los Alamos National Laboratory.

To build a Beowulf cluster we need a certain number of compute nodes. The number of nodes could depend on any of the following: size of the application, throughput desired, budget of the project, and availability of nodes. An additional node is used in the cluster for system management and cluster configuration. The nodes in the cluster are connected using a system interconnect. There are various options available from the traditional Ethernet to switched networks like ATM and Myrinet. The most commonly used operating system in Beowulf clusters is Linux. This is attributed to its free availability, high reliability and good efficiency. In addition, a Beowulf cluster also needs a message passing protocol to communicate and coordinate between the nodes of the cluster. The most commonly used protocols are implemented by the Parallel Virtual Machine (PVM) and Message Passing Interface (MPI) libraries.

We have proposed a Beowulf topology for our implementation of parallel search. Since our application requires a high bandwidth, ATM, Myrinet, Fast Ethernet or Gigabit Ethernet, all switched networking topologies, will be used in our final implementation. The task of this research is to provide a basis for an informed decision among these choices.

PARALLEL COMPUTING

Parallel processing is the use of multiple processors to execute different parts of the same program simultaneously. The main objectives of parallel computing are to reduce the wall clock time for an application and to increase the throughput of the system.

Programmers are moving from sequential programming to parallel programming. According to Moore's law, CPU speed doubles every 18 months. CPU technology has been maintaining this trend, but eventually there will be physical limitations to this growth [43]. Using multiple processors is a way to improve the speed and capability of large systems. In addition, the speeds of RAM and hard disks have not been keeping up with CPU speed. This difference has been increasing over the years and introduces a serious bottleneck in the performance that is illustrated in Figure 2.6. Parallel computing can reduce this effect.

In order to coordinate between the processors working on a single problem, some amount of communication is required. The ratio between computation and communication is known as granularity. If the granularity is too fine, that is, if the ratio of computation to communication is too low, the speedup is limited. There is a limit to the speedup that can be obtained by parallelizing a code. This is given by Amdhal's law, which states that the speedup is limited by the fraction of the code that must be executed sequentially.

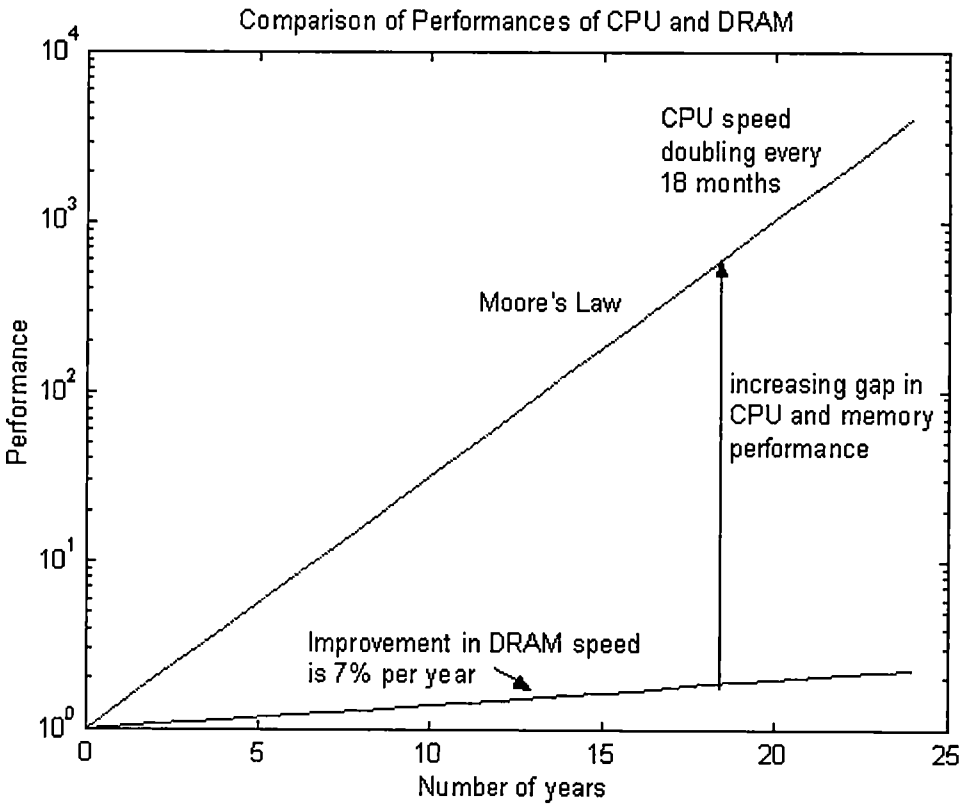


Figure 2.6: Comparison of Performances of CPU and DRAM.

Not every application is parallelizable. Sometimes, the speedup obtained by parallelizing a code is not substantial compared to the efforts put in to develop it or the money spent to implement it. Applications that can be parallelized fall under one of these categories:

- Perfect Parallelism: In such an application, the work can be divided among the processors, and computation can be done with minimal communication. These applications provide very high speedup.
- Data Parallelism: In such applications, the data are divided among different processors, and the same operations are performed on different data in each processor.
- Function Parallelism: The entire application is divided into separate functions that are distributed among the processors.

The CODIS search engine can benefit from all three types of parallelism. An easy way to obtain almost linear speed-up of the search engine is to run identical search engines with different databases on n processors. The stored data records are divided into n roughly equal subsets that are assigned to the processors. A root processor node distributes workload and collects results. This is shown in Figure 2.7.

Although the perfect parallelism of Figure 2.7 distributes workload effectively, increasing demands for both storage of new data records and search requests will eventually overwhelm the capacities of the host processors. Data parallelism and function parallelism can be used to overcome this. Each host is replaced by a cluster of

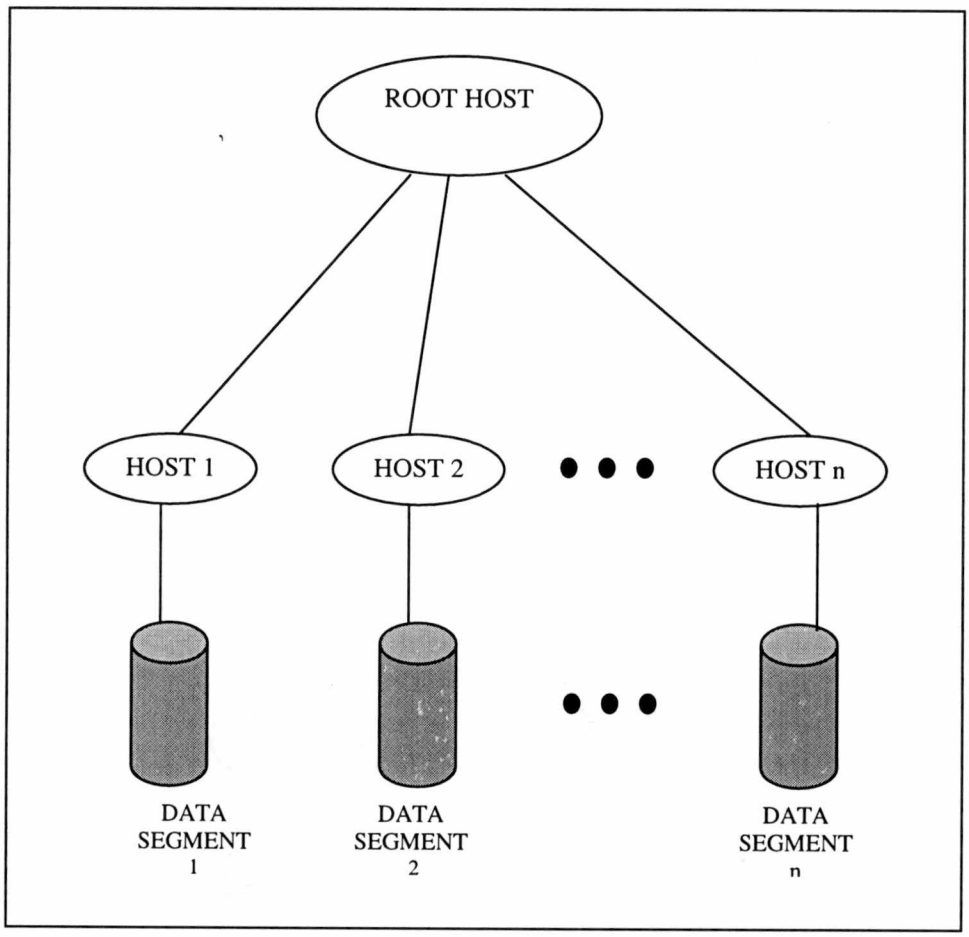


Figure 2.7: Perfect Parallelism in CODIS search engine.

hosts, each having identical data (data parallelism) as shown in Figure 2.8. Search and data storage requests are divided into elementary processing requests that can be distributed across the processors assigned to the cluster. These elementary requests can be performed by any processor because of data parallelism. A load balancing method transfers workload among the processors, ensuring a highly efficient utilization of the cluster's resources.

There are two basic approaches to parallel programming – Data Parallel Programming and Message Passing. Data parallel languages deal with single operations over large collections of data. In data parallel languages, the programmer does not explicitly handle communications and message passing. High Performance Fortran [16] and Fortran 90 [15] are examples of data parallel languages. Data parallel programming does not support functional parallelism, and does not appear to be relevant to the requirements of the CODIS search engine.

Message passing is a programming paradigm in which the programmer explicitly handles data distribution and communication. Non-local data are passed between the processors using messages. Message passing supports all three types of parallelism – perfect, data and functional parallelism. Parallel Virtual Machine (PVM) and Message Passing Interface (MPI) are the most frequently used message passing libraries.

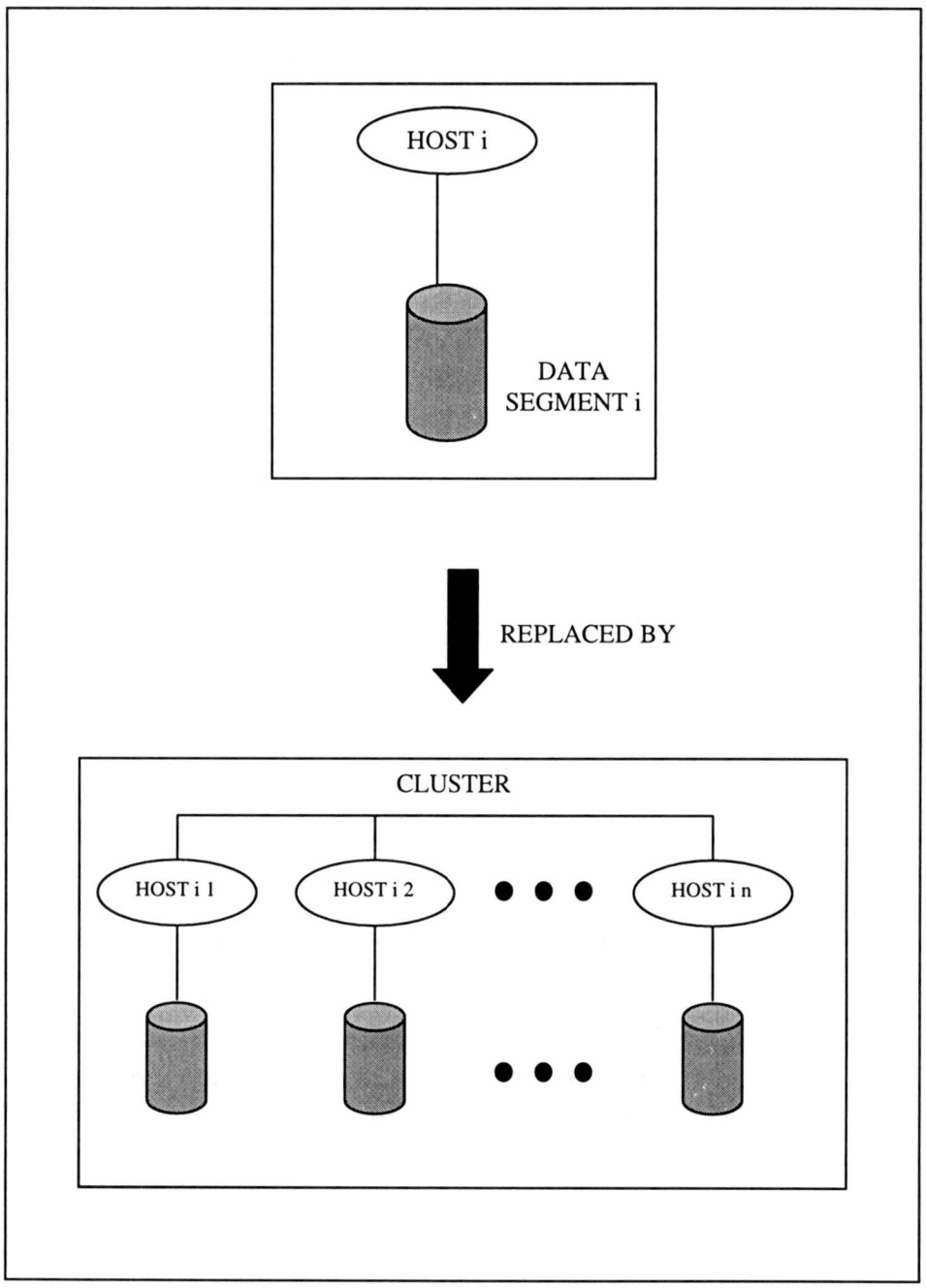


Figure 2.8: Functional and Data Parallelism.

(Every host of Figure 2.7 is replaced by a cluster)

MPI

MPI is the first standardized message-passing library, a collection of routines for facilitating communication among the processors in a distributed memory parallel program. It has been defined by the MPI Forum, which consists of a broad group of parallel computer users, vendors, and software writers [27].

In MPI a group and a rank relative to the group specify every process. Two MPI processes can only communicate if they share a communicator. A process can belong to more than one communicator. MPI_COMM_WORLD is a special communicator, which includes all the tasks in an MPI application.

MPI provides a wide variety of message passing modes for point-to-point communication. There are four options available to support blocking and non-blocking sending of data – synchronous, buffered, ready and standard. The receive call does not specify any mode. It only has options for a blocking or a non-blocking receive.

MPI also provides collective communication functions, which allow distribution and collection of data across a group of processors. Functions are available to perform global reductions, broadcasts, scatter and gather operations. All processes in a communicator call collective functions. Each call blocks all routines until it is locally completed.

MPI supports derived datatypes. This provides a method of communicating non-contiguous or mixed datatypes in a message. These datatypes are derived from the basic MPI datatypes. MPI also supports Cartesian grid topologies which makes it easier to program algorithms that are based on regular grid structures. It does not provide functions for multithreading, but it is designed to be thread safe.

MPICH

MPICH is a freely available, portable implementation of MPI. MPICH is a joint-effort project between Argonne National Laboratory and Mississippi State University. The main design goal of MPICH is to combine portability with high performance. MPICH can be implemented on a distributed memory parallel supercomputer, shared memory architectures and even a network of workstations.

MPICH divides its implementation into a top level, which translates the complicated MPI API into calls to a small number of low-level routines called the Abstract Device Interface (ADI). MPICH contains many implementations of the ADI. This makes it portable. At the lowest level, one implementation of the ADI is the channel interface. It consists of five functions and performs the basic task of transferring data from address space of one process to another.

PVM

PVM (Parallel Virtual Machine) is the result of a research project that began in 1989 at Oak Ridge National Laboratory to harness a group of heterogeneous computers to be used as a single 'virtual' machine. PVM predates MPI but has additional flexibility that MPI does not offer. Since its inception, PVM has undergone a number of revisions with new functions being added.

Under PVM, tasks are initiated under the control of a daemon that spawns the program on other hosts. Every PVM task is associated with a unique task identifier (TID). PVM tasks can be grouped. In each group, tasks are identified by a group instance number along with their TID.

PVM provides functions for point-to-point communications and for synchronization. It provides asynchronous blocking send, asynchronous blocking receive, and non-blocking receive functions. Collective functions such as broadcast, multicast and global reductions are available by linking to the group library (libgpvm3.a).

PVM provides dynamic resource management. It can add or delete hosts any time during the application's runtime. It also provides a number of fault tolerant features by which a host can notify other hosts of its status. These features make PVM attractive for the CODIS search engine.

PVM can run over a wide variety of networks, architectures and applications. In order to do so PVM provides various encoding techniques to overcome the difficulties due to heterogeneity in data formats on the machines.

PVM provides bindings for Fortran77, C and C++. Programs written under PVM in any of these languages are interoperable.

COMPARISON OF PVM AND MPI

Depending on the nature and the requirements of the application, a choice between PVM and MPI must be made. This comparison of PVM and MPI is needed in order to make an appropriate choice between the two message paradigms for our application. In [11], Geist presented an excellent contrasted comparison of the two, which can be summarized as:

- PVM can work across heterogeneous networks and architectures. MPI does not support communication between two different architectures.
- Since PVM supports heterogeneous computing, it provides a number of functions to facilitate better resource management and load balancing. MPI functions are primarily concerned with messaging.
- In MPI, a group of tasks can be arranged in a specific logical interconnection topology. In a PVM application a programmer has to manually arrange tasks into groups according to the desired topology.

- PVM has more fault tolerant features than MPI. It provides functions to notify a task being added, deleted or exiting.
- MPI was designed to be thread-safe. PVM was not explicitly designed to work in a multithreaded environment and so programmers have to be very careful when writing multithreaded PVM applications.
- PVM can add or delete hosts during the application's runtime and so the resource management provided in PVM is dynamic in nature. MPI lacks such dynamics, which in turn helps in improving the efficiency of the MPI system.
- In PVM, programs in C, C++, or Fortran may freely intercommunicate whereas this is not possible in MPI even if they are executing on the same architecture.
- PVM provides a command line interface that can be used to probe the state of the different processes running or to add or delete hosts. MPI does not provide any such interface.

CHOICE OF PVM OVER MPI

In our implementation of the parallel machine, the following features of PVM made it a better choice over MPI for message passing.

PVM has the advantage for applications running on collections of hosts that are heterogeneous. The final parallel computer hardware for the national CODIS database search engine could be a set of heterogeneous hosts. This feature of PVM does not restrict the addition of new hosts of a different architecture.

PVM supports dynamic configuration, so nodes can be added or deleted when tasks are running. Also, PVM provides notification of each task's exit, addition or deletion. This is an important feature because it assists the replacement of failed nodes or the addition of new nodes.

In PVM, a C program can send a message that is received by a Fortran program and vice-versa. In contrast, a program written in C cannot communicate with a program written in Fortran under MPI, even if they are executing on the same architecture.

The parallel computer where the national database resides needs to be available essentially all the time, 24 hours a day and 7 days a week. Down time will result in backlogs of DNA profile data waiting for upload and search requests waiting for processing. The parallel computer needs to be designed in a manner that minimizes the potential for damaging down time and allows for easy isolation and repair of hardware flaws when they occur. Thus, PVM's fault tolerant features become more important. The ability to write long running PVM applications that can continue even when hosts or tasks fail, and that can adjust as loads change dynamically due to outside influences such as date and time, is quite important to heterogeneous distributed computing.

PVM CONCEPTS

In the following sections, we discuss some important concepts of PVM to get a better understanding of message passing under PVM.

PVM COMPONENTS

There are two main components of PVM: the PVM daemon and the PVM libraries.

PVM Daemon (pvmd3)

The PVM daemon, pvmd3 is a process that coordinates inter-process PVM communications [12]. The first daemon started is called the master daemon. This daemon starts all other PVM daemons in the parallel virtual machine. This master-slave relationship between daemons exists only during startup or reconfiguration. During normal operation they are all considered equal. Exactly one daemon per user runs on each machine configured into a parallel virtual machine. Other users, with their own parallel virtual machines, can have their own PVM daemons running on the same hosts.

PVM Libraries

The three PVM libraries are:

- libpvm3.a – a library of C language interface routines, which is always required.
- libfpvm3.a – a library that is required in addition to libpvm3.a for Fortran codes
- libgpvm3.a – a library that is required for use with dynamic groups

Library routines do not directly communicate to other processes. Instead, they send commands to the local daemon and receive status information.

MESSAGE PASSING IN PVM

PVM sends messages in three steps. The first step is to initialize a send buffer and clear older buffers. Any encoding required for the message is also specified in this step. PVM supports three encoding options – PvmDataDefault, PvmDataRaw and PvmDataInPlace. In PvmDataDefault, data is translated into the External Data Representation (XDR) [38] format and copied into a buffer. This option is chosen when communicating between heterogeneous hosts. PvmDataRaw option does not encode the data. It sends it in the raw format. Only machines that have the same native format can receive this data correctly. With the PvmDataInPlace option, data are not copied into the message buffer. Instead, only descriptors and pointers to the data are copied. This reduces the packing costs.

The second step in sending is packing. PVM provides functions to pack data into the send buffer. The last step is the actual call to a send function. PVM has an additional send and receive function for a fixed type in contiguous form that avoids buffering and makes message passing faster.

On the receiving end, messages are received in two steps. First a call to a receive function is made, and then the returned buffer is unpacked. The unpack call must match the pack calls. PVM provides blocking and non-blocking receive routines. It also has a timeout version for receive.

COMMUNICATION PROTOCOLS IN PVM

Every host in a parallel machine can connect directly to every other host using the Transport Control Protocol (TCP) and User Datagram Protocol (UDP). The communication protocols used in PVM are as shown in Figure 2.9.

There are three types of communications: between two PVM daemons (pvmd's), between a pvmd and its tasks, and between two tasks.

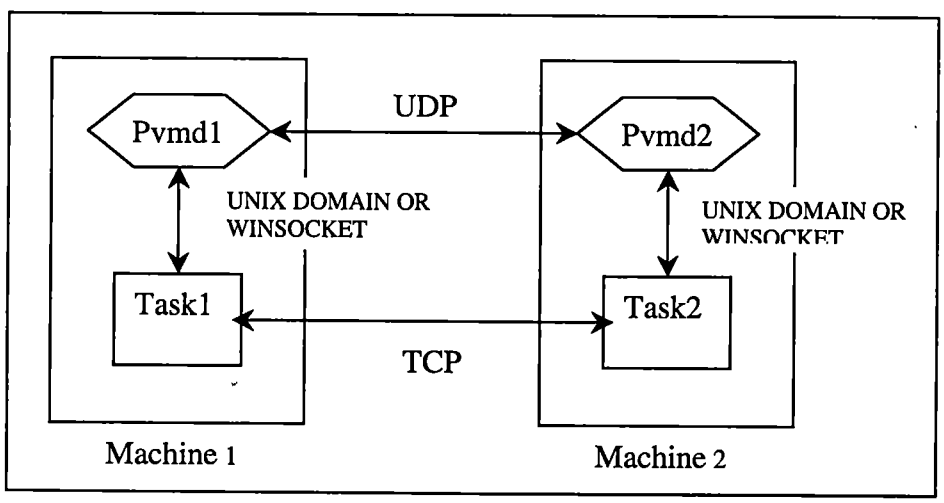


Figure 2.9: Communication Protocols in PVM.

Communication between two pvmds uses UDP. Even though TCP is a reliable protocol, UDP is used because of the 3 limitations of TCP. The first is scalability. Each TCP connection consumes a file descriptor in the pvmd. There is a limit to the number of opened files. On the other hand, a single UDP socket can communicate with a large number of remote UDP sockets. Since TCP is a connection-oriented protocol it has to set up a connection with every other pvmd in the virtual machine. This would mean establishing up to $n*(n-1)/2$ connections for an n-node virtual machine, which is expensive to maintain. The third reason is that it is easier to set timeouts in UDP to detect host, pvmd, and network failures.

UDP has its drawbacks. It can lose, duplicate or reorder packets. UDP also limits packet length so PVM has to break up long messages into smaller messages. PVM also implements some reliability features, and the PVM daemons implement an acknowledgment and retry mechanism. PVM maintains separate queues of packets being sent, packets waiting for acknowledgements and other packet types. The daemon also tracks round trip time and packet counts.

Communications between a pvmd and its tasks are through Unix Domain Sockets or the NT equivalent, Winsocket. In the earlier versions of PVM, TCP was used but Unix Domain Sockets results in faster communication.

Communication between two tasks can be accomplished in two ways – Normal mode (Figure 2.10) and Direct mode (Figure 2.11). In the normal mode, in order for a source

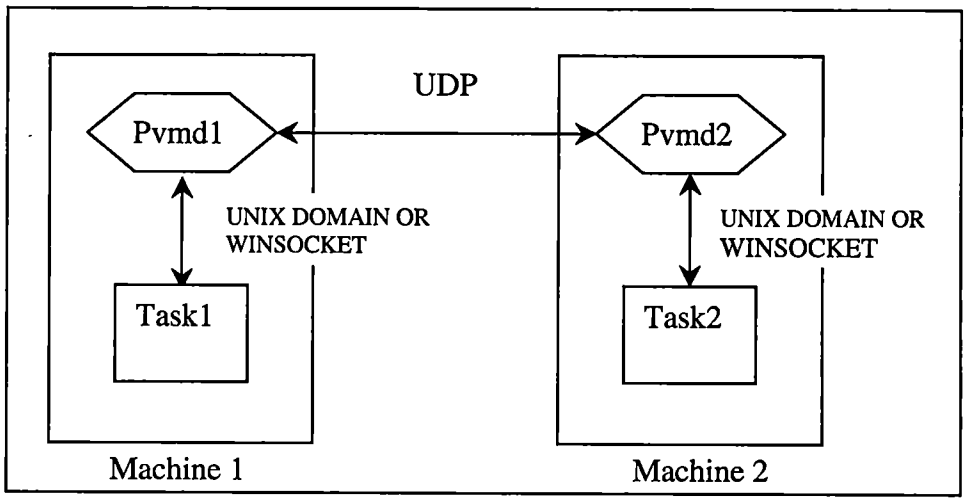


Figure 2.10: Communication between tasks in Normal mode.

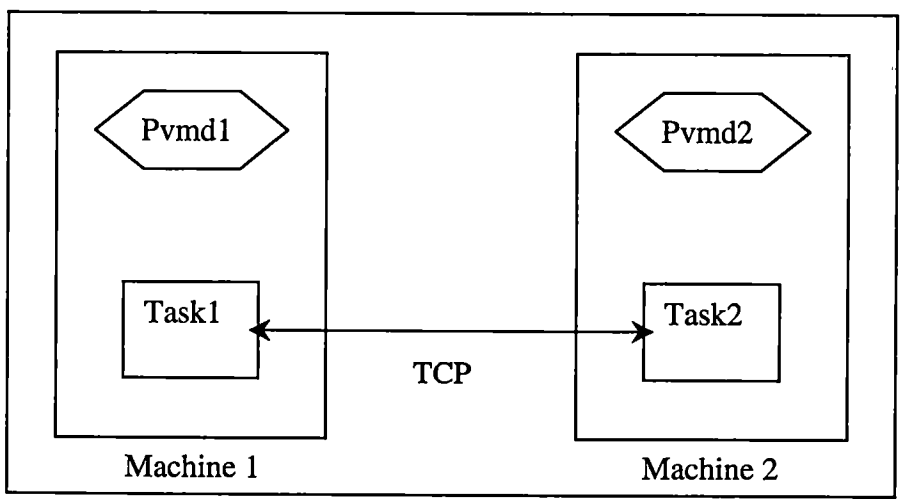


Figure 2.11: Communication between tasks in Direct mode.

task to communicate with a remote task, it must first communicate through a unix domain socket to its local daemon. The local pvmd then communicates through a UDP socket to the remote pvmd. The remote pvmd then communicates locally to the destination task through a unix domain socket. In the direct mode, a message can be directly sent from the source task to the destination task over a TCP connection by using pvm_setopt (). The second method reduces message routing overhead, but there are the drawbacks of TCP connections discussed above. In our benchmarks, the first method has been used though we did try the direct mode to see the effect on transfer rate. The results showed that direct routing made communication faster by 40% as compared to normal routing.

BOTTLENECKS IN PVM APPLICATIONS

The processor, operating system, network hardware, and PVM all impose limits on performance of an application.

Even if we use a fast switch in our application, the processor speed can be a limitation. The operating system can also affect the performance of PVM application. The TCP/IP protocol suite is the layer under PVM. For bulk data transfers, buffers in TCP tend to get full and congestion occurs. The congestion control scheme in TCP is explained in the following section. The layered structure of TCP/IP implies a certain number of memory-to-memory data movements. This introduces an overhead in our application. A network device driver attaches a network subsystem to a network

interface, prepares the network interface for operation, and governs the transmission and reception of network frames over the network interface. The device driver could also pose some restrictions but it could be tuned to suit the application.

The networking hardware also limits the performance of the application. The wire speed, buffering available in the switches, latency in switches and error rates are some of them. If the buffer available in the switch is not adequate, excess data is dropped which is highly undesirable. It could lead to retransmission of the data and thus affecting the transfer rate adversely.

The PVM daemon uses dynamically allocated memory to store message packets being sent from one task to another [24]. When the sending task sends out packets, it is accumulated in the daemon till the receiving task accepts the packet. The daemon does not limit the sending task from sending new packets, and so the packets can accumulate. If the receiving end is off or is not accepting the packets due to some other reason, the system could eventually run out of memory.

Another limitation that we need to keep in mind for our application is the maximum size of a PVM message. This is limited by the amount of memory available to the task. Also when a message is sent, the packets are stored in memory and so this reduces the available memory further.

CONGESTION CONTROL IN TCP

TCP connections provide reliable, stream-oriented, full duplex data transfer. In order to make TCP reliable, sequence numbers and acknowledgements are used. When TCP sends a large amount of data, it breaks the data up into smaller packets. At the receiving end, these packets may arrive in random order and must be reassembled. The sequence numbers are used in this process. Once the packet is received, the receiver sends an acknowledgement (ACK) packet with the packet's sequence number to the sender. This assures the sender that the packet has been received. If the packet does not reach the destination, the sender waits for a certain amount of time (which is a function of the approximate round trip delay time), and if it does not receive an ACK packet within this time interval, it retransmits the packet.

TCP uses a sliding window scheme for flow control. The window is the maximum number of packets that can be sent without receiving an ACK for the first packet. In the sliding window scheme, all new segments in the window are transmitted, and as ACKs are received the window is repositioned and equal number of new segments is transmitted.

To prevent congestion and collapse of the transmission data rate, TCP implements three mechanisms – multiplicative decrease, slow start and congestion avoidance [14]. When a packet times out, meaning that no ACK has been received within the allowed waiting time, TCP assumes the timeout to be a result of congestion, and reduces the

window size by half. With this multiplicative decrease, traffic reduces exponentially, quickly and effectively responding to network congestion.

When one ACK is lost, the packets that follow the lost packet cannot be ACK'ed as ACKs are cumulative. When congestion is reduced, TCP starts receiving ACKs for the packets transmitted. Since a large number of ACKs are likely to be received, this results in a large number of packets being transmitted, which could exceed the network's capacity and result in congestion again. To avoid this, TCP implements a slow start method.

With slow start, TCP starts with a window size of one packet and increases the window by one each time a packet is acknowledged. Once the sender receives one ACK, the window size increases to two. The sender then sends two packets. When both of those packets are acknowledged, the congestion window is increased to four. This process continues, and the window size increases exponentially as a function of round-trip time.

Exponential increase in the window size can lead to congestion. This is avoided by forcing a linear window increase after the window reaches half the size when congestion first occurred. The window size keeps increasing till either data packets start getting lost or the maximum window size allowable (usually 2^{16} bytes) is reached.

Figure 2.12 explains the congestion control mechanism in TCP. If we assume that congestion first occurred when the window size was 64, then slow start is used until the window size reaches 32. After that there is a linear increase in window size to avoid congestion collapse.

NETWORKING TECHNOLOGIES

The performance of cluster computing depends largely on the bandwidth and latency of the inter-cluster network. Achieving high bandwidth and low latency requires not only fast hardware but also efficient protocols and host-network interfaces that minimize the communication software overhead.

Several types of networks are used in cluster computing. These networks include Ethernet, Fast Ethernet, Gigabit Ethernet, Myrinet, ATM, HiPPI and Fiber Channel. We have examined Fast Ethernet, Myrinet and ATM OC-3c in this research. The methods should be adaptable to other network topologies that support PVM.

Several issues have to be considered when designing a cluster interconnect. The most important in this application is the price to performance trade off. Scalability is another crucial issue. It refers to the network's ability to support an almost linear increase in performance with the number of nodes. In addition, the cluster interconnect should provide a reliable network without additional software overhead for safe data transmission.

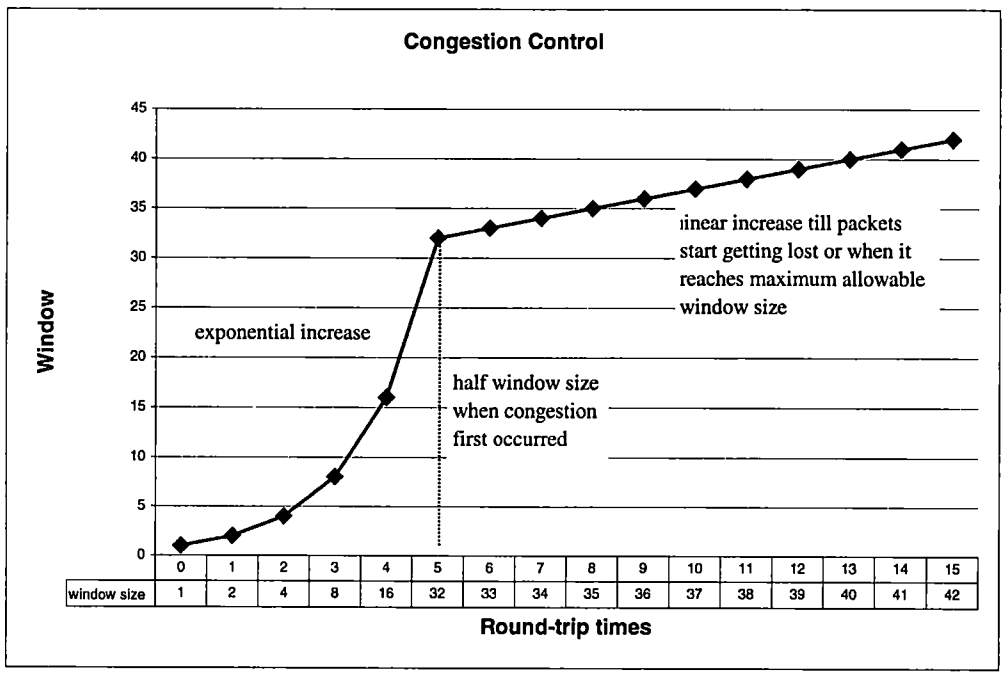


Figure 2.12: Congestion control in TCP.

FAST ETHERNET

Fast Ethernet is a local area network transmission standard that provides a data rate of 100 megabits per second (Mb/s). In shared Ethernet technology, data packets are visible to every other device in the network on a shared medium. Shared network media create unnecessary traffic and reduce network performance. Total aggregate bandwidth is limited to 100Mb/s; achievable bandwidth is significantly less (30-40Mb/s) due to contention. Switched media are becoming more prominent with the increasing size of networks and demand for network bandwidth.

Switches work by dividing a single large network into smaller, less congested LANs and passing data packets from one segment to another. A switch provides the capability to increase the total LAN bandwidth because it allows simultaneous switching of packets between its ports. The switch determines the exact destination for the data and transmits it only to the segment where the device is attached. A switch can also handle multiple transmissions at one time. Fully switched networks are becoming common as equipment costs continue to fall, where every host's connection to the network is a dedicated point-to-point link between host and a network switch. When point-to-point links are used, the links can be bi-directional. Since there is no contention on each link, most of the available bandwidth is usable. Switched Fast Ethernet links operate at 200Mb/s (bi-directional or full duplex, with 100Mb/s in each direction).

In a switched network, bandwidth limitations generally occur at a higher level in the network topology. For example, a 32-port Fast Ethernet switch could see transient traffic to and from all of its ports simultaneously of 3.2Gb/s (32 x 100Mb/s, assuming all traffic originates and terminates at hosts directly connected to the switch). If the switch's design allows a maximum aggregate bandwidth of 2.5Gb/s (a common limitation in low-cost switches), 600Mb/s of the traffic cannot be served. In this case, packets will be dropped by the switch causing timeouts in the TCP/IP implementations in the hosts and a subsequent reduction in traffic.

Two types of architectures determine switching applications and performance: cut-through and store-and-forward [28].

- Cut-through switching starts sending packets to the destination port as soon as they enter a switch. This reduces transmission latency between ports. In this architecture, there is no error check and so it can propagate bad packets to the destination port.
- Store-and-forward switching buffers incoming packets in memory until they are fully received. A cyclic redundancy check (CRC) is run on the packets. Since the packets are buffered in memory, this type of operation has high latency that is proportional to the frame (packet) size. This latency reduces bad packets and collisions that can adversely affect the overall performance of the segment.

A switching technique called threshold or adaptive switching takes advantage of both of the above techniques. The switch first starts with cut-through switching but

implements a CRC check. In case there is large number of errors, a store-and-forward technique is used until the error rate is reduced. Once this occurs, the switch changes modes to cut-through switching.

Network switches typically contain a small amount of memory to support store and forward operations. This allows the switch to temporarily queue packets while waiting for opportunities to send (forward) the packets to their destinations. Some manufacturers advertise the memory buffer size available to support store and forward operations, but others do not. Manufacturers may also advertise 'non-blocking' performance or bandwidth, which is the peak bandwidth the switch can support without restoring to store and forward operation. A few vendors provide equipment with sufficient non-blocking bandwidth to guarantee that all packets will be properly routed; this is the exception rather than the rule. The most important criterion for selecting a switch for our application is that the switch should not drop packets. A second criterion is the latency of the switch.

ATM

Asynchronous transfer mode (ATM) is a high-performance switching technology. ATM breaks up the data to be sent into 53 byte cells and transports it over the physical layer. At the receiving end, the cells have to be reassembled.

Every ATM cell consists of a 5-byte header and 48-byte payload (Figure 2.13). This is an approximate overhead of 100 x 5/53 = 9.43%.

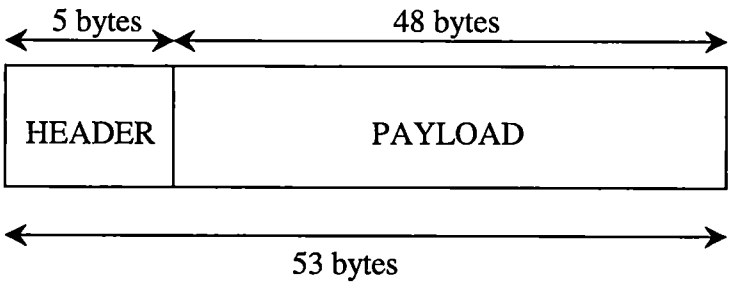


Figure 2.13: ATM cell.

The header consists of the following fields – flow control (4 bits), virtual path identifier (8 bits), virtual channel identifier (16 bits), payload type (3 bits), cell loss priority (1 bit) and cyclic redundancy check (8 bits). The 48-byte payload contains the information that has to be transmitted. Since the cell is of fixed size, long frames never delay communication.

ATM is a switched network technology, and all links are point-to-point. Since the cells have a fixed format and a fixed length, switching is simpler than that for Ethernet, Fast Ethernet and Gigabit Ethernet. Also since it is a switch-based technology, ATM gives higher aggregate bandwidth than a shared resource solution.

ATM is a circuit-switched, rather than a packet-switched, technology. Calls (circuits) may be set up and brought down dynamically, but this process incurs a performance penalty. For data networks, ATM networks frequently employ LAN Emulation (LANE), which imposes its own performance limitations, as discussed in the section on LANE.

An important feature of ATM is the sender's ability to negotiate with the network for a connection, or virtual circuit, to its destination. It can specify the speed of connection, the quality of service and the type of service. This is useful since ATM can be used for all traffic types - voice, data and video and each of them has different requirements.

ATM Architecture

ATM is a layered architecture allowing multiple services such as voice, data and video, to be mixed over the network. The different layers are shown in Figure 2.14. The Adaptation layer assures the appropriate service characteristics and divides all types of data into the 48-byte payloads that make up the ATM cell. These are passed down to the ATM layer that appends the 5-byte header information.

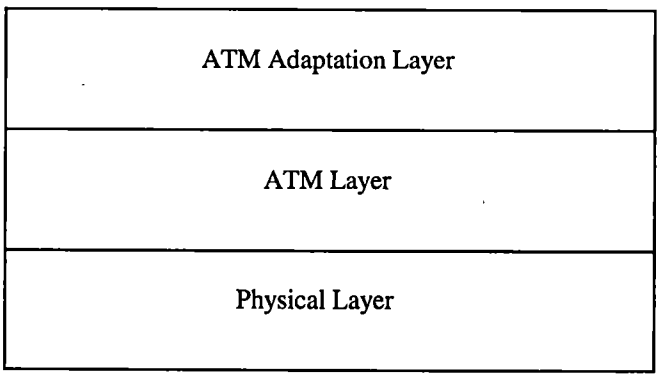


Figure 2.14: The ATM Network Architecture.

The header ensures that the cell is sent on the right connection. The Physical layer receives the cell from the ATM layer, converts it into electrical signals and passes it over the physical network.

LAN Emulation (LANE)

There has been a growing trend towards ATM networks due to its high bandwidth and scalability. ATM supports services for all traffic types – video, data and voice. A large portion of the Internet backbone and the high bandwidth connections to this backbone are now ATM connections. However, although ATM to the desktop was initially considered to be a leading contender for next generation networks, switched Fast Ethernet and Gigabit Ethernet have largely overtaken its role. To a large extent, this is due to the complexities and limitations of LANE over ATM. The basic architecture of ATM differs from the traditional LAN technologies. While ATM is a connection-oriented protocol, LANs are traditionally connectionless. In order to use the available LAN resources, it is necessary to emulate the services of existing LANs across an ATM network. Emulation of some services, such as multicast, is quite difficult under ATM.

LANs are connectionless whereas ATM technology is connection oriented. Broadcast and multicast are features of traditional LANs and since ATM is connection oriented, a different approach has to be used for these features. LAN MAC addresses are stored in the adapter card and so do not change. An ATM adapter's address is determined by

the switch to which it is attached. If ATM is used with existing LANs, it must have a database to map from LAN addresses to ATM addresses.

LANE follows a client/server model, with multiple clients connecting to LAN Emulation components. LANE defines three different types of components: the LAN Emulation Server (LES), the Broadcast and Unknown Server (BUS), and the LAN Emulation Configuration Server (LECS). The functions of each of these components are summarized in Figure 2.15.

LANE provides MAC to ATM address resolution. This introduces latency, thus affecting the network performance. LANE uses network layer routing. As the number of hosts increase, the computational requirements to calculate routes among different hosts and the memory requirements to store these routes also increases. It could exceed the capacity of the routers and so scalability is limited. The broadcast server could be another performance limitation. Since this implementation does not use any ATM point-to-point connections, it provides a negligible advantage over a traditional LAN.

MYRINET

Myrinet is a local area networking technology based on the technology used for packet communication and switching within massively parallel processors. A Myrinet link is a full-duplex pair of 1.28Gb/s point-to-point channels [25].

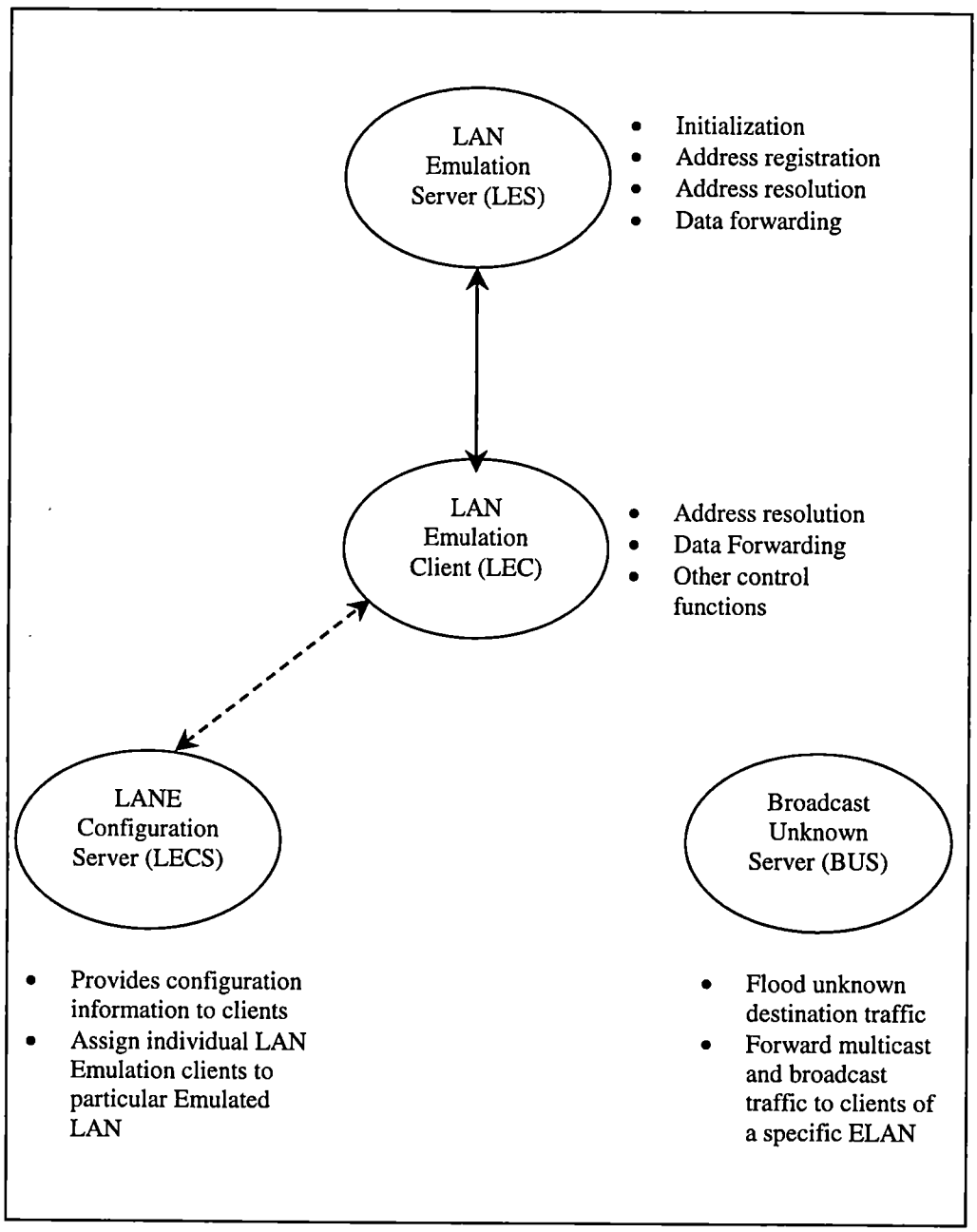


Figure 2.15: LANE Components.

A data-communication, flow-control unit in Myrinet is called a flit. Each flit is either a byte of data or a control symbol. Control symbols are used for flow control, to mark packet boundaries, and to indicate error conditions. Myrinet link has very low bit error rates, usually below 10^{-15}.

A Myrinet host interface is controlled by a general-purpose microprocessor called a LANai. The LANai executes a Myrinet control program (MCP) in order to handle sending, receiving and buffering of packets. It also performs network monitoring and mapping functions.

A Myrinet packet consists of a header, payload and a trailer. The header is a minimum of four bytes long. It consists of the source route (zero or more bytes) and the packet type (4 bytes). The payload of a packet is of arbitrary length. Myrinet computes a CRC on the entire packet including the header at each link because the header is modified at every switch. If the CRC on a packet is incorrect when it enters a switch, it will be incorrect in the same bit positions when it leaves the switch. Thus, if there is an error on any link on a routing path, the error can be detected at the destination. The CRC is sent in the trailer. Two packets are separated by the GAP control symbol.

Myrinet uses a crossbar switch and cut-through routing. When a packet reaches the switch, the destination port is extracted from the header. Myrinet switches use a relative addressing scheme to route packets from one port to another inside the switch. The output port is interpreted as an offset from the message's input port.

If the output port required by a message is being used, the switch uses flow control to block the incoming message. Flow control is accomplished on Myrinet channels by the receiver injecting STOP and GO control symbols into the stream being produced by the sender of the opposite-going channel [34]. Flow control applies only to data packets. Control symbols cannot be blocked using flow control and have higher priority over the data packets. Every link has flow control, so the sender will eventually be blocked if the output port remains unavailable. This protocol ensures that the network never drops a message. Contention for a receiver can reduce the flow of traffic from a sender. This may be problematic in the design of parallel algorithms. The absence of buffering means the effect of network loads is passed back at least to the host/network interface of the sender. This will probably occur more frequently in a Myrinet based parallel machine than one using store and forward technology. Whether this affects performance is unknown. This is one of the effects this research attempts to measure.

CHAPTER 3

EXPERIMENTAL DESIGN

ARCHITECTURE

The recommended implementation of the CODIS parallel search server is a multi-level parallel machine. The machine will consist of a number of Linux clusters with 4-8 hosts in each cluster. The number of clusters will depend on the size and complexity of the database.

The system consists of a root server, which divides the database and the processing load between the various clusters, and one or more clusters of 4-8 hosts as shown in Figure 3.1. The entire system operates using the Parallel Virtual Machine (PVM) parallel-computing environment.

Each cluster consists of 4-8 nodes. Each node may have dual processors and will implement a database Search Queue. Within the cluster the hosts will be interconnected using 100 base-TX Fast Ethernet through redundant network switches as shown in Figure 3.2 and Figure 3.3.

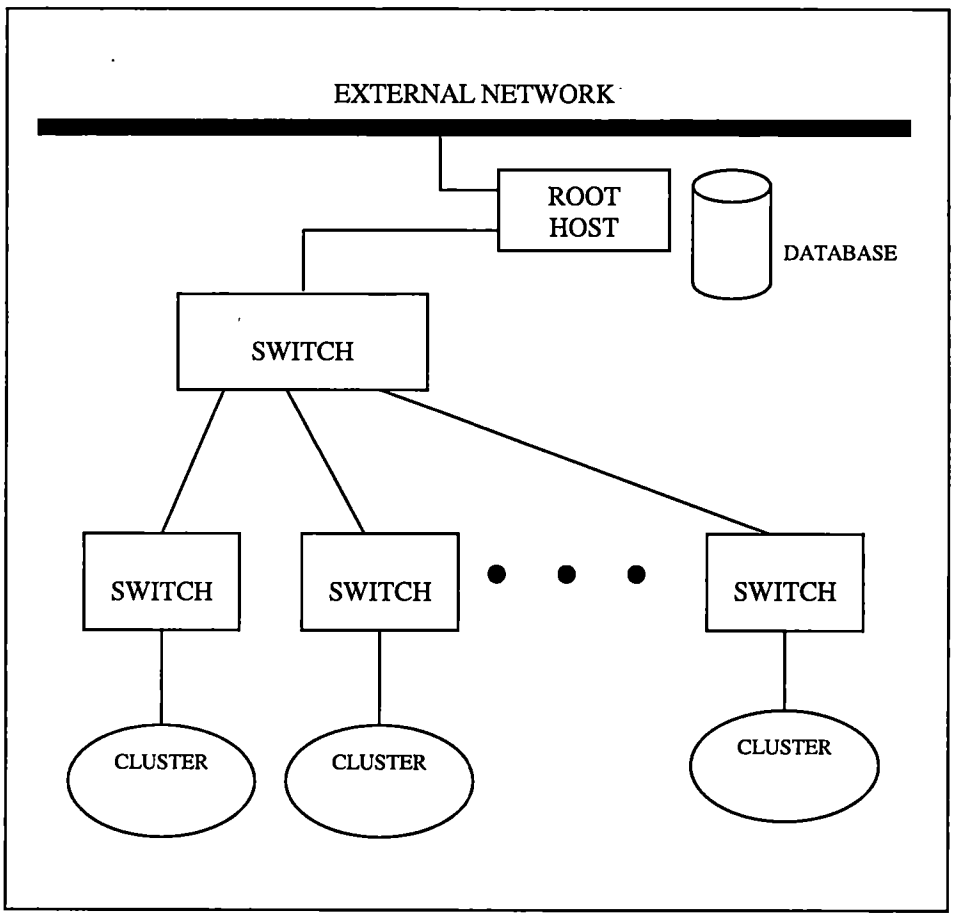


Figure 3.1: Recommended implementation of CODIS parallel search server.

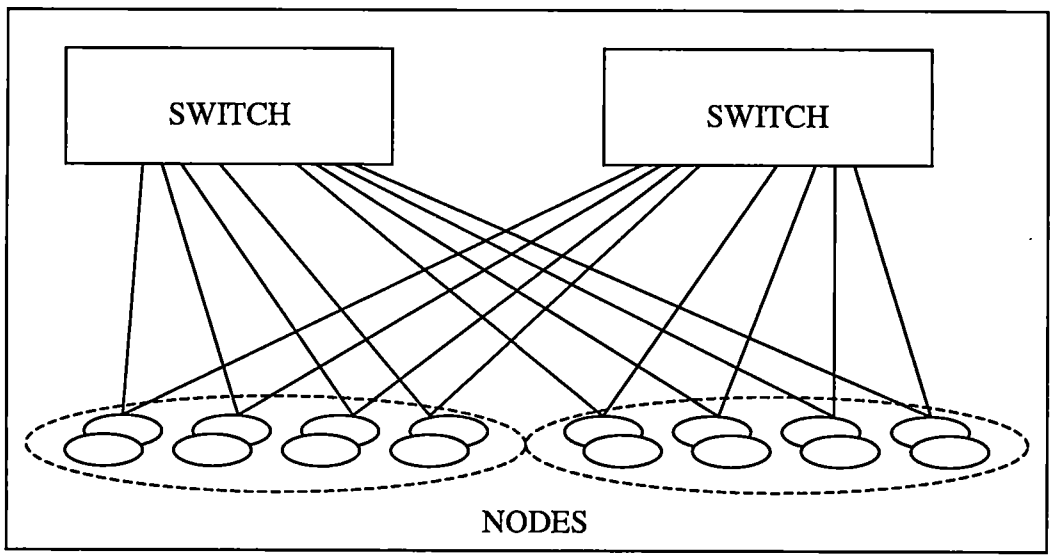


Figure 3.2: Redundant nodes connected through redundant network switches.

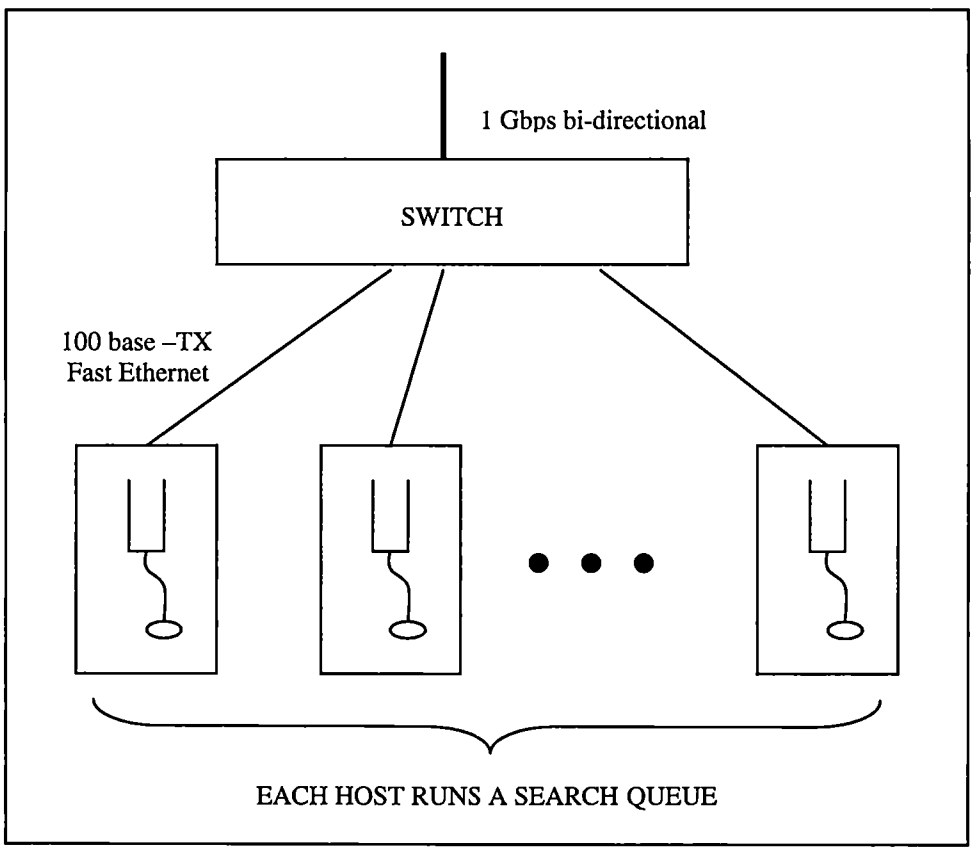


Figure 3.3: Architecture of each cluster.

The CODIS database will be partitioned and each partition will reside on the hosts of one cluster. Within each cluster, searches will be performed in parallel through load balancing and the generation of search requests. Searches initiated by the root host will be performed in parallel on independent partitions of the database across all clusters.

A high performance network switch, most likely providing Gigabit Ethernet service to each cluster's network switches will connect the parallel machine's clusters and its root host. This switch is anticipated to provide at least 16Gbps non-blocking bandwidth. To address security issues, either the root host or a network switch will be configured as a network filter to allow packets to or from the external network to only travel from or to the root host. The clusters of hosts will not have access to the external network.

TEST MACHINES

The benchmark tests for Fast Ethernet and Myrinet were done on identical machines to give an objective comparison for the two interconnects. Eight nodes of the Tennessee/Oak Ridge Cluster (TORC) [26] were used. TORC is a set of dedicated clusters of Linux/NT PCs used for research in parallel computing and applied math and scientific computations. The compute nodes used were dual processor Pentium III 550MHz computers running Red Hat Linux release 6.1.

The Fast Ethernet interface uses a 16-port Fast Ethernet switch (BayNetworks 350 T). This switch cost $1744 US when it was purchased. The network interface card is the 3Com EtherLink 10/100 PCI NIC, which cost approximately $133/node.

The Myrinet interface uses an 8-port Myrinet switch with 4 LAN ports and 4 SAN ports and PCI LANI 4.1 cards. The switch costs $1600 and the network interface card cost approximately $1300/node when they were purchased.

The machines (torc1 through torc8) used in the benchmark tests are identical. Specifications of these machines are given in Table 3.1.

Table 3.1: Specifications of the nodes in TORC cluster.

Architecture	Dual processor Intel Pentium III
Clock Rate	550MHz
Operating System	Red Hat Linux release 6.1
Memory	512M
Network Connection	Switched Ethernet / Myrinet

The same tests were repeated on the Fast Ethernet and ATM cluster in Laboratory of Information Technology (LIT) at University of Tennessee, Knoxville. The Fast Ethernet interface uses an 8-port Fast Ethernet Switch (Netgear 508), which cost approximately $400. A 16-port switch from Netgear, the Netgear 516 cost $530. Therefore the cost of this switch is almost half that of the switch used in the TORC cluster. Also the network interface card used is the Netgear FA310, which costs less than $25/node. The ATM infrastructure LIT uses is the Fore Systems ASX-200BX that cost roughly $20-30K. Specifications of the machines in the LIT cluster are summarized in Table 3.2.

PERFORMANCE TESTS

A new benchmark program was developed to evaluate the effect of various network interconnection strategies on Parallel Virtual Machine (PVM) performance. The following network interconnects were evaluated: Fast Ethernet, Asynchronous Transfer Mode (ATM OC-3c) and Myrinet. We found that for small clusters (8 or less hosts) and the communication strategy implemented in the benchmark, the costs of ATM and Myrinet are not justified by a corresponding performance improvement. We also observed the effects on the transfer rate of encoding instead of using the raw format and of choosing the direct route option instead of normal route.

Table 3.2: Specifications of the nodes in LIT cluster.

Host	Architecture	Clock Rate	Operating System	Memory	Network Connection
stout	Dual processor Sun E450	250MHz	SunOS 5.8	256M	ATM OC-3c
spliff	Sun Ultra 1/140	143MHz	SunOS 5.8	320M	ATM OC-3c
ice	Sun Ultra 1/140	143MHz	SunOS 5.8	160M	ATM OC-3c
linden	Sun Ultra 1/170	166MHz	SunOS 5.7	128M	ATM OC-3c
dust	Sun Ultra 1/170	166MHz	SunOS 5.7	128M	ATM OC-3c
zero	Dual processor Intel Pentium III	667MHz	Red Hat Linux Release 6.2	512M	ATM OC-3c
one through five	Intel Celeron	333MHz	Red Hat Linux Release 6.0	256M	SW 100base-TX
six	Intel Celeron	333MHz	Red Hat Linux Release 6.2	128M	SW 100base-TX
gum	Dual processor Intel Pentium III	350MHz	Red Hat Linux Release 6.2	256M	SW 100base-TX
sycamore	Dual processor Intel Pentium II	333MHz	Red Hat Linux Release 6.1	256M	ATM OC-3c
chestnut	Dual processor Intel Pentium II	333MHz	Red Hat Linux Release 6.2	256M	ATM OC-3c

The software exchanges data back and forth from every node simultaneously to simulate the stress on the network fabric that is expected in the parallel search engine. Each host measures the average, minimum and maximum times required to exchange data between itself and every other node in the PVM environment. The node initiating the send starts the timer, initiates a transfer and stops the timer when it receives the data back, so the round-trip transfer time is measured. This also avoids the problem of synchronizing clocks on two different machines. The data sizes vary from 40bytes to 40Mbytes. In order to ensure that anomalies in message timings are minimized the tests are repeated 20 times for each message size. In order to measure the latency or the timing overhead to establish communication, messages with a data size of zero bytes are transferred between the nodes. An overview of the software is shown in Figure 3.4.

The tests were repeated with a varying number of nodes to compare the effect on bandwidth and latency of the number of nodes. Additional runs were done to evaluate the effect of data encoding and also of using the direct routing option in PVM communication.

From these timing studies, the following performance metrics can be obtained:

- the time taken to transfer zero bytes gives a measure of the latency (L), or timing overhead required to establish and manage the communication process, and

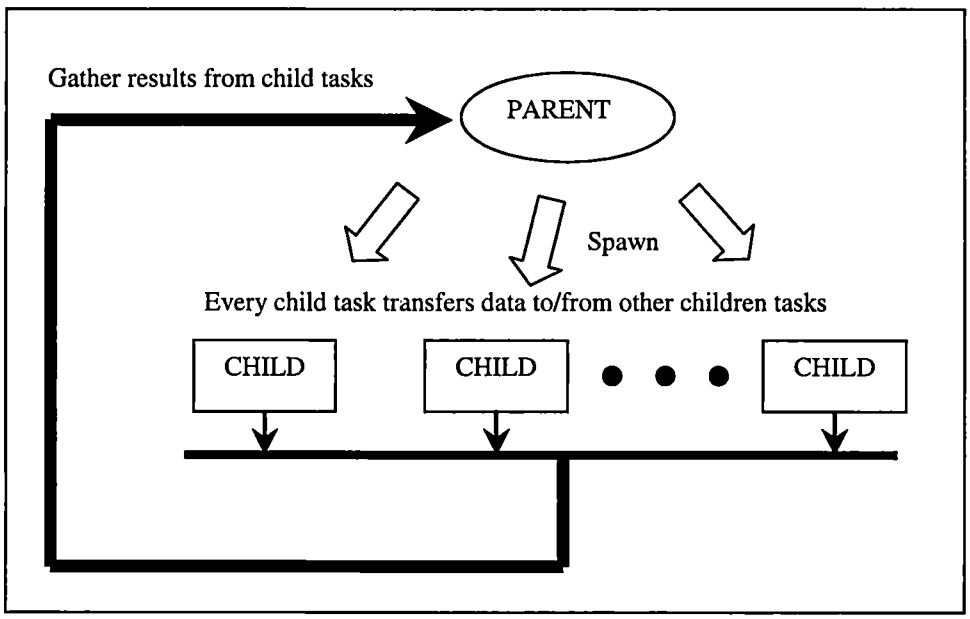


Figure 3.4: Overview of the benchmarking software.

- the average round trip time ($2D$) of b bytes of data can be used to characterize the end to end delay $D(b)$ of the messaging system for a message of size b

$$D(b) = L+b/B$$

where the asymptotic bandwidth is $B = b_L/D_L$. Asymptotic bandwidth is calculated by transferring a large amount of data, b_L bytes. The average round trip time corresponding to it is approximately $2D_L$. When a large amount of data is transferred, the effect of latency is minimized.

XPVM

XPVM [24] provides a graphical interface to the PVM commands. It can be used to analyze the performance of a particular application and so could be used as a management utility.

Figure 3.5 shows the Network view. This view displays the high-level activity on each of the hosts in the PVM environment. Each host is represented by an icon, which includes the architecture and the name of the host. In Figure 3.5, when an icon is white in color, it depicts that there are no tasks on a given host. Yellow color indicates that at least one task is executing PVM system routines. If the icon is green, it indicates that at least one task is executing user computation.

The Space-Time view shows the status of individual tasks as they execute across all hosts. The colors depict the same things as in network view. In this view there are additional red lines, which depict message transfers as seen in Figure 3.6.

The Utilization View summarizes the space-time view at each time instant, showing the number of tasks computing, in overhead, or waiting for a message at any given time. This information is represented by three colored rectangles, stacked vertically at each time instant, with computing on the bottom, overhead in the middle, and waiting on top as seen in Figure 3.7. The default colors are green for computing, yellow for overhead, and red for waiting.

Each of these views can be used to obtain a better understanding of the application's execution. It can be used to identify the areas where overheads are more and to improve the performance.

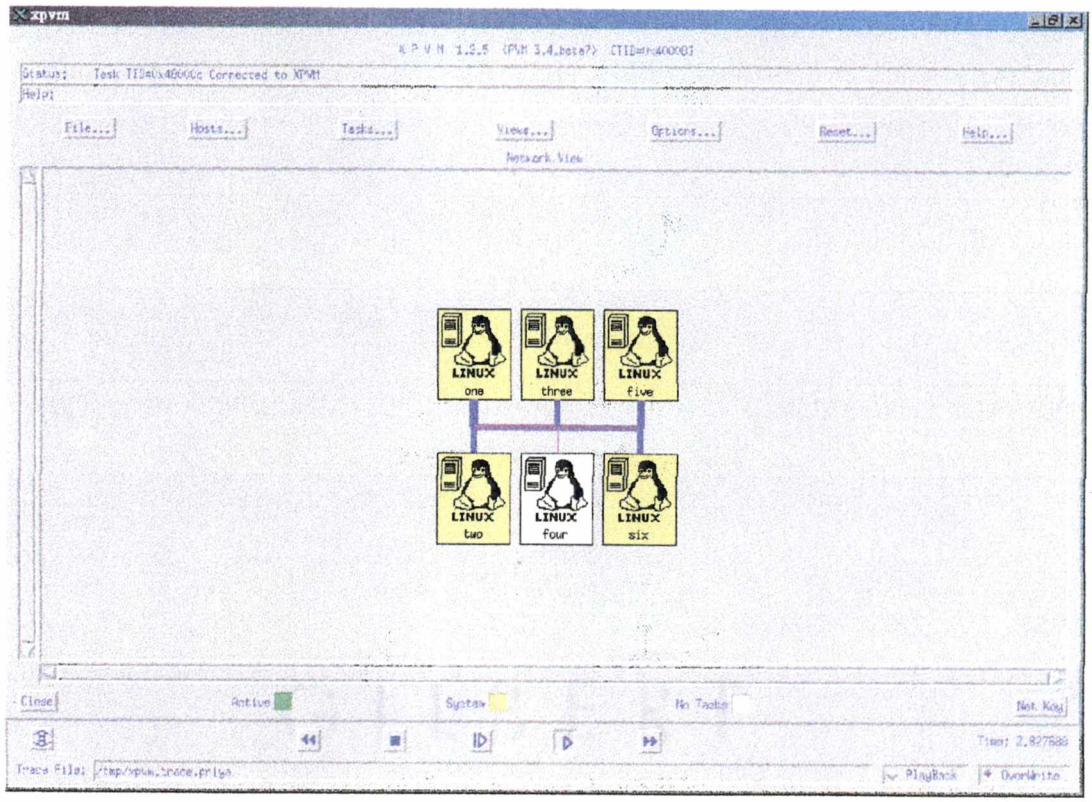


Figure 3.5: Network View.

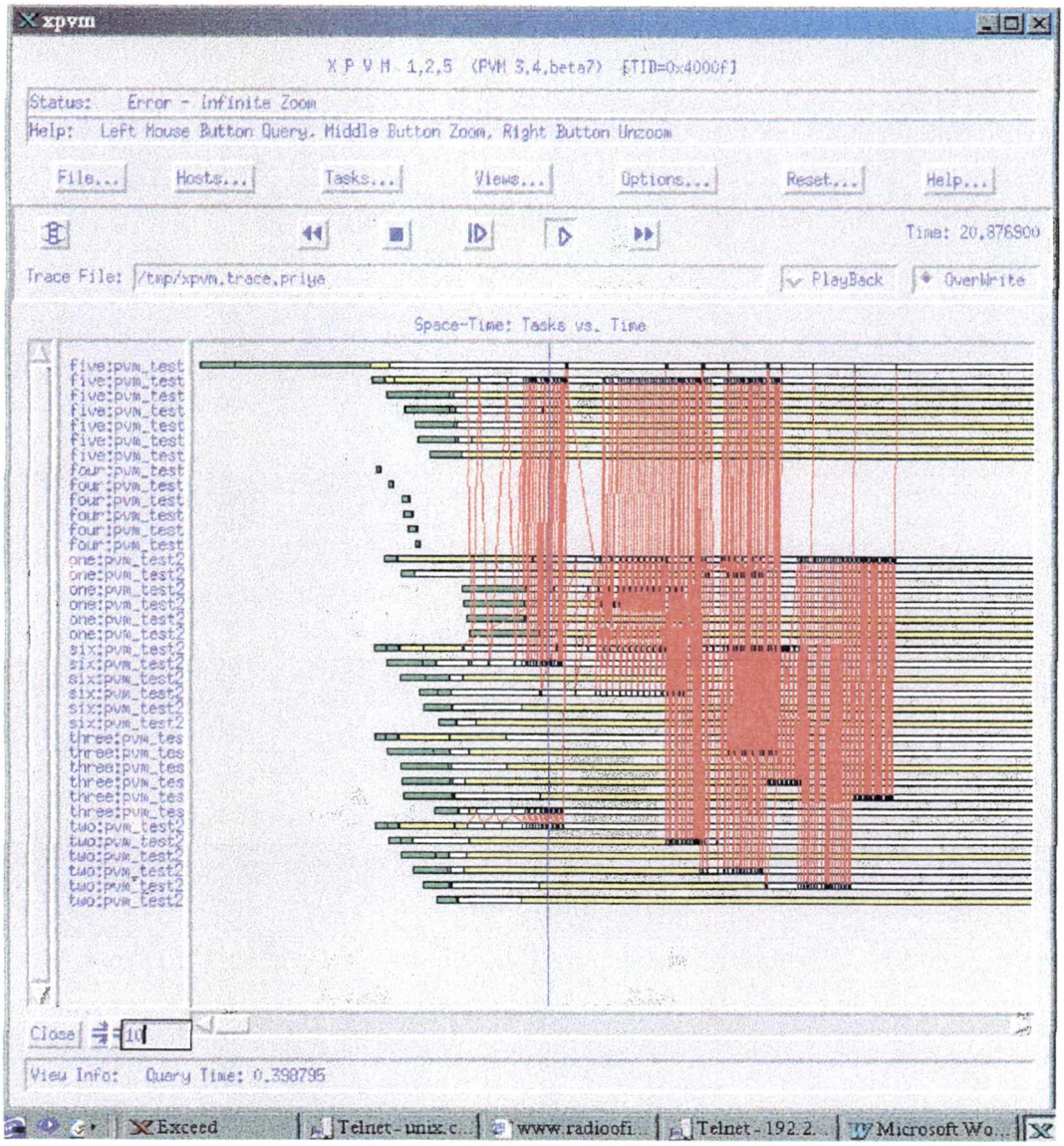


Figure 3.6: Space-Time View.

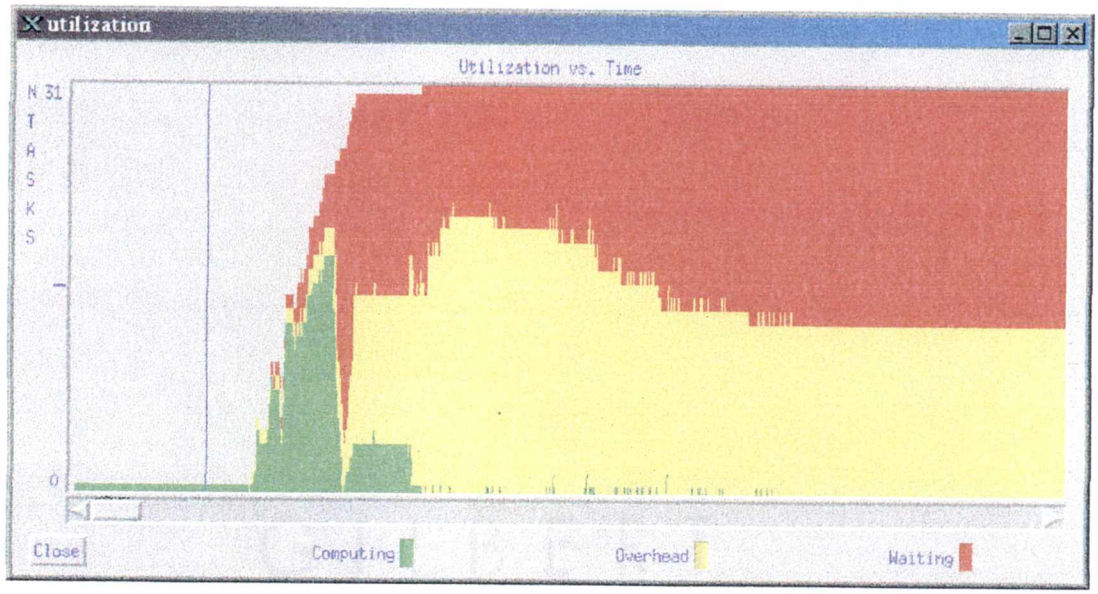


Figure 3.7: Utilization View.

CHAPTER 4

RESULTS AND DISUSSIONS

The timing results of the benchmark are used to build an Excel pivot table. An example of such a pivot table is shown in Figure 4.1. The timing results have also been represented in the form of bubble charts [A.1-A.7]. An example bubble chart is shown in Figure 4.2. A description of how one should interpret the bubble charts is explained in the appendix. Bubble charts give a better view of the round trip times between two nodes. The area of the circle is proportional to the round trip time. A smaller circle is always desirable. When calculating the average round trip times, measurements that are significantly smaller or larger than the average, are neglected. This minimizes the effect of variations in the network traffic.

In the following sections, we evaluate the effect of encoding and direct routing on transfer rates for Fast Ethernet, ATM and Myrinet. We also study the latency and the transfer rates for each of these networking technologies. The benchmark tests include the comparison of Fast Ethernet and Myrinet on the TORC cluster, ATM and Fast Ethernet on the LIT cluster. The comparison of ATM and Fast Ethernet on the LIT cluster is limited because of the differences in the processors used for the benchmarks.

Average of Average		To				
From	Length	torc1	torc2	torc3	torc4	Grand Total
torc1	10	0.00056285	0.0010875	0.00377245	0.0011048	0.0016319
	100	0.000597	0.0014975	0.001101	0.0012469	0.0011106
	1000	0.0007269	0.0027814	0.0025913	0.0029256	0.0022563
	10000	0.00403	0.0238441	0.0249794	0.0255792	0.019608175
	100000	0.0496516	0.251886	0.268647	0.265269	0.2088634
	1000000	0.479179	2.44642	2.58829	2.68785	2.05043475
	10000000	4.88151	22.5655	27.533	28.7011	20.9202775
torc1 Total		0.77375105	3.613288071	4.34605445	4.526439357	3.314883232
torc2	10	0.0010564	0.0006448	0.0033613	0.00103465	0.001524288
	100	0.001206	0.0006007	0.00110915	0.00122515	0.00103525
	1000	0.00272665	0.00082845	0.0025631	0.0026573	0.002193875
	10000	0.0227084	0.0045754	0.0239553	0.0242841	0.0188808
	100000	0.23966	0.0544529	0.255194	0.238586	0.196973225
	1000000	2.39364	0.46909	2.49731	2.49629	1.9640825
	10000000	24.4374	4.69811	26.278	26.4468	20.4650775
torc2 Total		3.871199636	0.746900321	4.151641836	4.172982457	3.235681063
torc3	10	0.00106505	0.0010559	0.00049845	0.00139395	0.001003338
	100	0.00118695	0.0013969	0.0005069	0.00128035	0.001092775
	1000	0.00263635	0.0037856	0.00074145	0.0029703	0.002533425
	10000	0.0204525	0.023496	0.00292925	0.0264607	0.018334613
	100000	0.215889	0.258407	0.0372984	0.275224	0.1967046
	1000000	2.09506	2.40955	0.363021	2.70057	1.89205025
	10000000	21.455	25.1896	3.7725	28.8193	19.8091
torc3 Total		3.398755693	3.983898771	0.596785064	4.546742757	3.131545571
torc4	10	0.0016182	0.00100785	0.0008564	0.00030655	0.00094725
	100	0.000998	0.00125155	0.00109235	0.0003242	0.000916525
	1000	0.00234925	0.0026558	0.002597	0.00046275	0.0020162
	10000	0.0220697	0.0238206	0.0210936	0.0028549	0.0174597
	100000	0.237099	0.238885	0.224271	0.0367746	0.1842574
	1000000	2.33475	2.46307	2.16753	0.365577	1.83273175
	10000000	23.5167	25.7966	23.1505	3.69002	19.038455
torc4 Total		3.730797736	4.075327257	3.652562907	0.585188571	3.010969118
Grand Total		2.943626029	3.104853605	3.186761064	3.457838286	3.173269746

Figure 4.1: Excel Pivot Table.

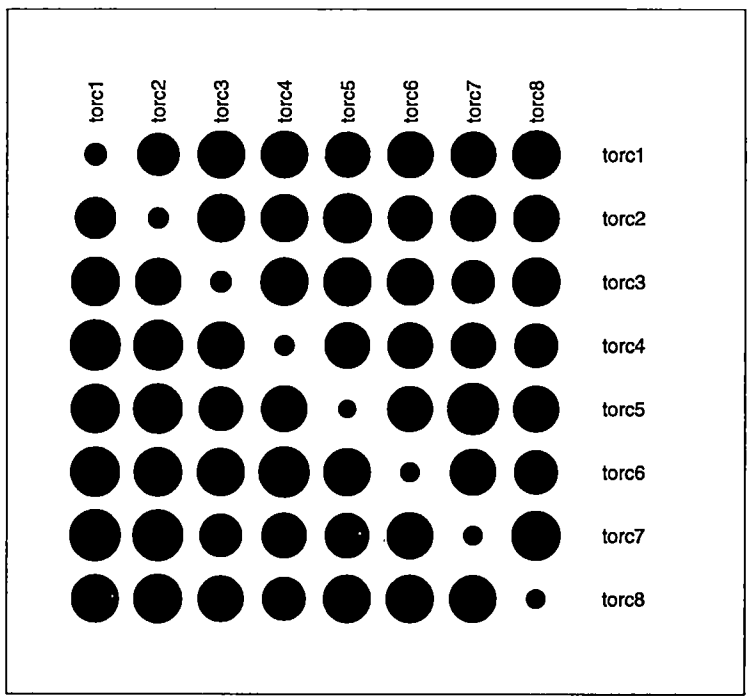


Figure 4.2: Example Bubble Chart

TIMING SCENARIO I: EFFECT OF ENCODING

Two separate runs of the benchmark software were performed on 8 nodes with the Fast Ethernet interconnect on the TORC cluster – one with data being transferred in the raw format and the other with data encoding. The raw format can be used only when a homogeneous cluster is used.

The comparison of delay times with encoding and using raw format for varying message sizes is shown in Table 4.1. Figure 4.3 depicts the delay and Figure 4.4 shows the transfer rates obtained with raw and encoded formats.

As expected, sending data using raw format is more efficient than encoding it. From the results, it can be seen that on an average, the raw format improves communications data rates by 20% over the encoded transfer method. The improvement in performance when using raw format is shown in Figure 4.5. The raw format may not be used, however, if the cluster is heterogeneous.

When sending data, PVM has to allocate the buffer, insert the buffer in sending queues and pack data in the buffer in an architecture independent format (if it has not been explicitly set to send them in a raw format). This requires time. Comparing the columns for delay with encoding and delay with raw format, we can see that for short messages there is no big improvement, probably because buffer management takes a significant fixed amount of the time.

Table 4.1: Comparison of delay times with encoding and using raw format.

Bytes Transferred	**Delay with encoding (s)**	**Delay with raw format (s)**	**Ratio (encoding/raw)**
40	0.0007254	0.00071105	1.020223051
400	0.0005705	0.00054914	1.038900741
4K	0.0013948	0.00118727	1.174822294
40K	0.0133012	0.01065752	1.248061082
400K	0.1367898	0.10972358	1.246676764
4M	1.3791657	1.10355937	1.249743056
40M	15.285612	11.3410066	1.34781794

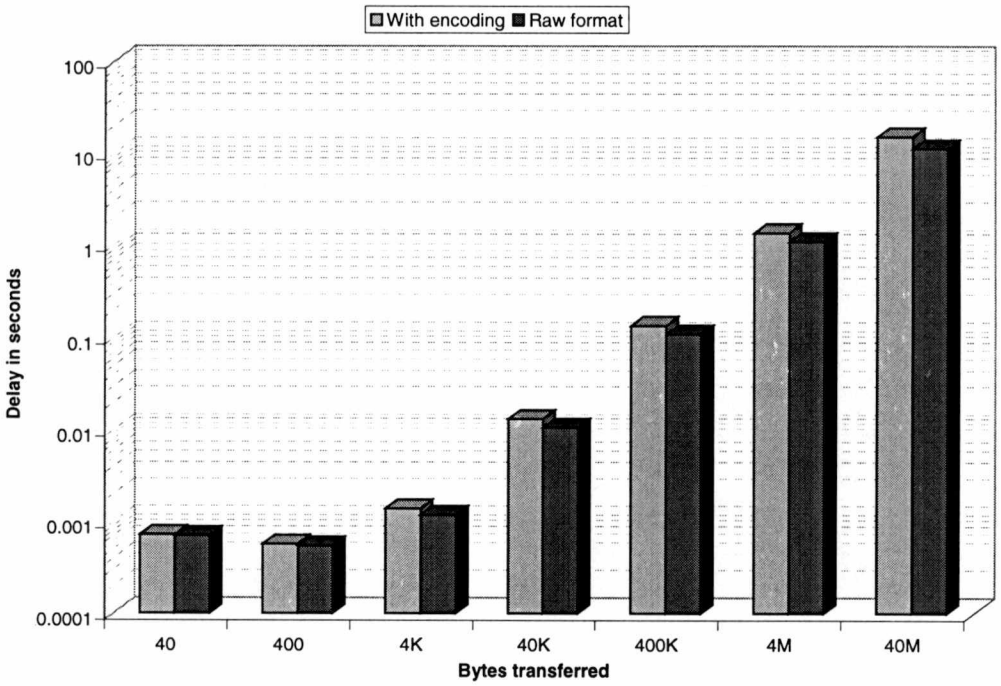


Figure 4.3: Comparison of delays with encoding and using raw format.

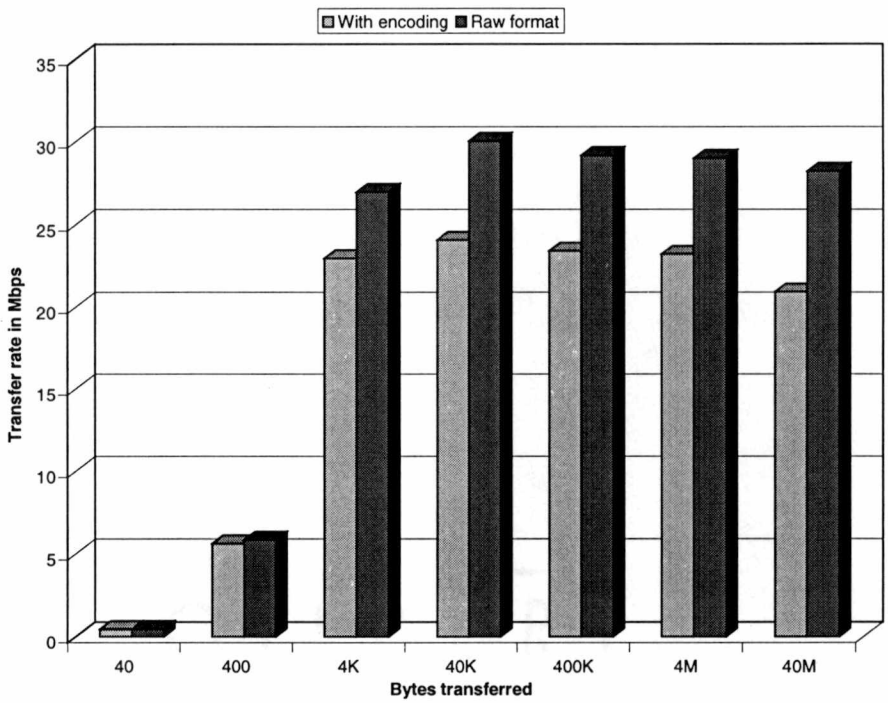


Figure 4.4: Comparison of transfer rates with encoding and using raw format.

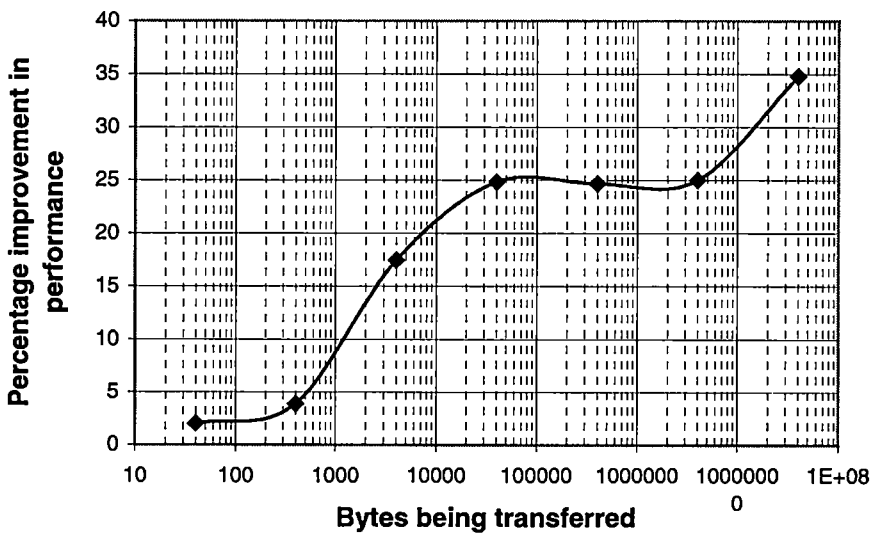


Figure 4.5: Improvement in performance using raw format over encoded format.

TIMING SCENARIO II: EFFECT OF DIRECT ROUTING

PVM can communicate between the tasks using the normal mode, by which the tasks communicate with their local daemons and the daemons communicate with each other. Another option for small cluster of machines is the direct route mode by which the tasks establish a direct TCP connection. The benchmarks were run on 8 nodes with the Fast Ethernet interface on the TORC cluster to evaluate the improvement in performance using the direct route mode.

Table 4.2 and Figure 4.6 compares the delay times obtained with normal and direct routing. The comparison of transfer rates using the two different modes of communication is shown in Figure 4.7.

From the results, when using the direct route method, the time taken to transfer a certain number of bytes is approximately 70% of the time taken to transfer them using the normal route method. In other words, direct routing makes communication 40% faster than normal routing. The improvement in performance with direct routing over normal routing is shown in Figure 4.8. The curve in Figure 4.8 shows an increase in improvement in performance first, and then it reduces and later increases again. We repeated the tests again to verify if the shape of the curve was consistent. On repeating the measurements it was observed, that the improvement in performance is always around 40% and that the shape of the curve varies. Therefore, we can conclude that the shape of the curve is not significant in this case.

Table 4.2: Comparison of delay times with direct routing and normal routing.

Bytes Transferred	**Delay with direct route (s)**	**Delay with normal route (s)**	**Ratio (direct/normal)**
40	0.00052961	0.00071105	0.744826
400	0.000374743	0.00054914	0.682418
4K	0.00087466	0.00118727	0.736701
40K	0.00838824	0.01065752	0.787072
400K	0.081123936	0.10972358	0.739348
4M	0.800975242	1.10355937	0.725811
40M	7.749094297	11.3410066	0.683281

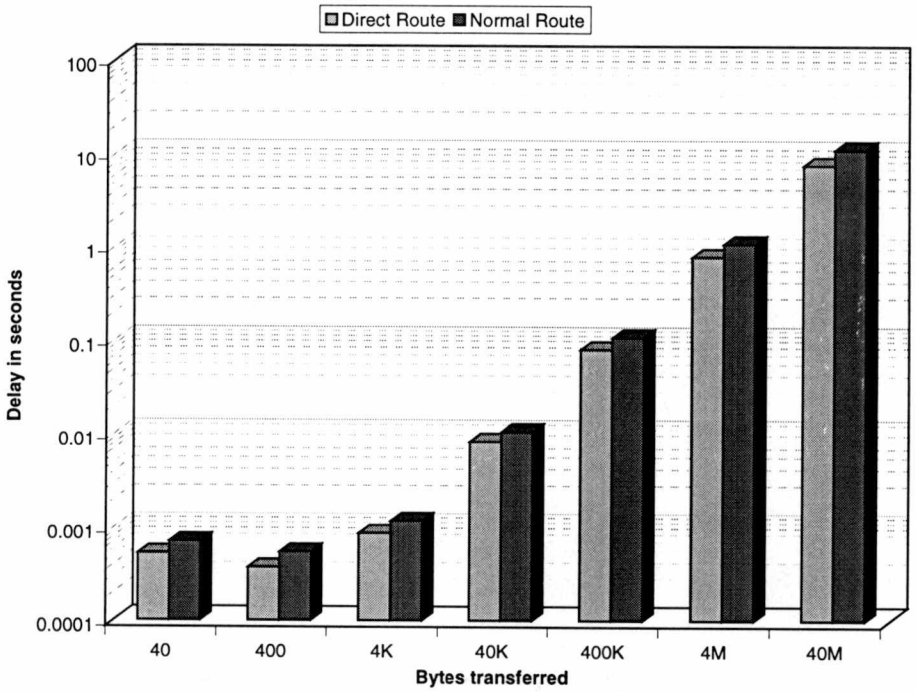


Figure 4.6: Comparison of delays with direct routing and normal routing.

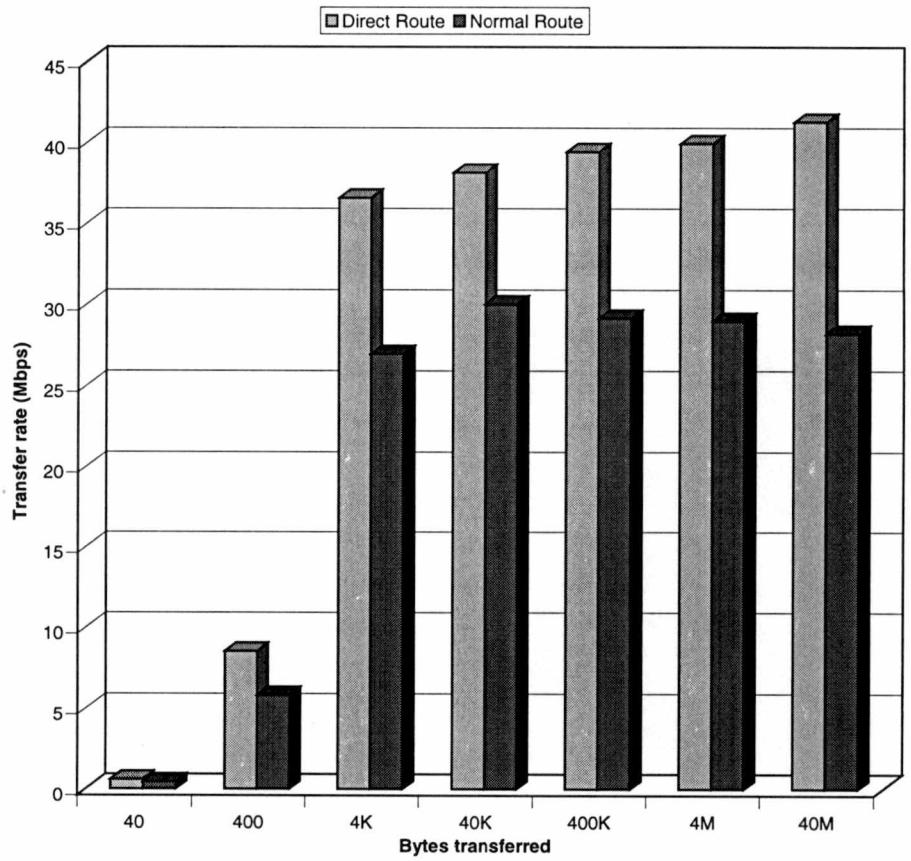


Figure 4.7: Comparison of transfer rates with direct routing and normal routing.

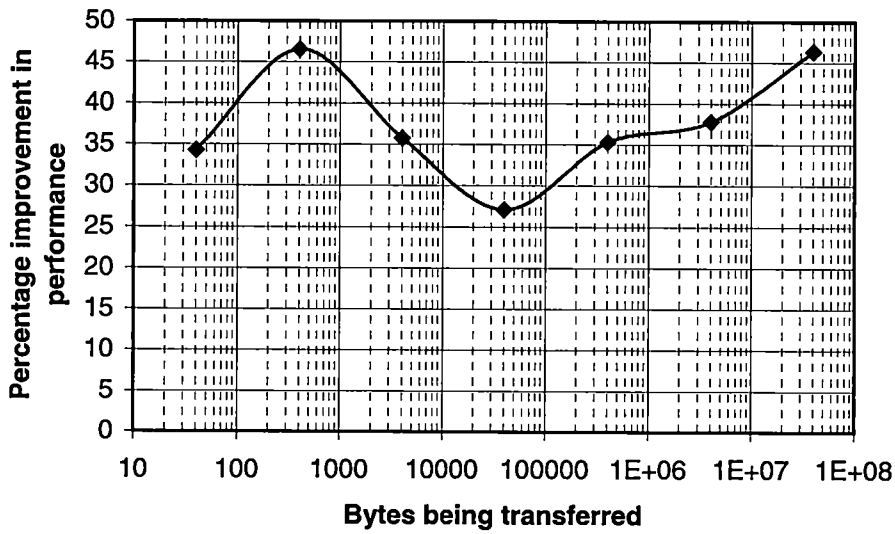


Figure 4.8: Improvement in performance with direct routing over normal routing.

The improvement in performance is obvious because with the direct route a single TCP connection is enough to communicate between the tasks whereas in the normal route, two unix domain socket connections and two UDP connections are needed. Although there is an improvement of performance, direct routing is normally not used because of the drawbacks of TCP. TCP opens a file descriptor for every connection. The operating system limits the number of opened file descriptors. Also since TCP is connection oriented in a parallel machine of N tasks, it would need to establish $N(N-1)/2$ connections, which can be a huge overhead.

TIMING SCENARIO III: LATENCY STUDIES

COMPARISON OF LATENCY ON FAST ETHERNET AND MYRINET ON TORC

In order to measure latency of Fast Ethernet and Myrinet, messages with a data size of zero bytes were sent to/from the nodes on the TORC cluster and the timing was measured. The number of nodes for the tests used varied from 2 to 8, so the effect of the number of nodes used during communication could also be evaluated.

Table 4.3 and Figure 4.9 show the comparison of latency for Fast Ethernet and Myrinet on the TORC cluster.

Table 4.3: Comparison of latency of Fast Ethernet and Myrinet on the TORC cluster.

Number of nodes	Latency for Ethernet (s)	Latency for Myrinet (s)	Ratio (Ethernet/Myrinet)
2	0.000325959	0.000319909	1.018912
4	0.000524429	0.000487037	1.076776
6	0.000578377	0.000496902	1.163966
8	0.000607737	0.00048973	1.240963

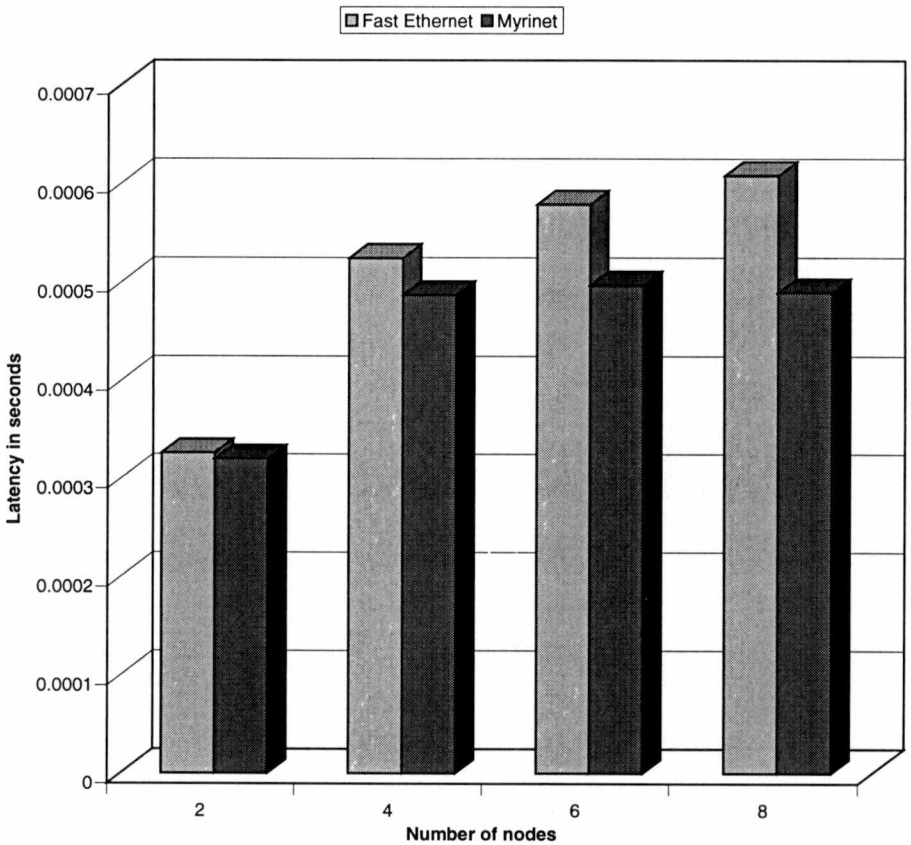


Figure 4.9: Comparison of latency for Fast Ethernet and Myrinet on TORC cluster.

The results show higher latency for Fast Ethernet as compared to Myrinet. The improvement in latency times of Myrinet over Fast Ethernet is shown in Figure 4.10. Myrinet specifications give a very low latency for Myrinet networks, but the results don't support the claimed specifications. One of the reasons could be that these values are not the measure of latency times for the network only. The values also include the PVM overhead of packing data and initiating transfers.

LATENCY MEASUREMENT ON ATM CLUSTER AT LIT

The same latency tests were carried out on the ATM cluster at LIT. The tests are repeated for 2 and 4 nodes. The latency values observed with ATM cannot be compared with the other testing scenarios for latency as the ATM cluster at LIT consists of slower processors. As seen in Table 4.4 and Figure 4.11, latency is quite high. This could be attributed to the fact that ATM delivers 53-byte packets even if data size is very small. Also, this latency time includes the PVM overheads for packing and unpacking data.

COMPARISON OF LATENCY OF FAST ETHERNET ON TORC AND LIT CLUSTER

The Fast Ethernet switch used in the LIT cluster (Netgear 508) is cheaper than the one used in the TORC cluster (BayNetworks 350 T). Also, the nodes in the TORC cluster

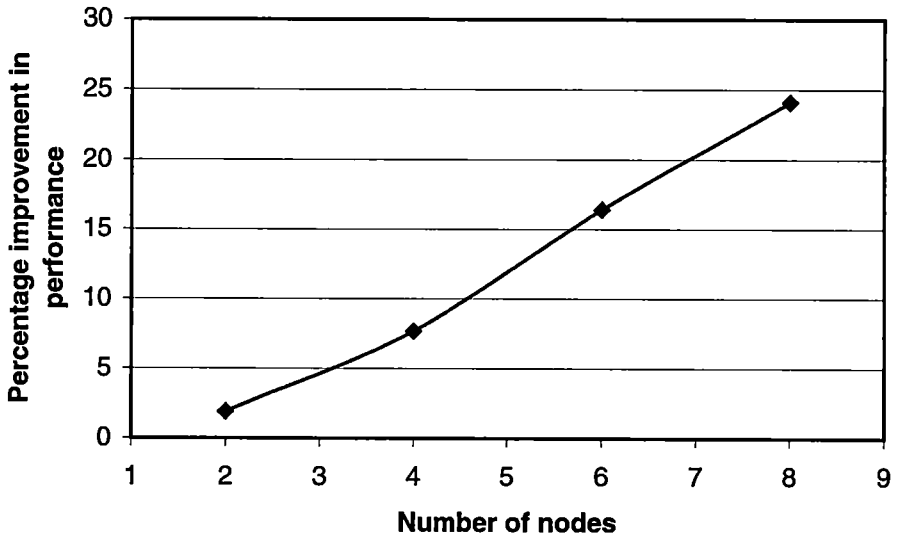


Figure 4.10: Improvement in performance of Myrinet over Fast Ethernet on TORC cluster.

Table 4.4: ATM Latency with varying number of nodes.

Number of nodes	Latency (s)
2	0.001799
4	0.002817

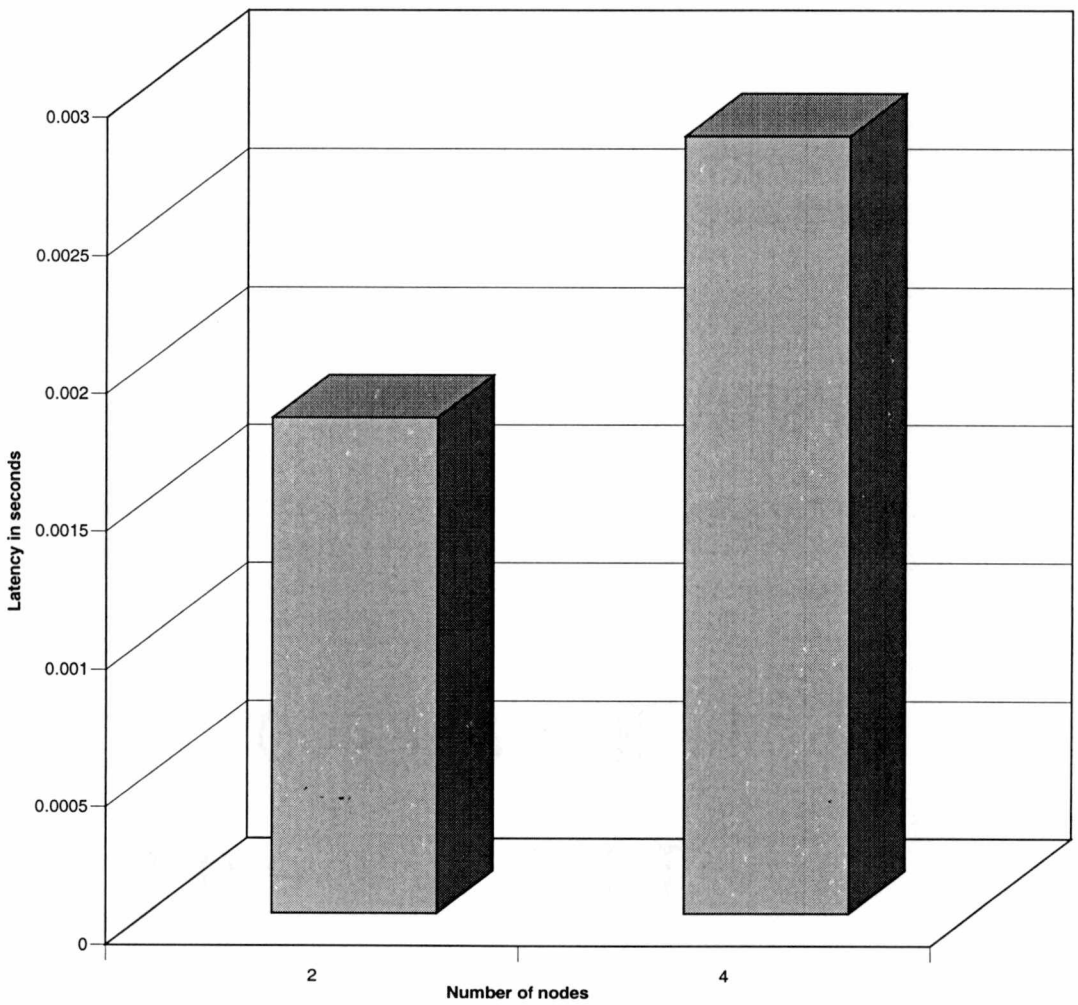


Figure 4.11: ATM latency.

are Intel Pentium III 550MHz dual processors whereas the nodes in the LIT cluster are Intel Celeron 333MHz. These results evaluate how the performance is affected on using a cheaper switch and slower processors. The latency tests were carried out on 2, 4 and 6 nodes of the LIT cluster and compared with the results obtained with the TORC cluster as shown in Table 4.5 and Figure 4.12.

The results show that LIT cluster has a latency of approximately 2.5 times that of the TORC cluster. The increase in latency with the LIT cluster when compared with the TORC cluster is shown in Figure 4.13. Each node of the TORC cluster costs approximately 3-4 times a node of the LIT cluster. The difference in cost gets more significant with increasing number of nodes. But our decision for using a cheaper switch cannot be based only on latency measurements. We need to see the maximum transfer rates and then decide if a cheaper switch would be a better alternative.

TIMING SCENARIO IV: EFFECT OF MESSAGE SIZE ON TRANSFER RATES

COMPARISON OF EFFECT OF MESSAGE SIZE ON TRANSFER RATES FOR FAST ETHERNET AND MYRINET ON TORC CLUSTER

Benchmark tests were repeated on 2 (Figure 4.14), 4 (Figure 4.15), 6 (Figure 4.16) and 8 (Figure 4.17) nodes respectively. Messages of sizes varying from 40 bytes to

Table 4.5: Comparison of latency of Fast Ethernet on LIT and TORC clusters.

Number of nodes	Latency for Ethernet on LIT (s)	Latency for Ethernet on TORC (s)	Ratio (LIT/TORC)
2	0.000919239	0.000325959	2.820111412
4	0.001065943	0.000524429	2.032578487
6	0.001492774	0.000578377	2.580969692

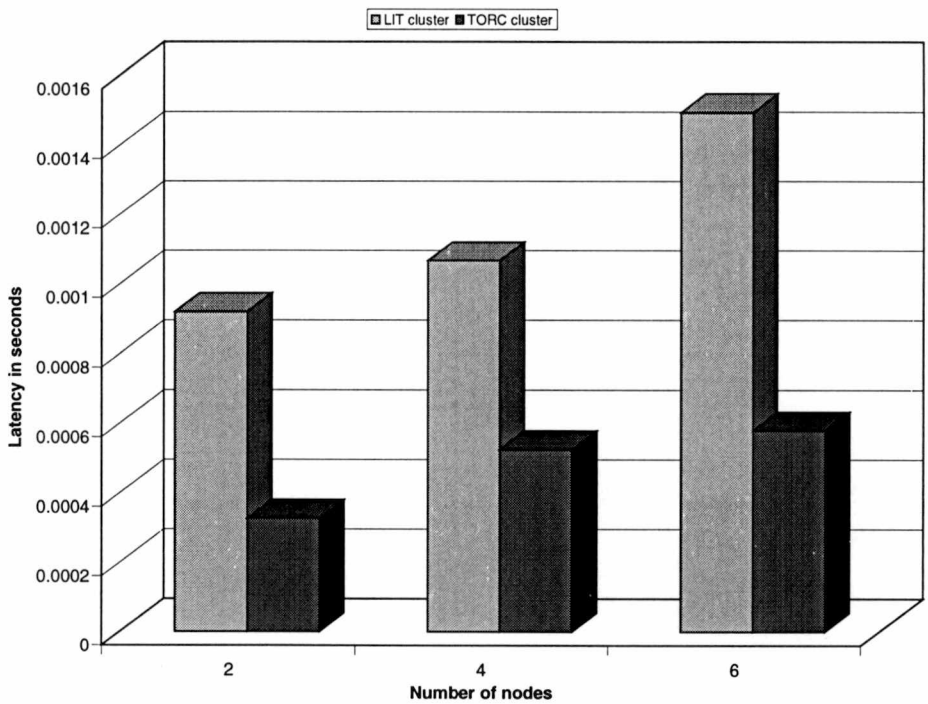


Figure 4.12: Comparison of latency of Fast Ethernet on LIT and TORC cluster.

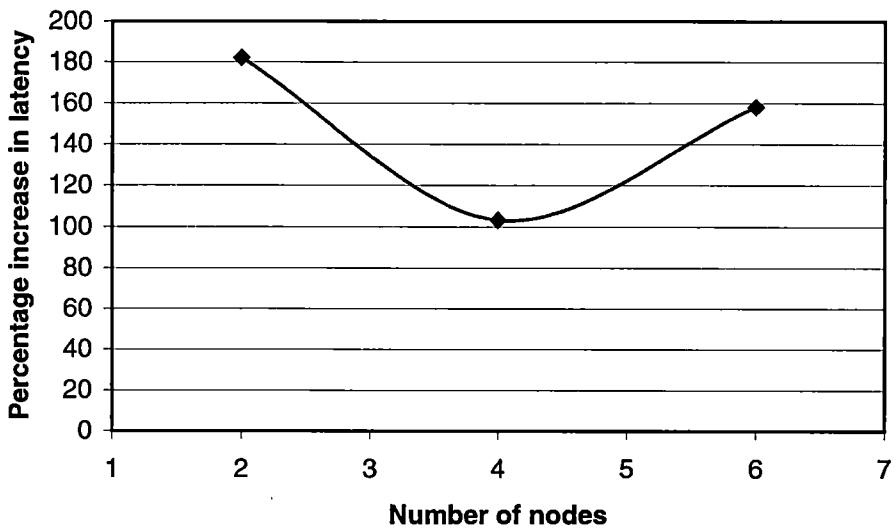


Figure 4.13: Percentage increase in latency for Fast Ethernet on LIT cluster over TORC cluster.

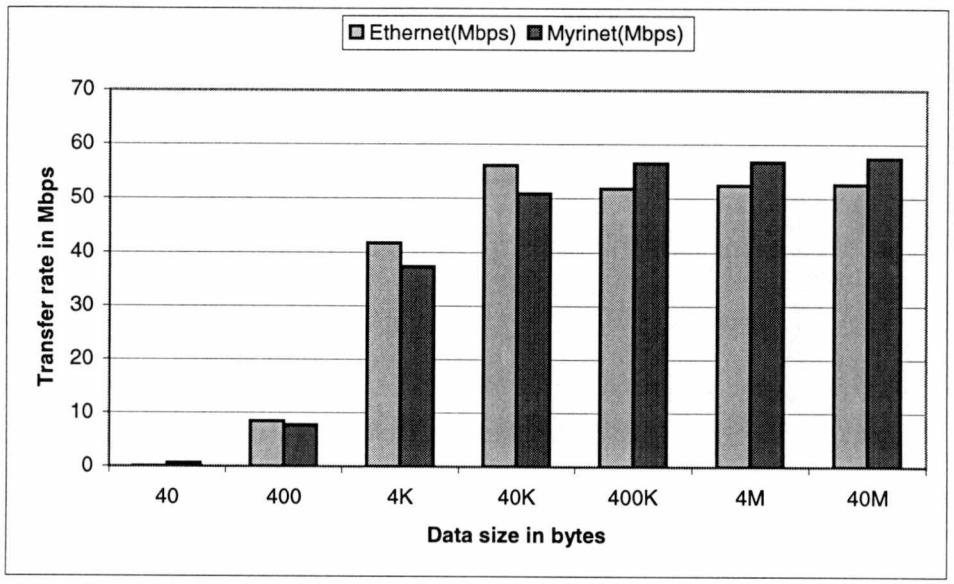


Figure 4.14: Transfer rates for Fast Ethernet and Myrinet with 2 nodes.

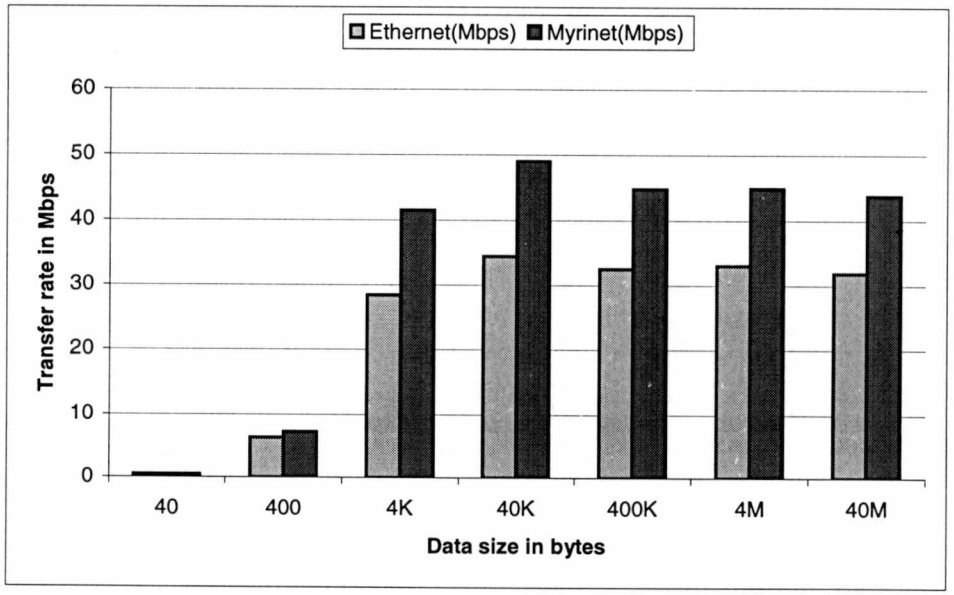


Figure 4.15: Transfer rates for Fast Ethernet and Myrinet with 4 nodes.

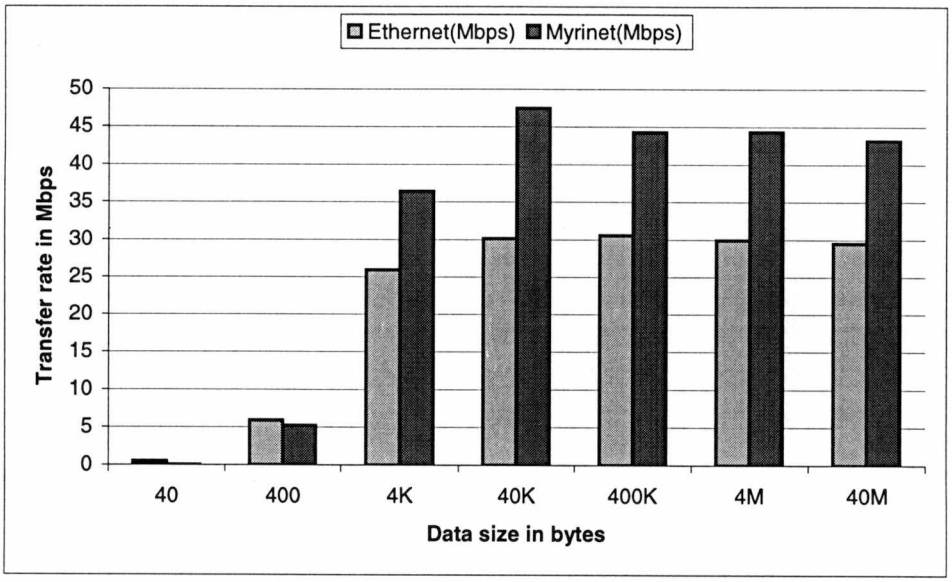


Figure 4.16: Transfer rates for Fast Ethernet and Myrinet with 6 nodes.

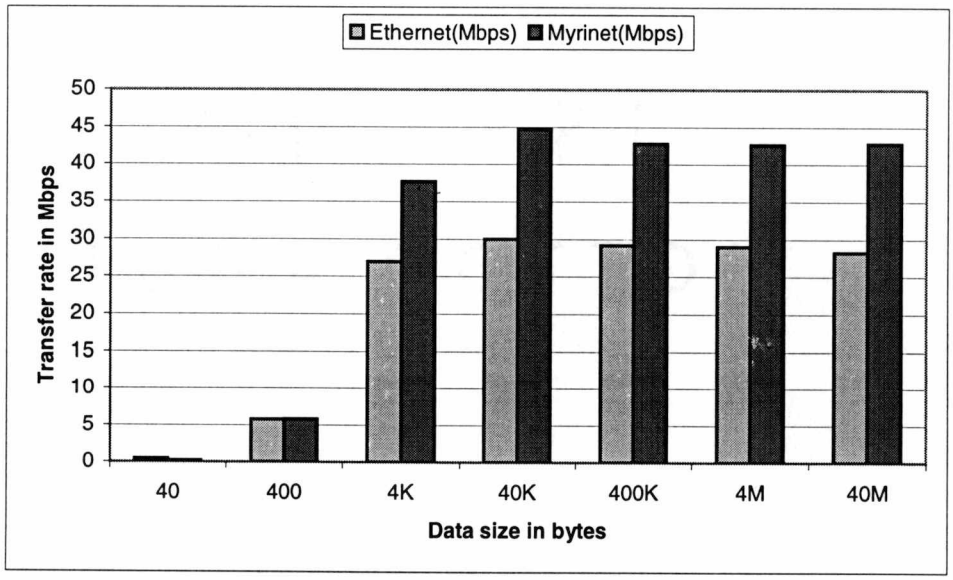


Figure 4.17: Transfer rates for Fast Ethernet and Myrinet with 8 nodes.

40Mbytes were transferred, and round trip times were measured. Using these timing results, transfer rates were calculated.

Figure 4.14, Figure 4.15, Figure 4.16 and Figure 4.17 can be summarized as in Figure 4.18. The graph in Figure 4.18 is a plot of ratio of transfer rates for Myrinet to Fast Ethernet versus bytes being transferred.

The results show that Myrinet performs better than Fast Ethernet but the performance is on the average only 30-50% better. Performance is affected by both network constraints and host processors, memory and bus clock rates. The first, however, is probably a realistic prediction of actual performance for communications-intensive applications.

EFFECT OF MESSAGE SIZE ON TRANSFER RATES FOR ATM ON LIT CLUSTER

Messages of varying sizes from 40 bytes to 40 Mbytes were sent to and from the nodes and the round trip times measured. Using these timing results, the transfer rates for ATM were evaluated and plotted as shown in Figure 4.19 and Figure 4.20.

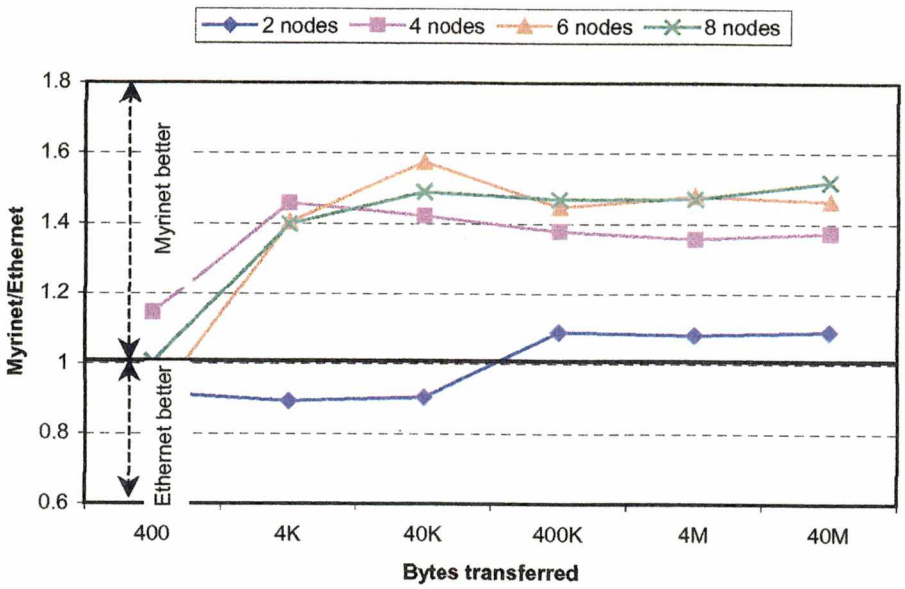


Figure 4.18: Plot of ratio of transfer rates for Myrinet to Fast Ethernet versus bytes transferred.

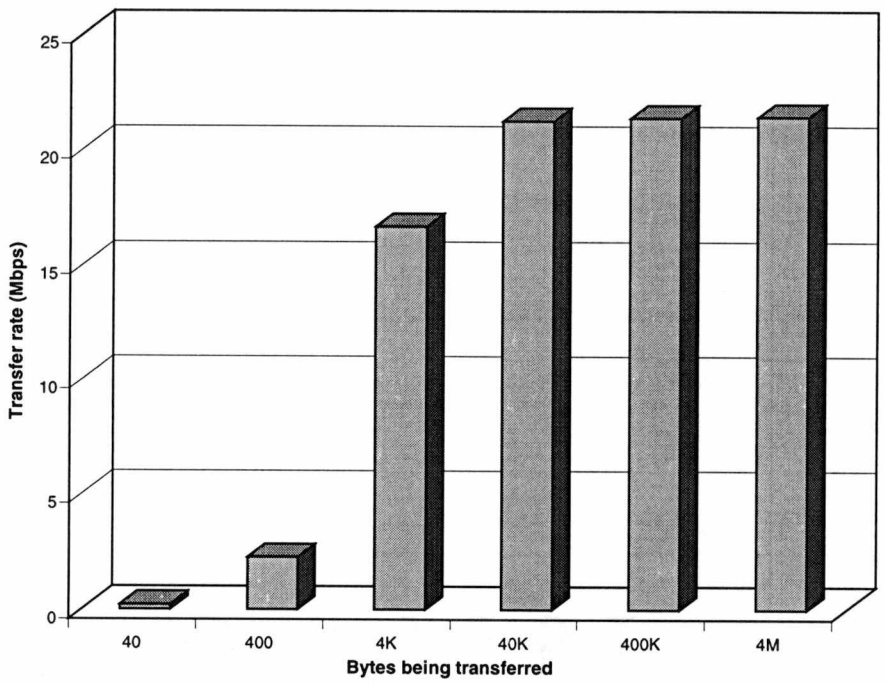


Figure 4.19: Transfer rate for ATM with 2 nodes on the LIT cluster.

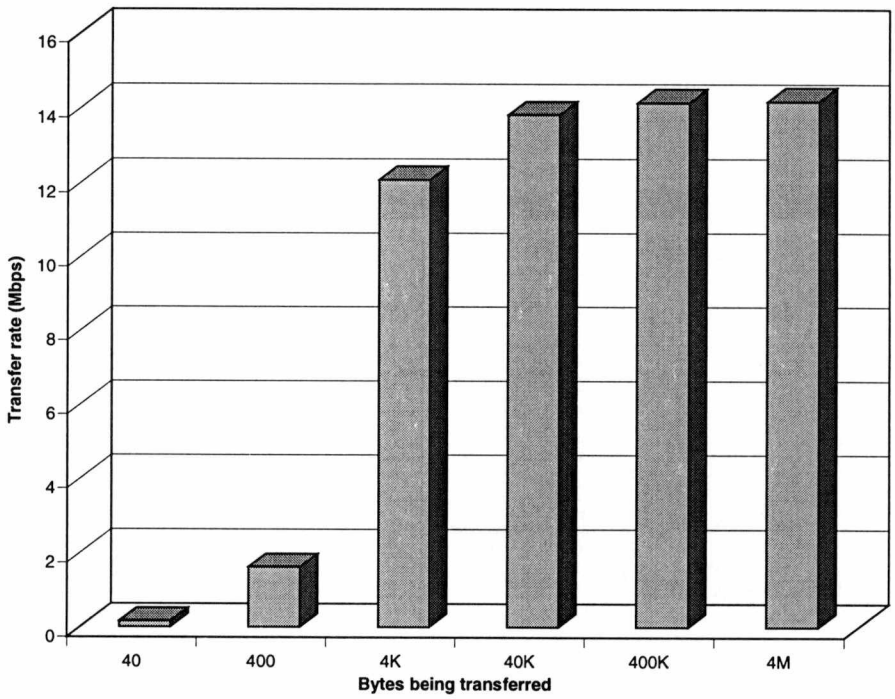


Figure 4.20: Transfer rate for ATM with 4 nodes on the LIT cluster.

COMPARISON OF EFFECT OF MESSAGE SIZE ON TRANSFER RATES FOR FAST ETHERNET ON TORC AND LIT CLUSTER

This comparison together with the latency comparison can help us to decide if a cheaper switch could be used as an alternative to the more expensive Fast Ethernet switch and how much degradation in transfer rate would it lead to. But this comparison is limited as the processors used in the TORC cluster are much faster than the ones used in the LIT cluster.

As seen from Figure 4.21, Figure 4.22 and Figure 4.23, the transfer rates observed on the TORC cluster is 1.5-3 times that obtained on the LIT cluster. Each node of the TORC cluster is 3-4 times more expensive than the LIT ones. This difference in transfer rates could be mainly due to the machine differences between the two clusters. Additional work is necessary, using the same host hardware for all tests, to establish if there is a benefit to more expensive Fast Ethernet switches.

TIMING SCENARIO V: EFFECT OF NUMBER OF NODES ON TRANSFER RATES FOR LARGE MESSAGE SIZES

The benchmark studies were repeated for 2, 4, 6 and 8 nodes on the TORC cluster with a fixed message size of 40 Mbytes.

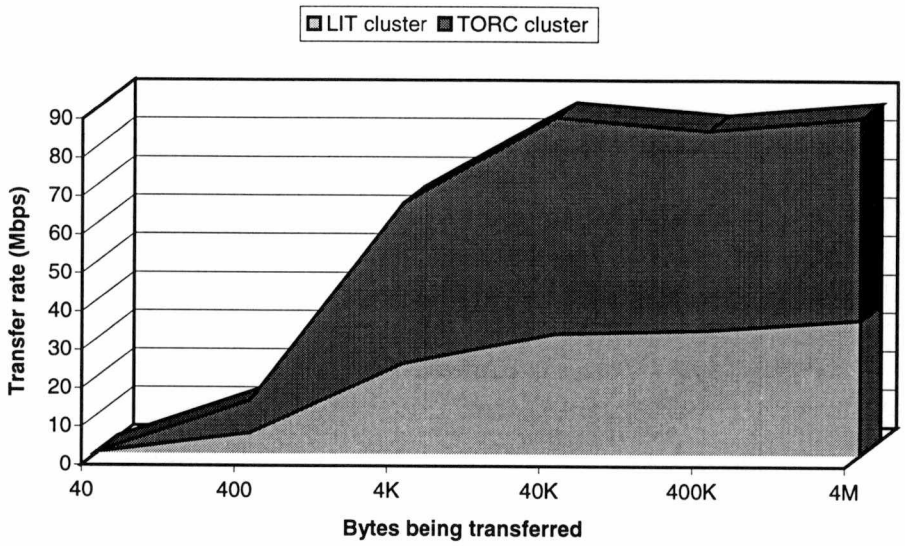


Figure 4.21: Comparison of transfer rates for Fast Ethernet on TORC and LIT cluster with 2 nodes.

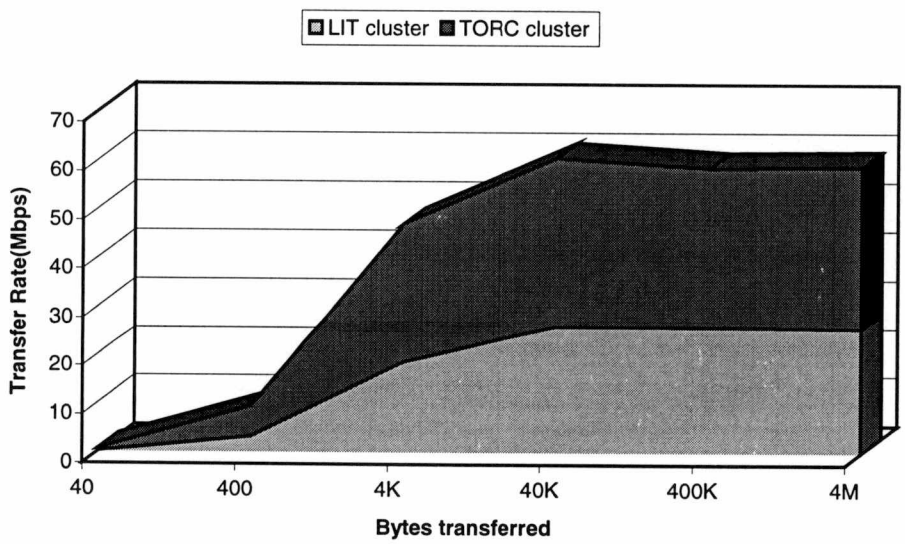


Figure 4.22: Comparison of transfer rates for Fast Ethernet on TORC and LIT cluster with 4 nodes.

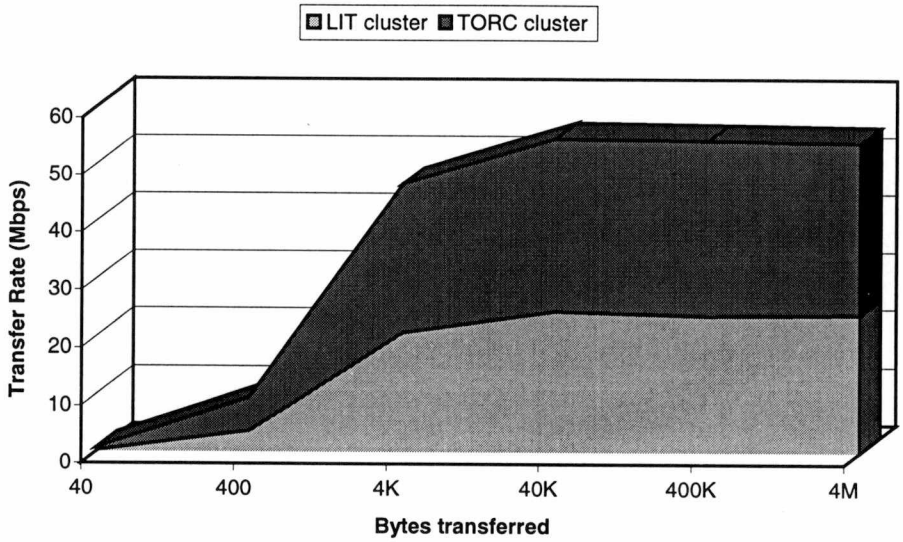


Figure 4.23: Comparison of transfer rates for Fast Ethernet on TORC and LIT cluster with 6 nodes.

As seen in Figure 4.24, transfer rate reduces as the number of nodes increase for large message size. This is possibly due to the saturation of the switch's backbone or bus. The effect could also result from contention for resources within the network switches or host processors. We have yet not attempted to isolate the cause.

COST ANALYSIS

The generic PC implementation will consist of a control rack as shown in Figure 4.25. The control rack will house either of the following high performance switches.

- ATM switch, such as the Fore Systems (now Marconi) ASX-1200
- Myrinet switch, such as Myricom's M2LM-Clos64
- Gigabit Ethernet switch, such as Extreme Networks Summit 7I

The control rack also houses the control processors, a tape backup subsystem, a video and keyboard switch, and dual uninterruptible power supplies (UPS). The control processor is specified as 1.4 GHz Pentium IVs with 2 GB of RAM and 54GB of hard disk space.

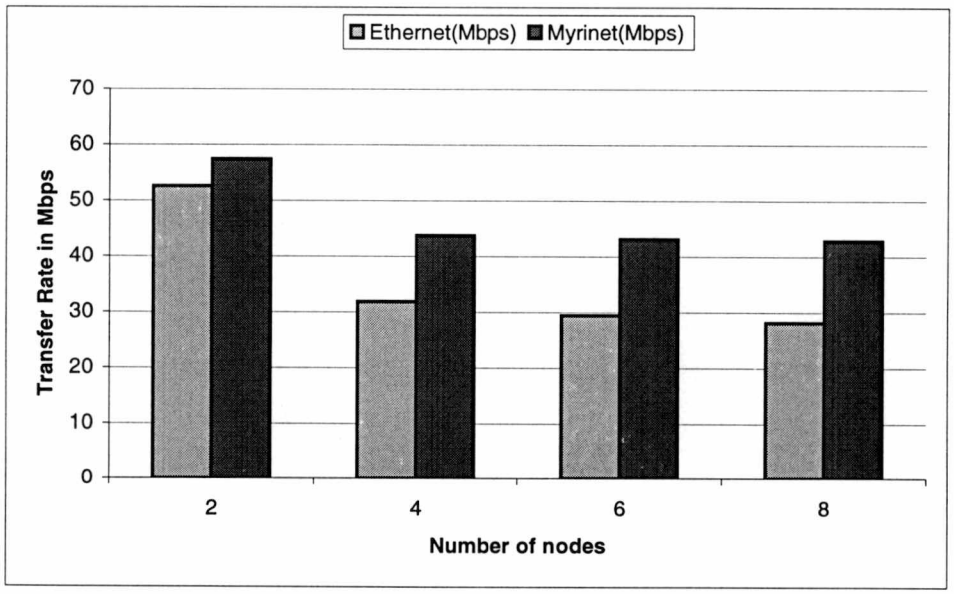


Figure 4.24: Effect on transfer rate with increasing number of nodes

(Bytes transferred 40 Mbytes)

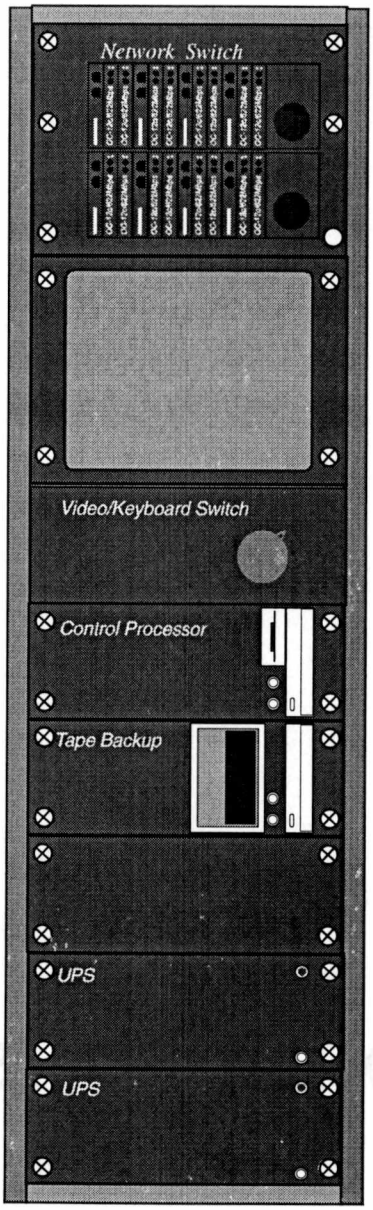


Figure 4.25: Control Rack.

This implementation can contain from one to eight PC racks, each housing eight rack mounted dual processor PCs, a midrange switch and an UPS. The alternatives for the Myrinet switch could be

- ATM switch, such as Fore Systems ASX-200
- Myrinet switch, such as Myricom's M2LM-Clos64
- Fast Ethernet switch, such as Netgear FS518T

One such rack is shown in Figure 4.26. The fully configured system with 8 racks will look like Figure 4.27.

To compare the effect of the network hardware on the cost of a parallel computer cluster, a hypothetical moderately sized cluster of 128 hosts is assumed. The cost of a Myrinet network for a 128-host parallel machine is roughly $1,500 per host. A sample configuration consists of 2 M2LM-Clos64 switches providing 64 LAN + 64 SAN connections each ($40,000 each), 128 Myrinet network interface cards (NICs) at $995 per card, 32 5' SAN cables at $140, and 128 30' LAN cables at $160, totaling $192,320 and supporting 128 hosts. The costs are current from the Myrinet website [25]. The hardware for Fast Ethernet is substantially cheaper if unmanaged switches are used in each rack of host computers. For example, 16 Netgear FS518T switches can be used, each providing 16 ports of 10/100 (Fast Ethernet) and 2 10/100/1000 (Gigabit Ethernet) up-link ports, costing about $1300 each, with two Netgear FA310 (10/100) NICs installed per host at $25 each, an Extreme Networks Summit 7I switch with 28 1000base-T Gigabit Ethernet ports at roughly $15,000, and 272 Cat-5/Cat-6 networking cables at $15 each, totaling $46,280. The cost figures are extracted from

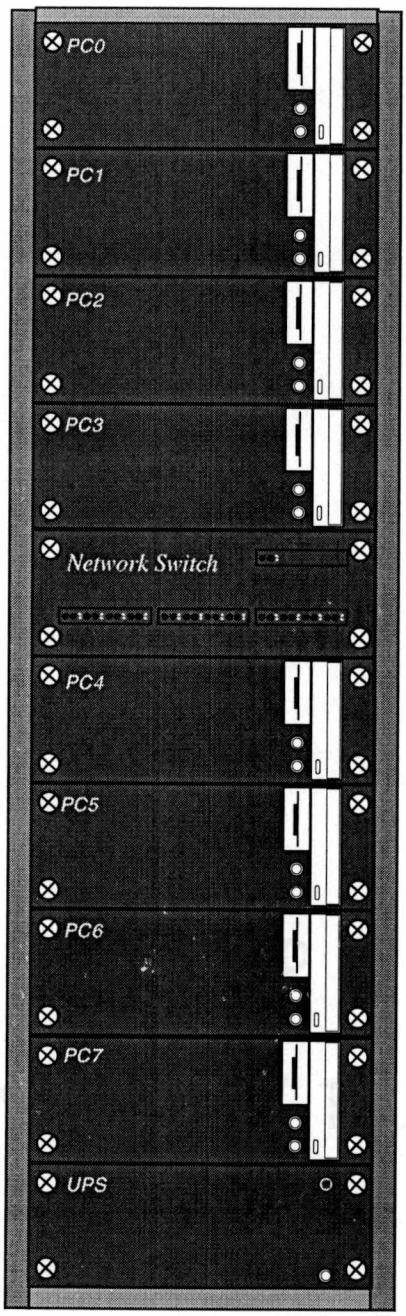


Figure 4.26: PC Rack.

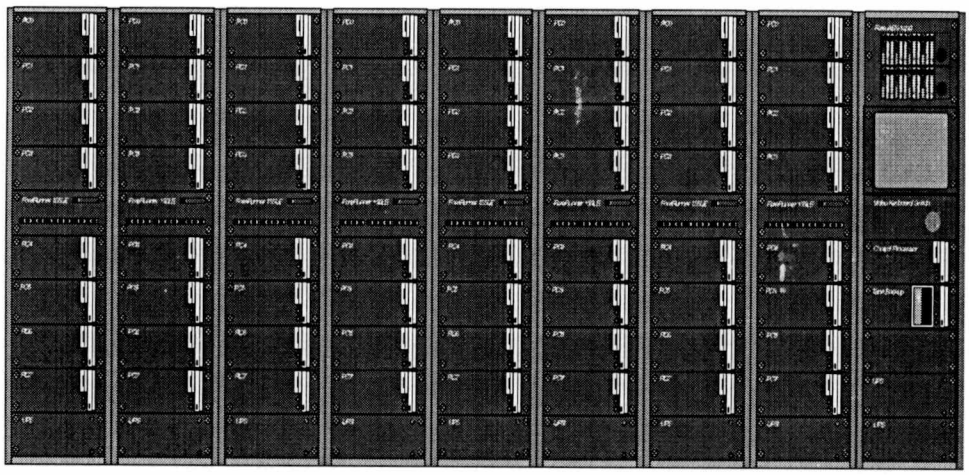


Figure 4.27: Fully Configured System.

Price Watch [37], except from the Extreme Networks switch, which is from discussions with a sales representative.

This is about $360 per host or over four times cheaper than a Myrinet solution. This cost difference is diluted over the total cost of the parallel machines, since each host costs roughly $2,500 to $4,500. At the low end of this range, the per-host cost of the parallel machine is roughly $4,000 for a Myrinet topology, versus roughly $2,860 for a Fast Ethernet/Gigabit Ethernet topology. Thus, the trade-off becomes whether a faster network or more hosts in the parallel machine would be more beneficial for a given application.

The Fast Ethernet solution that is outlined provides two full-duplex 100Mb/s interfaces to each host, with each host connected to two unmanaged network switches for automatic fail over redundancy on the switches. A total bandwidth of 400Mb/s is available to/from each host. The Netgear FS518T provides 9.6Gb/s bandwidth on its backplane and 2Gb/s bandwidth out of each rack of 8 hosts. In comparison, the Myrinet solution provides 2.48Gb/s (1.28Gb/s full duplex) bandwidth to each host, and 10.24Gb/s out of each rack of 8 hosts (due to the 20.48Gb/s maximum throughput of the Myrinet Xbar16 crossbar switch). On the surface, there is substantially more bandwidth at all levels of the Myrinet network except the main switch. (The Extreme Networks switch has a non-blocking bandwidth of 64Gb/s.) Based upon the bandwidth to each host, Myrinet offers a 6.2x improvement in bandwidth for 4x the cost. This decreases to a 2.6x improvement in bandwidth out of each rack (10.24Gb/s

/ 2Gb/s). At the level of the main switch, performance is difficult to compare due to the difference in networking topologies. For Myrinet, all network traffic passes through the two M2LM-Clos64 units, which have an aggregate bandwidth of about 160Gb/s. For the Fast Ethernet / Gigabit Ethernet solution, only a portion of the traffic passes through the Extreme Networks switch, and the limitations of the switches in each rack can impose a maximum of 32Gb/s load, which is well within the switch's maximum bandwidth of 64Gb/s.

The results that have been presented using our benchmark indicate that the achievable performance in a Linux cluster using Pentium III processors favors the use of Fast Ethernet and Gigabit Ethernet over either Myrinet or ATM (which has costs comparable to or exceeding those of Myrinet). A Myrinet solution can be expected to provide between 1.2-1.5x improved performance based upon the benchmarking software discussed in Chapter 3. These measurements are for a Fast Ethernet topology with only one connection to each host, rather than the two connections discussed earlier, so even less improvement than is indicated by the test results can be expected. We believe that for CODIS applications the equipment funds would be better spent in acquiring additional or faster computer nodes in the parallel machine rather than further improving network performance. In addition, the simplicity of cabling that is possible with a Fast Ethernet or ATM application is lost in a Myrinet network, as is partially indicated by the relative costs of the cables in the previous discussion. Therefore, our recommendation is that a combination of Fast Ethernet and Gigabit Ethernet be utilized for the parallel machine. This approach also provides a natural

migration path to Gigabit Ethernet. Gigabit Ethernet is currently in a very rapid growth phase of its development and deployment. If the current trends continue, a situation similar to what has occurred over the past couple of years with Fast Ethernet is likely, in which the selection of available equipment increases and prices fall dramatically.

SUMMARY OF RESULTS

The different testing scenarios and the conclusion derived from the benchmark studies discussed in the previous sections can be summarized as in Table 4.6.

Table 4.6: Summary of the Different Testing Scenarios

1	Comparison of transfer rates using raw format and encoding	Transfer rate is higher when raw format is used. The percentage improvement increases as the message size increases.
2	Comparison of transfer rates with normal and direct routing.	Direct mode shows an improvement of approximately 40% over normal mode.
3	Effect of number of nodes on transfer rate for large messages.	Transfer rate reduces with increasing number of nodes due to saturation of switch's backbone.
4	Comparison of latency for Fast Ethernet and Myrinet on identical machines.	Latency of Fast Ethernet is higher than Myrinet. The difference becomes prominent with increasing number of nodes.
5	Performance of ATM switch on LIT Cluster.	ATM shows high latency. Transfer rate obtained is low. The performance is bad due to LANE.

CHAPTER 5

RECOMMENDATIONS FOR FUTURE RESEARCH

In this thesis, the following network interconnects – Switched Fast Ethernet, ATM and Myrinet were evaluated. The cost and performance comparisons make Fast Ethernet the best option for our parallel machine.

A networking method that has not been studied in this research is the Gigabit Ethernet. Gigabit Ethernet technology had been gaining acceptance, and this could lead to price reductions quite similar to the trend seen for Fast Ethernet. This means that Gigabit Ethernet could be an option for network interconnects in our application. Tests should be conducted to evaluate the performance of a Gigabit Ethernet switch interconnecting the hosts using the benchmark software described in this research. It is expected, however, that while Gigabit Ethernet will probably be a good choice for the network uplink ports from each equipment rack containing eight computer hosts, it will prove to be too expensive at this time to justify the benefit of Gigabit Ethernet connections to each host.

Another interesting point would be to compare the performance of the cheaper Fast Ethernet switches, such as the Netgear 508 and 516 (used in LIT cluster) and more

expensive switches, such as the BayNetworks 350 T (used in TORC cluster) with identical hosts. The tests that were performed on these switches in this research showed an improvement of approximately 1.5-3 when using the expensive switch; however much or all of this improvement could be due to differences in host processors. Testing both switches with identical hosts would help isolate the cause for the improvement in performance. If the improvement is due to performance limitation of the hosts, then a cheaper switch could be used in the final implementation of the parallel machine.

BIBLIOGRAPHY

BIBLIOGRAPHY

[1]. Lewis, M., and Cline, E., "PVM Communication Performance in a Switched FDDI Heterogeneous Distributed Computing Environment," *Proceedings of the IEEE Workshop on Advances in Parallel and Distributed Systems*, October 1993, pages 13-19.

[2]. Michael, W., Cronin, J., and Pieper, K., *FDDI: an introduction to fiber distributed data interface,* Digital Press, June 1992.

[3]. Blade, E., et al., "An experimental assessment of network impact on cluster based computing," *Proceedings of the Gigabit Network Workshop*, Toronto, Canada, June 1994.

[4]. Benner, A., *Fibre Channel: Gigabit Communications and I/O for Computer Networks*, McGraw-Hill, November 1995.

[5]. IBM, *IBM Allnode Interconnect System*: Overview. Part # 57G2977, April 1994.

[6]. Graphs comparing bandwidths using MPICH on LACE, http://www.lerc.nasa.gov/WWW/ACCL/PARALLEL/TIMING/june96/graph.mpich.big.ps

[7]. Graphs comparing bandwidths using PVM on LACE, http://www.lerc.nasa.gov/WWW/ACCL/PARALLEL/TIMING/june96/graph.pvm.big.ps

[8]. Kim, J., and Lilja, D., "Performance-Based Path Determination for Interprocessor Communication in Distributed Computing Systems," *IEEE Transactions on Parallel and Distributed Systems*, Vol. 10, No. 3, March 1999.

[9].Bell, J. "The HiPPI protocol", 1995.

[10]. Steed, M., "Performance Prediction of PVM Programs", *Proceedings of the 10th International Parallel Processing Symposium*, April 1996.

[11]. Geist, A., et al., "PVM and MPI: a comparison of features", *Calculateurs Paralleles*, Vol. 8, No. 2, 1996.

[12]. Jackson, W., "Network access speeds bureau's forensic data to law enforcement officers," *Government Computer News*, Vol. 18, No. 34, October 11, 1999. http://www.gcn.com/vol18_no34/com/788-1.html

[13]. Buyya, R., *High Performance Cluster Computing: Architectures and Systems*, Vol. 1, Prentice Hall, 1999.

[14]. Stevens, R., *TCP/IP Illustrated – The Protocols*, Vol. 1, Addison-Wesley, 1994.

[15]. Chivers, I., and Sleightholme, J., *Introducing Fortran 90*. Springer-Verlag, Sept. 1995.

[16]. High Performance Fortran Forum, *High Performance Fortran Language Specification*, January 31, 1997.

[17]. FBI Press Room, "DNA Index System", Press Release, October 13, 1998. Website http://www.fbi.gov/pressrm/pressrel/pressrel98/dna.htm

[18]. Calle, D., "Supercomputers", Website http://ei.cs.vt.edu/~history/SUPERCOM.Calle.HTML

[19]. *Top 500 Supercomputer List* (November 2, 2000) Website, http://www.netlib.org/benchmark/top500/top500.list.html

[20]. *ASCI White* Website, http://www.llnl.gov/asci/news/white_news.html

[21]. *Blue Gene Project* Website, http://www.research.ibm.com/bluegene/

[22]. *Beowulf Project* Website http://www.beowulf.org

[23]. Dongarra, J., et al., "The Marketplace for High-Performance Computers", *Parallel Computing*, 1999.

[24]. Beguelin, A., et al., *PVM: Parallel Virtual Machine – A User's Guide and Tutorial for Networked Parallel Computing*, MIT Press, Massachusetts Institute of Technology, 1994.

[25]. *Myrinet* Website, http://www.myri.com

[26]. *Tennessee Oak Ridge Cluster* Website, http://icl.cs.utk.edu/projects/torc/

[27]. Walker, D., et al., *MPI: The Complete Reference*, MIT Press, Massachusetts Institute of Technology, 1996.

[28]. Riley, S., and Breyer, R., *Switched, Fast, and Gigabit Ethernet,* Macmillan Technical Publishing, January 1999.

[29]. Siu, K., and Jain, R., "A Brief Overview of ATM: Protocol Layers, LAN Emulation, and Traffic Management," *Computer Communications Review (ACM SIGCOMM)*, vol 25, no 2, April 1995, pages 6-28.

[30]. Birdwell, J., et al., "A Hierarchical Database Design and Search Method for CODIS," *Promega Conference*, Orlando, Florida, September 1999.

[31]. Wang, T., et al., "CODIS Matching Algorithm for Large Databases," *Fifth Annual CODIS User's Group Meeting*, November 18-19, 1999.

[32]. Scientific Supercomputer Subcommittee of the Committee on Communications and Information Policy, *Supercomputing - An Informal Glossary of Terms.*

[33]. Fosdick, L., and Jessup, E., "An Overview of Scientific Computing", September 28, 1995.

[34]. Boden, N., et. al., "Myrinet--A Gigabit-per-Second Local-Area Network," *IEEE Micro*, Vol. 15, No. 1, February 1995.

[35]. Fischer, M., and Dongarra, J., "Another Architecture: PVM on Windows 95/NT," October 4, 1996.

[36]. Dunigan, T., "Performance of ATM/OC-12 on the Intel Paragon," May 1996.

[37]. *Price Watch* Website, http://www.pricewatch.com

[38]. Sun Microsystems Inc, "XDR: External Data Representation Standard," June 1987, RFC 1014.

APPENDIX

APPENDIX

The bubble charts show the average round trip time between any two nodes for a particular message size. In the timing studies, we obtain the round trip times for 20 ping-pongs between any two nodes. The area of the bubble is proportional to the average delay time between the two nodes. The bubbles on the leading diagonal are always smaller compared to the other bubbles in the chart because, the bubbles on this diagonal represent the time required to send a packet to itself. If the nodes are identical, it is expected that all bubbles be of comparable size. Each bubble chart can be summarized as a single point by calculating the average of each of the bubble sizes. Bubbles, which are significantly larger or smaller, are not included when calculating the average to minimize the effect of variations in the network traffic.

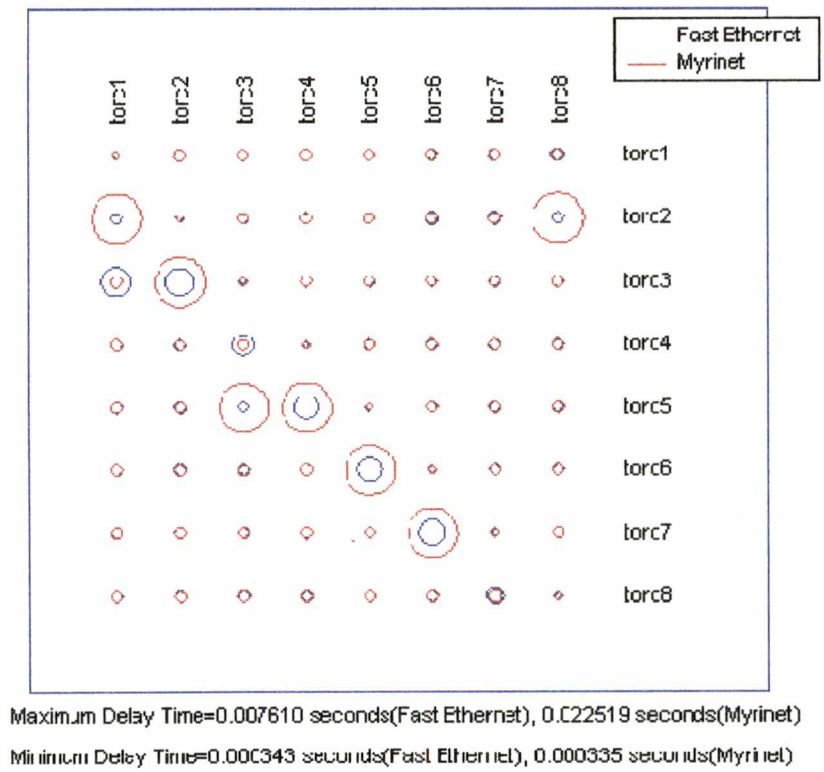


Figure A.1: Bubble chart for message size of 40 bytes and on 8 nodes of the TORC cluster.

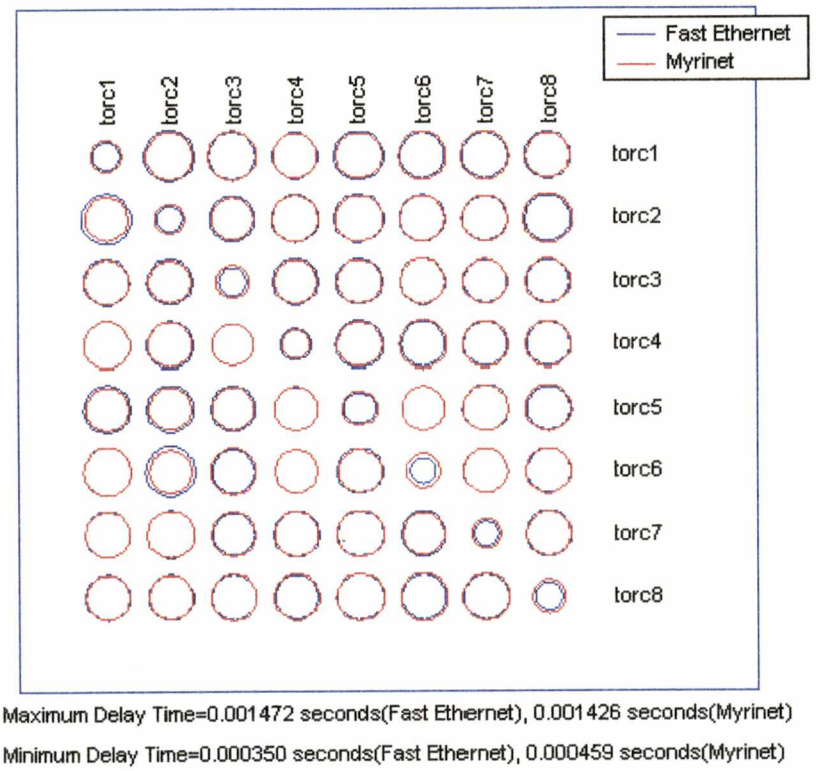


Figure A.2: Bubble chart for message size of 400 bytes and on 8 nodes of the TORC cluster.

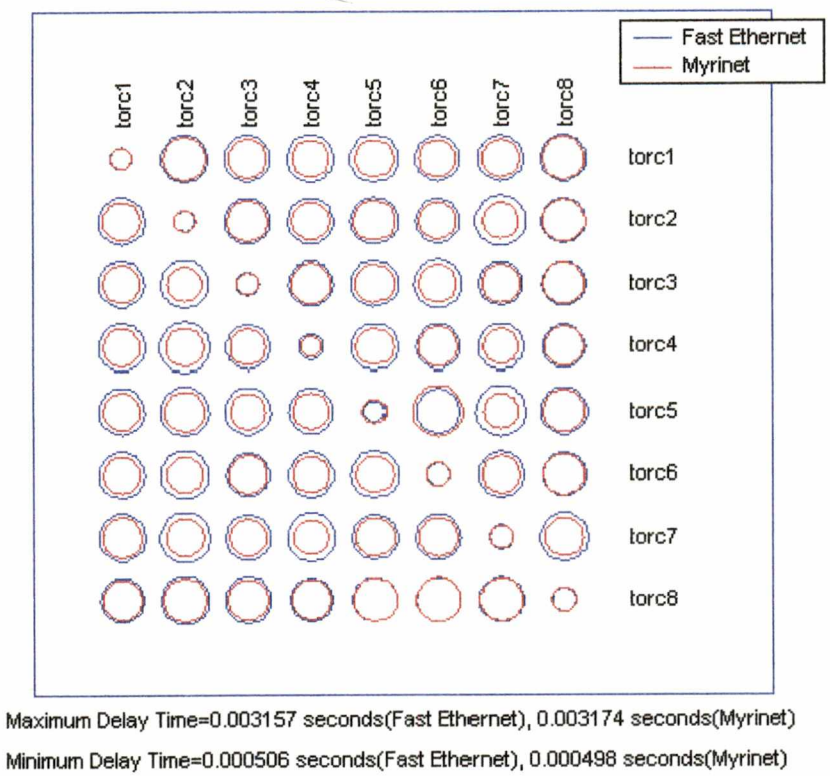


Figure A.3: Bubble chart for message size of 4 Kbytes and on 8 nodes of the TORC cluster.

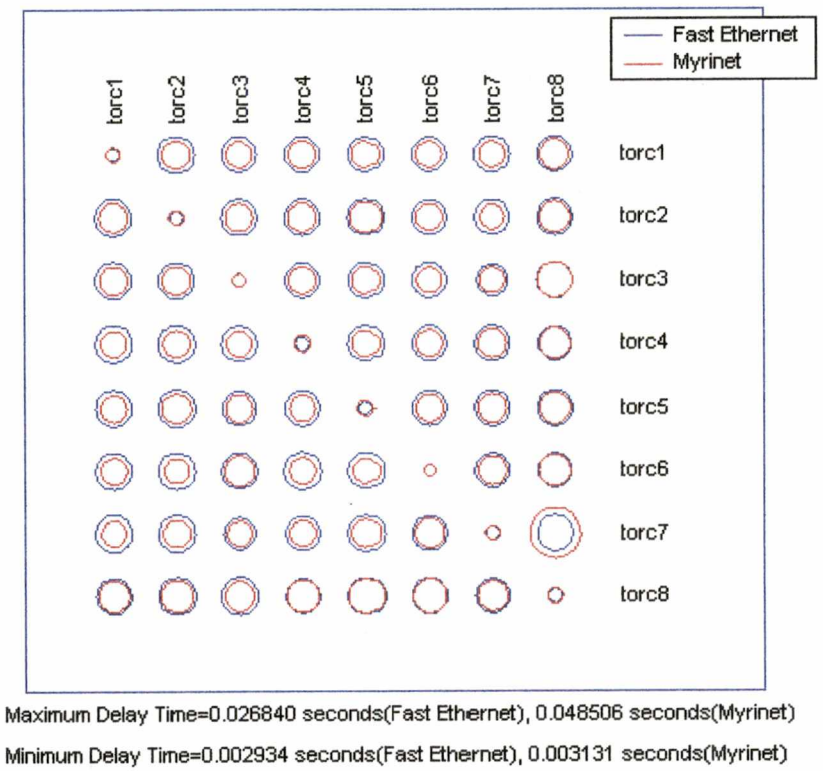


Figure A.4: Bubble chart for message size of 40 Kbytes and on 8 nodes of the TORC cluster.

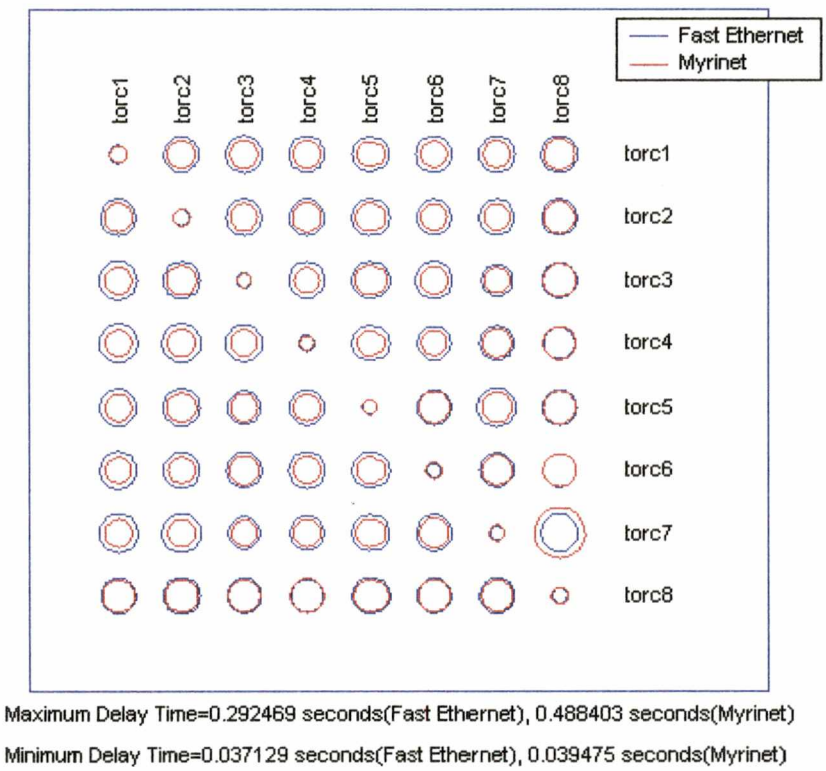


Figure A.5: Bubble chart for message size of 400 Kbytes and on 8 nodes of the TORC cluster.

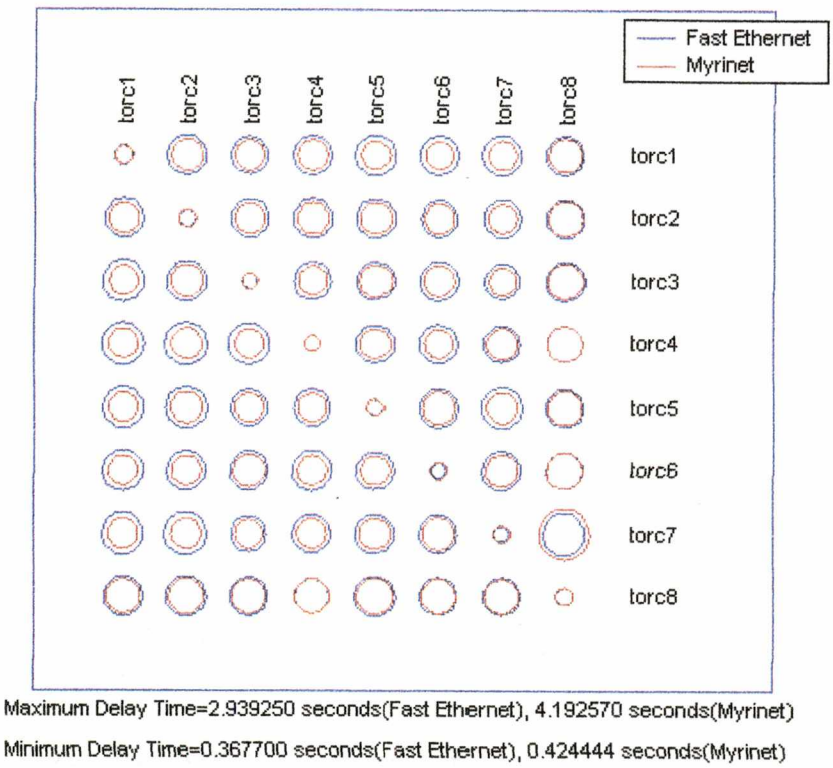


Figure A.6: Bubble chart for message size of 4 Mbytes and on 8 nodes of the TORC cluster.

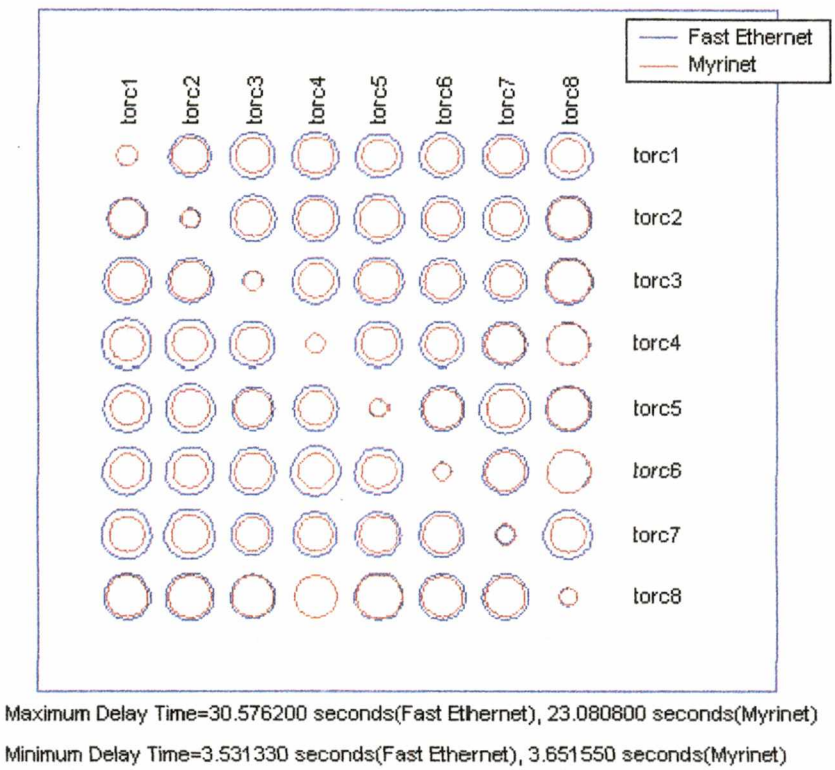


Figure A.7: Bubble chart for message size of 40 Mbytes and on 8 nodes of the TORC cluster.

VITA

Priyanka Dasgupta was born on April 24, 1978 in Jabalpur, India. She received her Bachelor of Engineering (B.E.) degree in Electronics and Telecommunications from Government College of Engineering, Pune, India in May 1999. In August 1999, she joined the Master's program in Electrical Engineering at University of Tennessee, Knoxville. She began working for Laboratory of Information Technologies in January 2000 on the project '*Rapid DNA Identification for Forensic Application*' funded by the Federal Bureau of Investigation (FBI). Her Master's thesis was on *Performance Evaluation of Fast Ethernet, ATM and Myrinet under PVM*. She graduated with her Master's Degree in December 2000.