Ouachita Baptist University

Scholarly Commons @ Ouachita

Honors Theses

Carl Goodson Honors Program

1999

Issues in Automated Distribution of Processes Over the Networks

Alexey Morozov

Ouachita Baptist University

Follow this and additional works at: https://scholarlycommons.obu.edu/honors_theses

Part of the OS and Networks Commons

Recommended Citation

Morozov, Alexey, "Issues in Automated Distribution of Processes Over the Networks" (1999). *Honors Theses*. 122.
https://scholarlycommons.obu.edu/honors_theses/122

This Thesis is brought to you for free and open access by the Carl Goodson Honors Program at Scholarly Commons @ Ouachita. It has been accepted for inclusion in Honors Theses by an authorized administrator of Scholarly Commons @ Ouachita. For more information, please contact mortensona@obu.edu.

SENIOR THESIS APPROVAL

This Honors thesis entitled

"Issues in Automated Distribution of Processes over Networks"

written by

Alexey Morozov

and submitted in partial fulfillment of the requirements for completion of the Carl Goodson Honors Program meets the criteria for acceptance and has been approved by the undersigned readers.

thesis director (Dr. Terry Sergeant):

second reader (Dr. Stephen Hennagin):

third reader (Dr. David Griffith):

honors program director (Dr. Douglas Reed):

April 29, 1999

Issues in automated distribution of processes over the networks

Alexey Morozov

morozov@ava.obu.edu

April 29, 1999

RILEY-HICKINGBOTHAM LIBRARY
OUACHITA BAPTIST UNIVERSITY

Contents

1 Introduction 2

1.1 Distributed computing . . . 3

1.2 Parallel computing . . . 3

1.2.1 SIMD . . . 4

1.2.2 MIMD . . . 4

1.2.3 MISD . . . 5

2 Reasons for distributing applications across the networks 5

3 Existing models and tools 7

3.1 Models . . . 7

3.1.1 Client-server . . . 7

3.1.2 Networks of workstations (NOWs) and clusters . . . 7

3.1.3 Metacomputing . . . 8

3.2 Communication protocols . . . 8

3.2.1 OSI – Open Systems Interconnection . . . 9

3.2.2 IP – Internet Protocol . . . 9

3.2.3 TCP – Transmission Control Protocol . . . 10

3.2.4 UDP – User Datagram Protocol . . . 11

3.2.5 XTP – Xpress Transport Protocol . . . 11

3.2.6 ATM – Asynchronous Transfer Mode . . . 11

3.3 Tools . . . 12

3.3.1 PVM – Parallel Virtual Machine . . . 12

3.3.2 MPI – Message Passing Interface 13
3.3.3 CORBA – Common Object Request Broker Architecture 14

4 Related issues **15**
4.1 Security 15
4.2 Scheduling and load balancing 18

5 Developing and experimenting with distributed systems **19**
5.1 Study of the MPI point-to-point communication methods 20
5.2 Some results in distributing processes over "slow" networks 23
5.3 Developing a general-purpose distributed system 24
5.3.1 System requirements 25
5.3.2 System design 26

6 Conclusions **30**

1 Introduction

The main goal of this paper is to survey the issues an application developer would have to resolve in producing a system that would be able to spread its computational load across several computers connected by a network. Before this can be done, a brief introduction to distributed and parallel computing is necessary.

1.1 Distributed computing

Distributed computing is generally defined as a type of computing in which different components and objects comprising an application can be located on different computers connected to a network [6]. Thus, for example, a word processing application may consist of an editor component residing on one computer, spell-checker component residing on a second computer and a thesaurus component residing on a third computer. It is possible that these different computers may be running different operating systems. Since different components may be written by different developers or groups of developers, all the parties must must agree upon a certain standard for inter-object communication. One of the most common standards in the industry is CORBA (section 3.3.3), which will be discussed later.

1.2 Parallel computing

Parallel computing is the use of more than one CPU to execute a program. [6] There are some differences in programming such systems as compared to "regular" or "sequential" programming. In sequential programming there is usually one process (program) working on a problem. When a problem is solved in parallel, there are multiple processes working on it. As in any combined effort, the participants need to communicate with each other. Often, some sort of management is also required. Programming parallel systems also provides a challenge because most programmers have been trained to think sequentially. For example, if there are two tables of numbers that need to be multiplied, the most common approach that comes to mind is iterating through every element of the table. However, in the parallel paradigm, iteration is not needed, since one would have a host of CPUs, each adding one element, thus, in the combined effort, adding all of them at once. Overall, particular problems and solutions depend on the type of the parallel algorithm, problem at hand, and the hardware available.

There is a widely recognized classification of parallel machines, known as Flynn's taxonomy, that dis-

tinguishes between three types of architectures: SIMD, MIMD, and MISD. It is not difficult to generalize the concept and apply it to the algorithms and, since this paper is dealing more with software methods than hardware, the following sections will do so.

1.2.1 SIMD

SIMD stands for "Single Instruction, Multiple Data". It is one of the most common uses for parallel computing, mainly due to the ease with with which such algorithm can be distributed across multiple processors. In this model all of the processors are running the same set of instructions, but they are supplied with different data. For example, if the task at hand is to calculate GPA for a group of students, the same algorithm is applied to all of them. Thus each processor is using the same program, but is accessing data for different students. Clearly, this way the task will be performed much faster. However, not all tasks can be parallelized as easily as the one mentioned above.

1.2.2 MIMD

MIMD stands for "Multiple Instruction, Multiple Data". This paradigm can be easily compared to the manufacturing of an automobile. Before the final product can be assembled, different parts (wheels, engine, interior) must be made. For each part, a different production algorithm and different set of source materials is used. Although this method may seem more flexible than SIMD, there are certain drawbacks. One such drawback is that the whole process needs to be orchestrated by some sort of manager. For example, the engine cannot be assembled until pistons have been made. In addition to that, some tasks are completed faster than others, thus creating the possibility that some processors are left with nothing to do.

1.2.3 MISD

MISD stands for "Multiple Instruction, Single Data". There is no commonly known implementation of this paradigm, but it is not hard to imagine one. For example, if one has a signal recording with a lot of interference and wishes to apply different noise reduction techniques to it, he or she can use multiple processors running different sets of instructions, yet operating on the same data.

2 Reasons for distributing applications across the networks

There are many computer applications that simply would take an unacceptably long time to produce results if done sequentially and due to that have to be done in parallel. Examples of such applications are image processing, code breaking, calculations with large numbers, complex mathematical models. Traditionally, parallel processing was the domain of large and extremely expensive supercomputers. However, as faster and better networks developed, an alternative approach was realized: to combine multiple PC's connected by a network into a virtual supercomputer. In addition to increased network speed, the workstations themselves are becoming more and more powerful (for example, according to research at UC Berkeley [1] a top end 1994 workstation is roughly one-third the performance of a Cray C90 processor) As the Beowulf project has proved [14] [15], a cluster of such machines can provide a much cheaper alternative to supercomputers. Although Beowulf clusters have shown their massive computing power, this power depends largely on the fact that all of the nodes in the cluster are of the same architecture, run the same type of software (operating system, applications, etc.) and are connected by a high-speed local network.

Another reason for developing distributed applications is the fact that processors are getting much faster, but disks are improving mostly in capacity, not performance [1]. If this trend continues the

increases in processor performance will yield little improvement for the end-user, since most of the time will be spent waiting for the I/O. With the availability of fast networks a different alternative develops: a potential existence of a huge memory pool on the network which can be accessed faster than local disk.

A brief survey of the usage of the workstations and PC's would reveal that most of the time their CPUs are idling or using only a small percentage of their cycles for computations of any sort. Clearly, a great abundance of resources could be harvested if those cycles could be utilized. Two main problems opposing this idea are the difficulty of (or, rather, delays in) communication between the hosts and the heterogeneity of the computers and their systems. Some groups are already working on finding solutions to these problems. Programmers of UC Berkeley's SETI@home are working on a project that would combine the power of the PCs connected to the Internet and use their idle time (that is usually spent on screen-savers) to look for patterns in the data collected by SETIs Arecibo radio telescope [18]. The investigators of SETI@home hope to get at least fifty thousand PCs involved and, according to them, that would rival all current SETI projects. This project should be very successful in utilizing the idle resources available throughout the Internet, but it is rather inflexible in several ways. It is strictly a client-server application with clients designed for one particular number-crunching task, where the server simply provides the data upon the request of the client and has no influence whatsoever on the scheduling of the client tasks.

Yet another (and on a much larger scale) approach was taken by a group from the University of Virginia. The goal of the Legion Project [4] [5] is to develop the architecture that would combine multiple computers of various types across the Internet into a virtual world-wide computer that is relatively transparent[1] to the user.

It is evident that the resources of the networked computers are there to be harvested by the developers.

[1] appears as a single machine, rather than a combination of many

To take advantage of those resources, programmers will have to know and understand the issues related to the environment and have the appropriate tools to manipulate it.

3 Existing models and tools

3.1 Models

There are several models for distributing processes across several hosts connected via a network or networks. These paradigms are not mutually exclusive, but rather intersecting and some are the evolved versions of others.

3.1.1 Client-server

The client-server model is one of the oldest models for distributed processing. In this model an application consists of a group of cooperating processes, called servers, that offer services to the users, called clients [17]. The client and server machines normally all run the same operating system. A machine may run a single process, or it may run many clients, many servers, or a mixture of the two. The communication between clients and servers is usually based on a simple, connectionless request/reply protocol. The client sends a request message to the server, asking for some service (e.g. find an entry in the database). The server does the work and returns the data requested or an error code indicating why the work could not be performed.

3.1.2 Networks of workstations (NOWs) and clusters

Networks of workstations model is a relatively general idea of utilizing resources of the workstations on a network. The workstations may run the same or different operating systems and can be of the same

or different architectures. Also, the functions of the nodes of these networks may be diverse with some systems being used for storage, others as computational drones, and so on.

Clusters are usually a more specific case of NOW. In the case of clusters, the nodes are confined to a local-area network segment consisting entirely of the cluster nodes. Most of the nodes have no direct connection with the outside world, all necessary interface provided by a limited number of "entry-point" stations. All the nodes are of the same architecture and run copies of the same operating system. Most often, some sort of shared disk space exists, and sometimes a network shared memory model is used.

3.1.3 Metacomputing

A metasystem is a collection of geographically separated resources (people, computers, instruments, databases) connected by a high speed network. A metasystem is distinguished from a simple collection of computers by a software layer, often called middleware, which transforms a collection of independent resources into a single, coherent, virtual machine. This machine should be as simple to use as the machine on the user's desktop, and should allow easy collaboration between colleagues located anywhere in the world [7].

3.2 Communication protocols

It is clearly understood that if processes are to be distributed across the network or networks, there must exist some mechanism that would allow data to be passed from one computer connected to the network to another. A range of communication protocols is already in place, with new protocols[2] continuing to emerge.

[2] An agreed-upon format for transmitting data between two devices [6]

3.2.1 OSI – Open Systems Interconnection

Developed by ISO (International Standards Organization), OSI is a standard for worldwide communications that defines a networking framework for implementing protocols in seven layers. Control is passed from one layer to the next, starting at the application layer in one station, proceeding to the bottom layer, over the channel[3] to the next station and back up the hierarchy [6]. Although at one time most vendors agreed to support OSI, it was too loosely defined, and as a result currently very few of the communication protocols are fully OSI-compliant. However, majority of the protocols have most of the OSI functionality, often combining several OSI layers into one.

OSI consists of seven layers: application layer, presentation layer, session layer, transport layer, network layer, data link layer and a physical layer. The physical layer manages putting data onto the network media (like copper or optical cable) and taking the data off. The data link layer is responsible for physically passing data from one node to another. The network layer routes data from one node to another. The transport layer is responsible for end-to-end integrity of data transmission. The session layer is responsible for establishing and maintaining communication channels (in practice, this layer is often combined with the transport layer). The presentation layer manages data representation conversions. Finally, the topmost layer, application layer, is responsible for program-to-program communication.

3.2.2 IP – Internet Protocol

The Internet Protocol is designed for use in interconnected systems of packet-switched computer communication networks. The Internet Protocol provides for transmitting blocks of data called datagrams (or packets) from sources to destinations, where sources and destinations are hosts identified by fixed length addresses. The Internet protocol also provides for fragmentation and reassembly of long datagrams, if

[3] a communication path between two computers or devices

necessary, for transmission through "small packet" networks [11].

The Internet Protocol is specifically limited in scope to provide the functions necessary to deliver a datagram from a source to a destination over an interconnected system of networks. There are no mechanisms to augment end-to-end data reliability, flow control or sequencing [11]. IP is called on by host-to-host protocols in an Internet environment. It calls on local network protocols to carry the Internet datagram to the next gateway or destination host.

3.2.3 TCP – Transmission Control Protocol

Most often IP is used in conjunction with TCP, creating the well-known TCP/IP suite.

TCP is a connection-oriented, end-to-end reliable protocol designed to fit into a layered hierarchy of protocols which support multi-network applications. TCP provides for reliable inter-process communication between pairs of processes in host computers attached to distinct but interconnected computer communication networks. Very few assumptions are made as to the reliability of the communication protocols below the TCP layer. TCP assumes it can obtain a simple, potentially unreliable datagram service from the lower level protocols. In principle, the TCP should be able to operate above a wide spectrum of communication systems ranging from hard-wired connections to packet-switched or circuit-switched networks. [12].

TCP/IP suite usually includes a wide range of additional protocols: User Datagram Protocol (UDP), Simple Network Management Protocol (SNMP), Internet Control Message Protocol (ICMP), Address Resolution and Reverse Address Resolution Protocols (ARP/RARP), TELNET Protocol, File Transfer Protocol (FTP), Simple Mail Transfer Protocol (SMTP), Hypertext Transfer Protocol (HTTP), Domain Name Service (DNS), Remote Login protocol (RLOGIN), Dynamic Host Configuration Protocol (DHCP), Post Office Protocol (POP), Internet Message Access Protocol (IMAP). I will briefly describe some of

the protocols relevant to the topic.

3.2.4 UDP – User Datagram Protocol

The User Datagram Protocol is defined to make available a datagram mode of packet-switched computer communication in the environment of an interconnected set of computer networks. This protocol assumes that the Internet Protocol is used as the underlying protocol. UDP provides a procedure for application programs to send messages to other programs with a minimum of protocol mechanism. The protocol is transaction oriented, and delivery and duplicate protection are not guaranteed [10].

3.2.5 XTP – Xpress Transport Protocol

TCP/IP suite has held up rather well through the changes in the underlying networking infrastructures. However, there were several efforts to examine the services TCP/IP fails to provide and to research the possibilities of bettering transport services. One of the results was the development of the XTP protocol (although it has already been incorporated into some proprietary protocol stacks and several experimental suites, it has not been standardized by either ANSI or IEEE). In comparison to TCP, XTP boasts better flow and rate control and superior error-recovery model. XTP also allows for reliable multicast support (TCP fails to provide multicasting altogether), message priority and scheduling, quality of service negotiations, selective retransmission, and several other features not available in TCP [16].

3.2.6 ATM – Asynchronous Transfer Mode

Asynchronous Transfer Mode is emerging as the primary networking technology for next-generation, multi-media communication. ATM protocols are designed to handle time critical data (video, audio) in addition to more conventional data communications. ATM protocols are capable of providing a homogeneous network for all traffic types. The same protocols are used regardless of whether the application is

to carry telephone conversation, video or computer traffic over local area networks (LANs), metropolitan area networks (MANs), or wide area networks (WANs).

3.3 Tools

As one cannot build a car without tools, one also cannot build a computer application without them. This is even more true for distributed applications. Most of the communication protocols described in the previous sections come with some sort of function library that can be used by the developer, yet this is clearly not enough. To develop distributed applications one needs tools that treat different processes as a conglomerate, while allowing for individual control and access at the same time. These tools need to provide not only data sharing capabilities but also various managerial tasks.

3.3.1 PVM – Parallel Virtual Machine

The PVM software provides a unified framework within which parallel programs can be developed in an efficient and straightforward manner using existing hardware. PVM enables a collection of heterogeneous computer systems to be viewed as a single parallel virtual machine. PVM transparently handles all message routing, data conversion, and task scheduling across a network of incompatible computer architectures [3]. More precisely, PVM allows for the following: user-configured host pool [4], translucent access

[4] the application's computational tasks execute on a set of machines that are selected by the user for a given run of the PVM program which can also be altered by adding and deleting machines during operation

to hardware [5], process-based computation [6], explicit message-passing model [7], heterogeneity support [8], and multiprocessor support [9] [3].

PVM was developed mainly by the PVM group at the University of Tennessee and Oak Ridge National Laboratory.

3.3.2 MPI – Message Passing Interface

Message passing is a relatively simple paradigm that allows different processes to exchange messages. This paradigm is used widely on certain classes of parallel machines, especially those with distributed memory. Over the last ten years, many applications have been cast into this paradigm, each vendor implementing its own variant of message passing. To enable creation of portable applications that utilize the message passing concept, it was necessary to develop some sort of widely accepted standard. Through the collaboration of about 60 people from 40 organizations (mainly from the US and Europe) the Message Passing Interface standard [8] emerged.

The goals of the MPI forum were: to design an application programming interface[10], to allow efficient communication, to allow for implementations that can be used in a heterogeneous environment, to allow convenient C and Fortran 77 bindings for the interface, to define an interface that can be implemented

[5] application programs either may view the hardware environment as an attributeless collection of processing elements or may choose to exploit the capabilities of specific machines in the host pool by positioning certain computational tasks on the most appropriate computers

[6] the unit of parallelism is a task, an independent sequential thread of control that is capable of computation and communication, with a possibility of multiple tasks executing on a single processor

[7] collections of computational tasks, each performing a part of an application's workload cooperate by explicitly sending and receiving messages to one another, with message size limited only by the amount of available memory

[8] PVM supports heterogeneity in terms of machines, networks, applications and data representations

[9] PVM uses the native message-passing facilities on multiprocessors to take advantage of the underlying hardware

[10] API – a set of routines, protocols, and tools for building software applications [6].

on many vendors' platforms, to allow for language-independent semantics, and to allow thread-safety[11]. In the design process forum members assumed a reliable communication interface, meaning that user does not have to cope with communication failures – such failures are dealt with by the underlying communication subsystem (such as TCP). The forum also tried not to deviate too much from the existing message passing and parallel processing models (such as PVM). The resulting standard includes point-to-point communication[12], collective operations[13], process groups[14], communication contexts[15], process topologies[16], bindings for Fortran 77 and C and environmental management and inquiry.

3.3.3 CORBA – Common Object Request Broker Architecture

Developed by Object Management Group, CORBA is an architecture that enables pieces of programs, called objects, to communicate with one another regardless of what programming language they were written in or what operating system they are running on.

The main components of CORBA are object systems, Object Request Brokers (ORBs) and clients. An object system includes entities known as objects. An object is an identifiable, encapsulated entity that provides one or more services that can be requested by a client. Client is an entity that wishes to perform an operation on the object. ORB is responsible for all of the mechanisms required to find the object, to prepare the object implementation for the request, and to communicate the data making up the request. The interface the client sees is completely independent of where the object is located or what programming language it is implemented in. To make this possible, objects need to describe their

[11] thread – a part of a program that can execute independently of other parts [6].

[12] in this case, communication method in which there is one sender and one receiver

[13] operations involving more than one sender and/or receiver

[14] ordered collections of processes, each with rank [8]

[15] contexts provide the ability to have a separate safe "universe" of message passing between the two groups, such that a send in a local group is always a receive in the remote group, and vice versa

[16] define a special mapping of the ranks in a group

interfaces. These definitions of interfaces can be defined in two ways. They can be defined statically in an Interface Definition Language. This language defines the types of objects according to the operations that may be performed on them and the parameters to those operations. Alternatively, or in addition, interfaces can be added to an Interface Repository service; this service represents the components of an interface as objects, permitting runtime access to these components. [9]

4 Related issues

4.1 Security

In the modern age of the Internet, computer security has become one of the most important issues in the industry. This issue becomes even more important if one wishes to develop applications that would function across the networks. Moreover, in the paradigms such as metacomputing, the resources that are combined into the metacomputer often belong to different people and organizations. The owner of each host would want to be ensured of the security of his system. In addition, different hosts may need different levels of security.

When dealing with network security, two main issues arise. One is the problem of authenticating users (or clients) that should have access to a given computer or service. The other issue deals with protecting the data when it is traveling on the network.

There are several mechanism available for authenticating users. The simplest one is shipped with any Unix or Windows NT operating system. It is based on a user database that keeps a user's name and his/her password. When the user is trying to log in, the system will ask for a password and compare it with the one stored in the database. It is also conceivable that any given user database may become compromised. To avoid damages that could be inflicted, the passwords in the database are usually stored

RILEY-HICKINGBOTHAM LIBRARY
OUACHITA BAPTIST UNIVERSITY

in an encrypted form. Moreover, the encryption scheme used is one-way, meaning that once the password is encrypted, even the system does not know how to decrypt it. In this case, the password supplied by the user is also encrypted before the comparison is made.

The issue becomes a little more complicated when the same user (or set of users) wishes to have access to more than one system. The simplest solution would be to duplicate the user database and distribute it to multiple computers. However, this method is very cumbersome and inflexible in many ways. For example, a user currently logged into one system changes his/her password. That means that all copies of this database on other systems need to be updated by some mechanism. The problem becomes even more complicated when multiple systems do not have exactly the same user set. Finally, this solution becomes unacceptable if one wishes to make the network transparent to the user.

Network Information Service (NIS) may provide a solution to this issue. NIS provides generic database access facilities that can be used to distribute information to multiple hosts on the network. It is based on RPC[17] and consists of a server, a client-side library and several administrative tools. NIS keeps database information in so-called maps containing key-value pairs. Maps are stored on a central host running the NIS server, from which clients may retrieve the information through various RPC calls. [13]

Another authentication mechanism is Kerberos, originally designed at MIT. Kerberos is a trusted third-party authentication service. It is trusted in the sense that each of its clients believes Kerberos' judgment of the identity of each of its other clients to be accurate. Kerberos consists of a Key Distribution Center (KDC) that runs on a physically secure computer somewhere on the network and a library of subroutines that are used by distributed applications which want to authenticate their users. Kerberos generates private keys which are given to the authenticated clients. Using these keys, Kerberos' clients can convince each other of their identity [2].

[17] remote procedure call – a protocol that allows a program on one computer to execute a program on a server computer[6]

It may also be necessary to encrypt the data traveling across the network. This can be done in various ways. One possible scenario is to utilize a proprietary protocol, but this method defeats the purpose of combining the resources of systems possibly belonging to different organizations. Another solution is to use some publicly known encryption scheme. This scheme must be constructed in such a way that the knowledge of how to implement the scheme would not allow one to decrypt a message without proper authorization. A rather elegant method of encryption, called public-key (or assymetric) encryption was put forth in 1976 to solve the above-mentioned problem. The method is based on the following paradigm: the encryption algorithm is known to everyone and each party involved creates a pair of keys. Either key can be used with the known encryption algorithm. One of the keys is the encryption key and it is made public. The other key, known only to the owner, is the decryption key. Thus, for each person A, there is a an encryption function $encrypt_A()$ that is known to everyone, and a decryption function $decrypt_A()$ that is known only to A.

Each pair of the keys is constructed so that it has a special property. For any message m,

$$decrypt_A(encrypt_A(m)) = m$$

Thus, applying the private decryption key to a message that has already been encrypted with the public encryption key recovers the original message. It is this particular property that makes public/private pairs work. It is also important that knowledge of the public encryption key does not make it easy to find the corresponding decryption key.

Additional functionality of public-key encryption is the ability to create digital signatures. A digital signature allows the recipient of a message to verify the identity of the sender and to ascertain that the contents of the message have not been altered. To enable this, the public/private pair needs to have an additional property

$$encrypt_A(decrypt_A(m)) = m$$

Thus, if A wishes to send a signed, verifiable message m to B, A needs to apply A's private decryption key, and then B's public decryption key to the message, resulting in $encrypt_B(decrypt_A(m))$. This, now, can be sent to B, who will apply his/her private decryption key, resulting in

$$decrypt_B(encrypt_B(decrypt_A(m))) = decrypt_A(m)$$

Applying A's public encryption key will recover the message:

$$encrypt_A(decrypt_A(m)) = m$$

Since only A could have known A's private decryption key, the message must have been truly from A and not corrupted.

4.2 Scheduling and load balancing

One of the most interesting problems in the distributed computing is that of optimal scheduling of processes. Since in the distributed environment applications have more than one host available for computation, it becomes necessary to determine where and when to run a certain process. Scheduling can be done either statically or dynamically.

In static scheduling the assignment of processes to the hosts/processors happens before the execution begins. This method, although very simple, has multiple drawbacks. To achieve any sort of efficiency, the application's behavior needs to be predicted before the application itself is started. Although, knowing the logic of the program, it may be possible to derive some prediction. However, such prediction would not be able to take into account any changes in processing environment that were brought about by other processes occupying the system. That is, it is virtually impossible to predict the processor load due to other applications.

In dynamic scheduling processes are (re)distributed during the execution of the application. The

process of transferring processes from heavily loaded processors to lighter loaded processors is called load balancing. For this scheme to work, several things need to be known to the scheduler/load balancer:

- how loaded are the processors available to the application
- how much effort will it take to transfer the application to a different host/processor
- how loaded is the current processor

Load balancing operations may be centralized in a single processor or distributed among all processing elements. Also, load balancing can be either sender or receiver initiated. In the sender initiated scheme a local host makes a decision as to where a new task is to be executed and sends it there. At the target processor a queue of scheduled jobs may be formed. In the receiver initiated scheme a local host keeps the new task until a processing element reports ready to accept a new job. Load balancing techniques are also classified into static and adaptive. In a static model the scheduling decisions do not depend on the current state of the system, but rather on the average behavior and some predetermined scheme (like round-robin or random). Adaptive load balancers, on the other hand, react to changes in the system state. It is obvious that an adaptive balancer/scheduler is much more complex to implement and will itself produce a heavier load on the system.

5 Developing and experimenting with distributed systems

After having discussed various issues a programmer or software engineer may have to deal with when building a distributed system, it may be helpful to see more concrete examples of such dealings. This section will provide the reader with some of such examples. It will describe development of the general-purpose distributed system. Sections 5.1 and 5.2 will explore the feasibility of distributing such system

across the non-local networks (i.e. networks with relatively long delay times and possibly high error rates) and section 5.3 will focus on design of this system.

5.1 Study of the MPI point-to-point communication methods

Since communication between the processes is one of the most important aspects of a distributed system, it would be wise to see how certain communication tools act under various network conditions.

For this experiment, MPI was chosen as a message passing mechanism. Originally, it was intended to compare the performance of MPI under two different transport protocols – TCP and XTP. That would identify which network protocol is better suited for MPI. However, due to resource and time limitations the experiment was limited only to the TCP suite and did not allow any comparison with XTP. Despite that, it yielded some interesting results that may be helpful not just in the areas of distributed computing, but also in the general data communications research.

The experiment consisted of sending messages of varied length back and forth using MPI point-to-point send mechanisms between two computers. The computers involved were two SPARC-stations connected by an ATM network. The pathways were set up so all the cells traveling between station one and station two were routed through Adtech's AX/4000 test equipment. AX/4000 was used to introduce errors and delays into the cell stream, thus simulating various network conditions (ranging from local area networks to metropolitan area networks to wide area networks with various degrees of errors.) The round-trip time of the message was measured and the rate of message passing was calculated by dividing the size of the message by the round-trip time. It must be noted that the rate found in this experiment is not the network throughput because there is additional overhead added by the MPI message processing mechanism.

Some results of this experiment are shown in figures 1, 2 and 3. Figure 1 shows the relation between

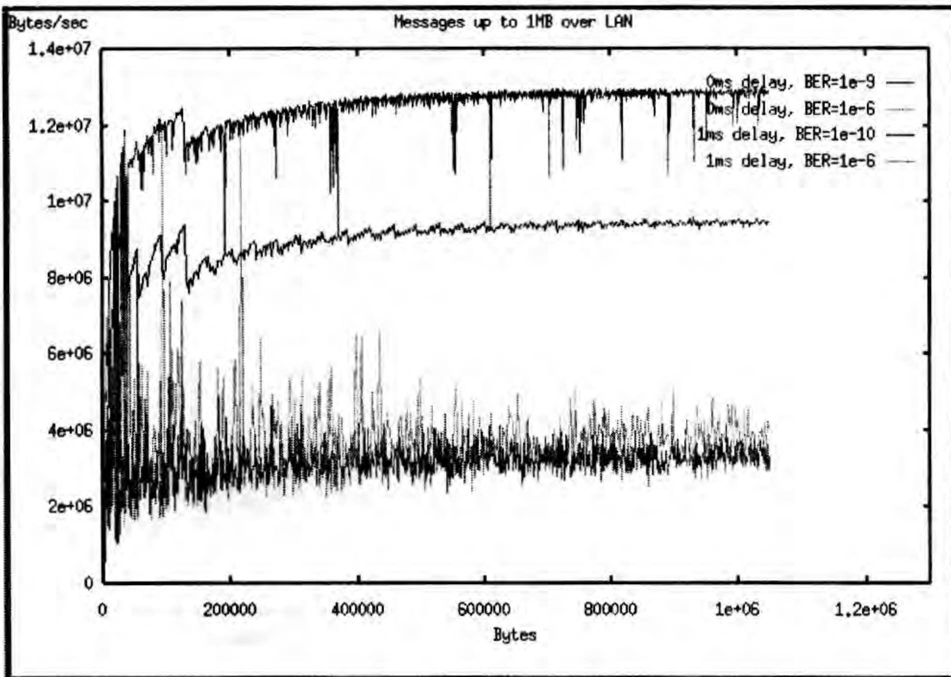


Figure 1: Rate of passing small (up to 1MB) messages over a LAN-type network.

the rate of message passing to the size of the message for the networks with characteristics similar to those of local area network (like a university campus network). The measurements were made for the delay levels of 0 millisecond and 1 millisecond with small and large error rates. It can be seen from the graph that, as the error rate rises, the difference between the 0 millisecond network and 1 millisecond network (otherwise significant) seizes to exist. It also shows that a slower network with lower error rate behaves better than a fast network with a higher error rate. Figure 2 shows the similar dataset for the metropolitan area type of networks (a network connecting campuses in different areas of the metropolis.) Results here are similar – as the error rate increases, the difference between slow and fast networks seizes to exist.

Figure 3 shows the situation described by graph one from a slightly different perspective. Here the round-trip time of the message is plotted against the size of the message for different delay end error rates. Each point on the plot represents a message. At the lower error rates the time values are densely packed

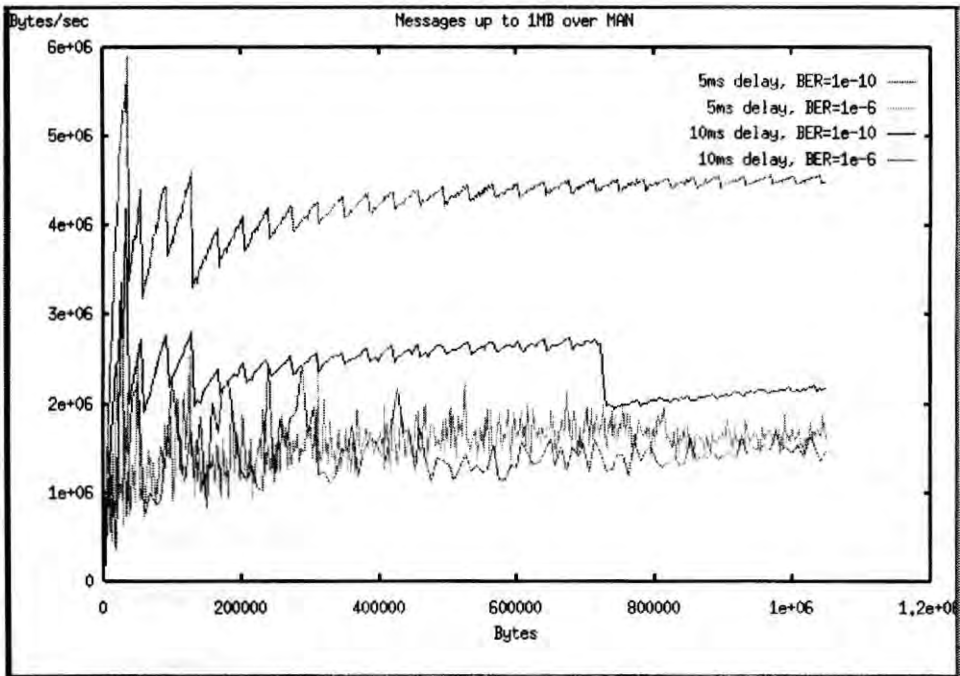


Figure 2: Rate of passing small (up to 1MB) messages over a MAN-type network.

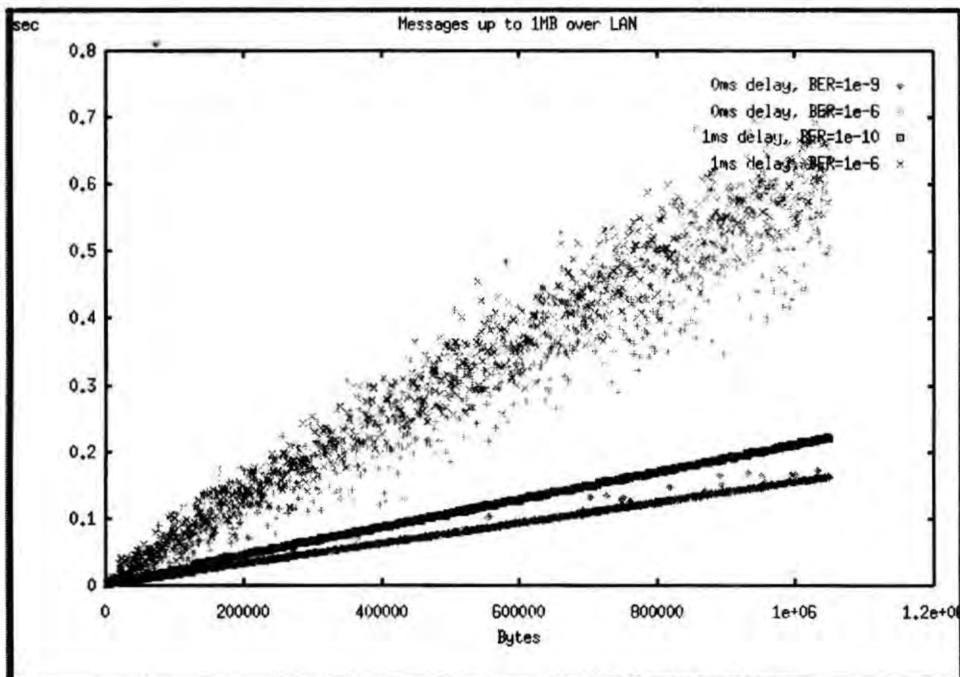


Figure 3: Rate of passing small (up to 1MB) messages over a MAN-type network.

into a line. However, as the error rate increases the time values become more diverse. This provides an insight not only into any particular message passing mechanism, but also into any network and protocol benchmarking. Normally, a network is benchmarked by providing a throughput vs. message size graph for any given network, protocol, delay or error rate. Throughput is a ratio of message size to the round-trip time. Normally, the round-trip time is considered a definite value for any given condition, thus producing a single-line throughput graph. The data represented in figure 3 shows that the round-trip time should be considered more like a probability function rather than any definite value. As the experiment has shown, as the error rate increases, the distribution becomes less dense. This approach would be very helpful in the evaluation of different protocols. For example, when one tries to compare XTP and TCP protocols, the throughput graph would not be sufficient, since it would not disclose error handling capabilities of the protocol as well as a probability function graph would (i.e. a protocol with a superior error-reduction techniques would boast a much dense probability distribution for time values.)

Although the XTP implementation was not available to make the comparisons, it is possible to hypothesize that in the environments with increased error rates, XTP would be a better transport agent that TCP because XTP is supposed to have superior error-handling capabilities.

The source code of the program used to conduct this experiment can be viewed at

`http://ava.obu.edu/~morozov/VSEP98/src/commtest.c`

5.2 Some results in distributing processes over "slow" networks

Most of the reasons given for distributed computing were given in the light of fast local networks. Currently, even with the advances in the technology, the speed of Internet has much room for improvement. Thus, one could argue that relatively slow transmission speeds of the Internet would outweigh the benefits of distributing tasks to multiple hosts. This is certainly true of the tasks where communication is a rela-

tively large component compared to computation. However, it is conceivable that some applications do not require much communication. These applications should not suffer much from lower network speeds.

To test this idea, a simple experiment was set up. In this experiment a computational problem was distributed to four processors separated by a "non-fast" network and the performance was tested and compared to the performance of a program executing the same problem in sequential mode.

The computational problem consisted of squaring large tables of integers. Distributed implementation consisted of computing elements and a "manager" process that divided the problem into several parts and assigned them to the computing elements. The program was written in C using the MPICH implementation of MPI. Four workstations running Solaris operating system were available. These were connected by an ATM network through the AX/4000 test equipment. AX/4000 was used to simulate various network environments by introducing delays and errors into the traffic.

Results of this experiment are shown in figure 4. It is clearly seen that there is virtually no difference between the times it takes to compute the problem on the network with 10 millisecond delays or 180 millisecond delays. It is true that in this case the distributed method is not very much faster than the sequential, but that is mainly due to the fact that only 4 nodes were available for processing.

The source code of the program used in the experiment is available at:

`http://ava.obu.edu/~morozov/VSEP98/src/mat-star.c`

5.3 Developing a general-purpose distributed system

Having shown that distributed processing over the Internet is possible and even beneficial, one might want the reader to get a feel of what it would be like to develop a distributed application. This section will briefly discuss the design of a general-purpose system distributed over multiple computers.

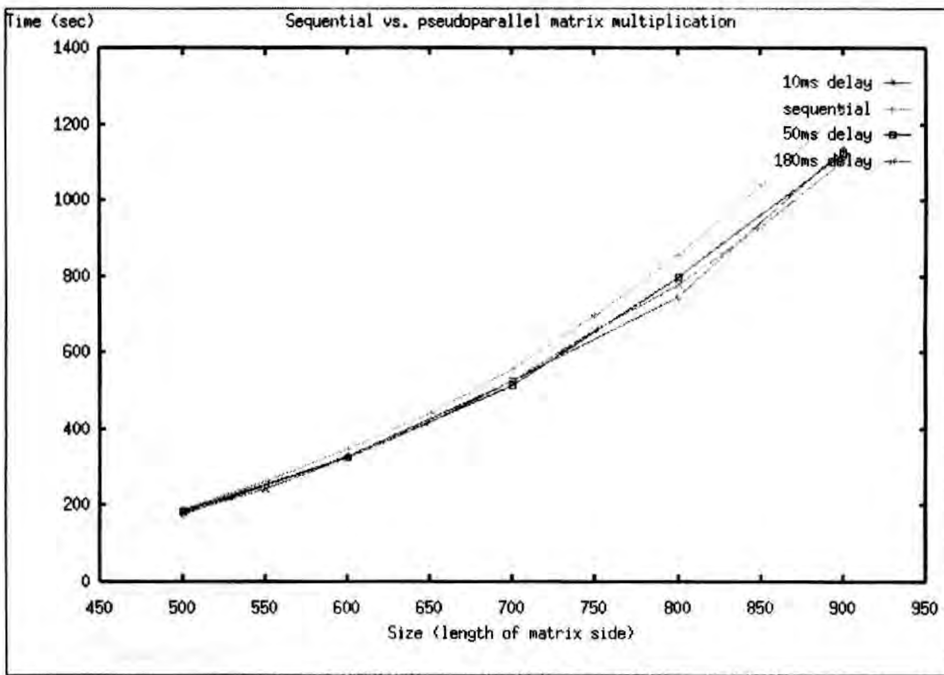


Figure 4: Comparison of sequential and parallel table multiplication utilizing networks of various speed.

5.3.1 System requirements

Before proceeding to the design, it is necessary to state the features one would want this system to have.

Firstly, the model should be general enough to be adapted to a multitude of problems. This could be understood in two ways:

- construct the model as a superclass from which multiple systems could be derived, each system dealing with an individual problem
- generalize even further and design a system that would alone deal with varied tasks

As often happens, the increased generality could come at a cost of lesser efficiency. These trade-offs could be studied once prototypes of different models are implemented.

Secondly, the system should not be limited to a Local Area Network, but must be able to distribute across MANs, WANs, or Internet. This means that the system should be fault tolerant. It also implies

the possibility of resources being owned by different entities, thus creating the need for a security model that provides sufficient protection, yet does not require monopoly over the resources to enforce it.

Thirdly, the system should be as transparent as possible. That means that it needs to provide automatic resource management so that user would not be concerned with locating or scheduling tasks.

5.3.2 System design

Having defined the requirements, it is possible to begin thinking about the design. It must be noted, however, that this section will provide a general design of a system, and not the discussion on implementation of such, although some implementation details are mentioned.

It is clear that any system having the above described characteristics should have at least two main parts:

- Client – the process requesting for a problem to be solved
- Server – the process solving the problem and returning the result to the client

However, the combination of generality, transparency and process distribution creates a need for another tier: a resource manager. The resource manager would be responsible for receiving clients' requests and sending them to the appropriate servers. At this point it is helpful to discuss distribution of problems to resources. There are two possible avenues: one-problem-one-resource or one-problem-multiple-resources. It is clear that different problems would require different approaches.

Consider two simplified examples. In case one the client has a number of various polynomials that need to be solved. This is clearly a one-problem-one-resource case, because nothing would be gained by solving a polynomial in distributed manner. However, this problem can be solved in a distributed fashion in a manner where each node on the network solves a particular polynomial. In case two one

has two very large matrices that need to be multiplied. This would be a one-problem-multiple-resources situation, since matrix multiplication could be efficiently done in parallel over several nodes. Having considered both situation one comes to the conclusion that for a system to be truly general in nature, it would have to be able to deal with both types of problems. To allow that, the system will have a four-tier architecture, with the third and fourth tiers being problem dependent. A more detailed description of the four tiers follows.

Tier One – Client This is the tier that would be made available to the end user – the programmer trying to utilize the system. This tier will provide the user-level API and the necessary level of transparency. On the side visible to the user it will provide interface for submitting a problem and returning the result. Everything beyond this side will be invisible to the user. On the other side of this tier will be the mechanisms responsible for contacting a resource manager, submitting the problem and receiving the result. Possible design issues for this layer will be discussed later.

Tier Two – Resource Manager This is the only tier visible to the client tier. It will be responsible for locating an appropriate problem solver, submitting the problem and retransmitting the result to the client tier. This tier would act as a universal scheduler / load balancer. If there are more than one problem solvers available, it would be responsible for choosing the one that would be most efficient in a given situation.

Tier Three – Problem Solver This tier is responsible for solving a problem submitted by the resource manager. It will consist of a collection of problem-solver objects, each knowing how to solve a particular type of problem. If a given problem is of a one-problem/one-resource type, then it would be solved in this tier, and the result would be returned to the resource manager. However, if the problem is of a one-problem/multiple-resource kind, then this tier will act as a resource manager

for this particular problem and launch parallel / distributed execution on the fourth tier.

Tier Four (Optional) – Distributed / Parallel Problem Solver This is the least generalized tier, visible only to the third tier. It consists of units that report to the respective problem solver and are used in solving a particular type of distributed problem.

To provide the highest level of generality, one would have to allow for different tiers and parts of tiers be developed and implemented by various people and organizations. One of the possible ways of doing so, is to utilize the already developed CORBA model (see section 3.3.3). CORBA would allow for the tiers to be constructed of objects, where each object can be developed independently with all the communication being handled by ORBs.

The next issue that comes up is the one of communicating the problem to the resource manager. Because the resource manager is the only tier visible to the client, it must be able to accept all types of problems. Moreover, the abstraction mechanism that is adopted by this four-tier architecture implies that the resource manager itself knows nothing about the problem or its solution – it only knows the third-tier objects that can solve it. This issue can be resolved in two possible ways. One is to define a Problem Definition Language (similar to that of Interface Definition Language of CORBA mentioned in section 3.3.3). However, this language would either severely limit the types of problems for which problem-solver objects can be developed, or would require those objects to have almost human intelligence. Thus, it appears that developing such language is not feasible at this time. Another approach would be similar to the one adopted by the Internet Protocol (section 3.2.2) and similar network standards. IP datagrams carry various types of traffic without really knowing the contents. However, when the packets arrive to the destination, they have to be passed to the appropriate service, each service having its own standard. To allow this, IP includes a number in its datagram header that identifies the type of service that receives the contents. The contents themselves are encapsulated in an IP packet without regard to their meaning.

This requires the sender to know the number of a receiving service. Such a requirement, however, does not pose any problem when the service numbers are standardized and published. To adopt this idea to the client-manager communication issue, the model would assign numbers to the types of problems for which problems-solver objects are developed. The client in this case would not have to provide the explanation of the problem — it would only supply the arguments and the problem number. The arguments would be encapsulated in a message that is attached to a problem number and sent to the resource manager. The resource manager would not have to know which problems correspond to which numbers, it would simply find the problem-solver object in tier three with the corresponding problem number. Then it would retransmit the message it received from the client to the problem-solver object. The most important benefit of this architecture is that the resource manager does not need to be redesigned every time a new problem-solver object is developed for a new type of problem. This approach also solves the question of how the resource manager would communicate the problem to the problem solver.

There is really no issue in communicating between tiers three and four, since the object in those tiers that need to communicate with each other are a part of a single mechanism, and would be written by a single developer (group).

The issue of fault tolerance could be solved by allowing more than one resource manager. Any given manager would send a copy of the problem request to several other managers it knows and inform them who the client is, and who was assigned to solve the problem. In the case that the original resource manager dies or becomes unavailable the secondary managers would be able to step up to the bat. If resources permit and the fault tolerance is of big importance, resource manager could also redundantly assign the same problem to several problem-solver objects.

The issue of security can be broken up into two parts. Firstly, is the question of how much access the resource owners would allow to the outside users. This can easily be regulated by picking which objects

the computer will run. Since the structure of the whole model is highly modular, not all parts have to be run on a computer to participate in the combined distributed system. The second issue is that of ensuring that the object on the other side is really what it says it is. This could be done by augmenting all communication channels with public-private key encryption mechanism (section 4.1).

This section will not discuss the issues related to the design of problem-solver objects. Those are designed to fit the particular problem they are solving and could employ a wide variety of solutions, using CORBA, MPI, PVM, or multitude of other mechanisms. To ensure the compatibility with other objects, though , this modules would have to adhere to a certain argument passing protocol. This could be worked out by various solver designers, since the overall system is oblivious to such details.

These are some of the main, but certainly not all of the issues in developing such a system. More specific details will depend on the implementation which is outside the scope of this paper.

6 Conclusions

This paper has given a brief overview of issues a developer of a distributed system would face. It has also given a generalized architecture of a distributed system that could be used for solving a wide array of problems. This architecture is so far only theoretical and has not been implemented by the author.

References

[1] Thomas E. Anderson, David E. Culler, and David A. Patterson. A case for networks of workstations: NOW. *IEEE Micro*, Febuary 1995.

[2] Distributed Computing Group, Stanford University. *Security in a Public World: A Survey*, 1996.

[3] Al Geist, Adam Beguelin, Jack Donagarra, Weicheng Jiang, Robert Mancheck, and Vaidy Sunderam. *PVM: Parallel Virtual Machine – a Users' Guide and Tutorial for Networked Parallel Computing.* The MIT Press, 1994.

[4] Andrew S. Grimshaw, William A. Wulf, James C. French, Alfred C. Weaver, and Paul F. Reynolds Jr. A synopsis of the Legion Project. Technical Report CS-94-20, University of Virginia, Department of Computer Science, 1994.

[5] Andrew S. Grimshaw and Wm. A. Wulf. The Legion vision of a worldwide virtual computer. *Communications of the ACM*, 40(1), January 1997.

[6] Internet.com, http://webopedia.internet.com. *PC Webopaedeia.*

[7] Greg Lindahl, Andrew Grimshaw, Adam Ferrari, and Katherine Holcomb. Metacomputing – what's in it for me?

[8] Message Passing Interface Forum. *MPI: A Message Passing Interface Standard*, 1995.

[9] Object Management Group. *The Common Object Request Broker: Architecture and Specification*, 1998.

[10] J. Postel. User Datagram Protocol. Technical Report RFC-768, Information Sciences Institute, University of Southern California, 1980.

[11] Jonothan B. Postel. Internet Protocol. DARPA Internet Program Protocol Specification. Technical Report RFC-791, Information Sciences Institute, University of Southern California, 1981.

[12] Jonothan B. Postel. Transmission Control Protocol. DARPA Internet Program Protocol Specification. Technical Report RFC-793, Information Sciences Institute, University of Southern California, 1981.

[13] John Purcell, editor. *Linux: The Complete Reference.* Linux Systems Labs, 1997.

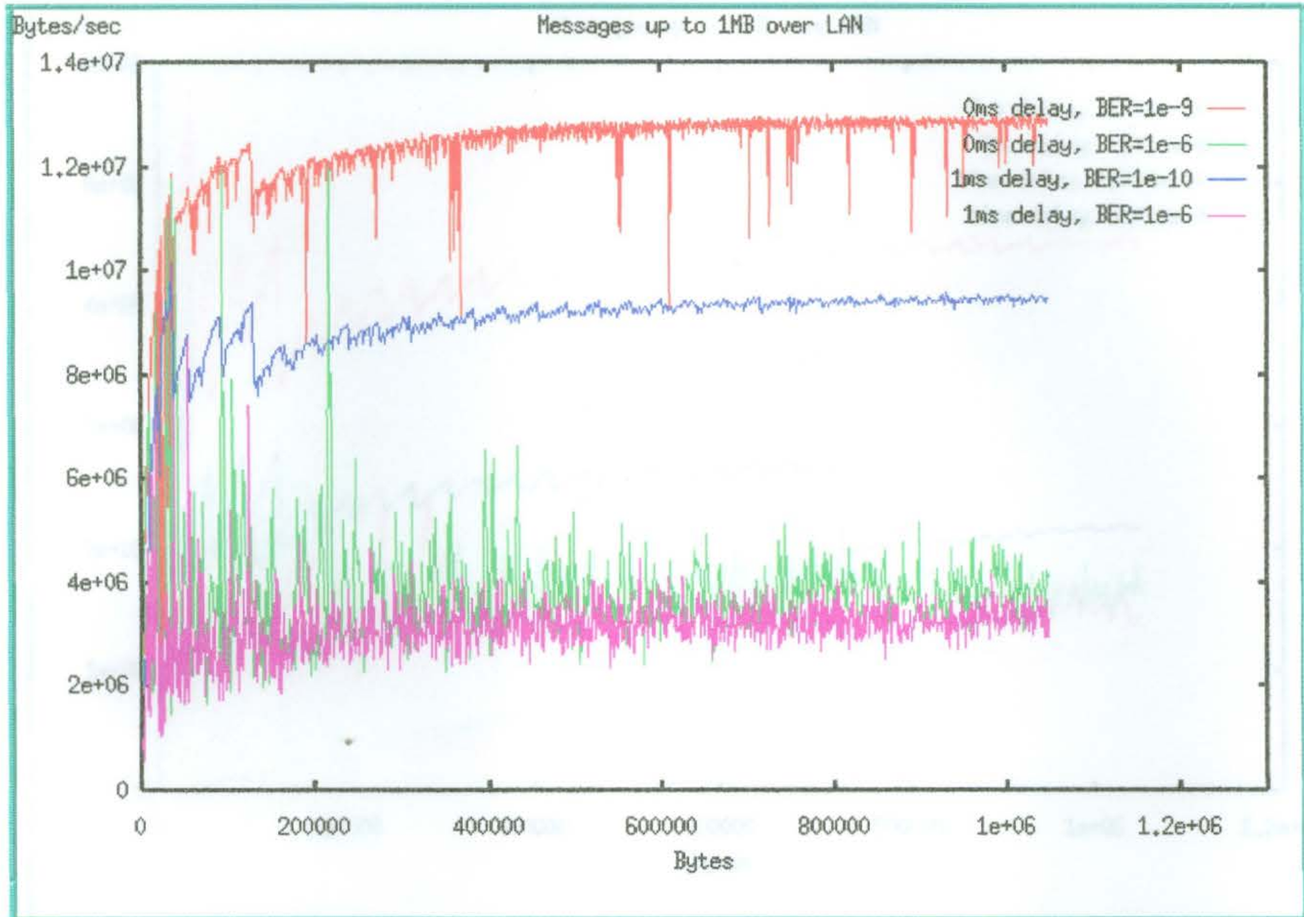
[14] Daniel Ridge, Donald Becker, Phillip Merkey, and Thomas Sterling. Beowulf: Harnessing the power of parallelism in a Pile-of-PCs. In *Proceedings, IEEE Aerospace*, 1997.

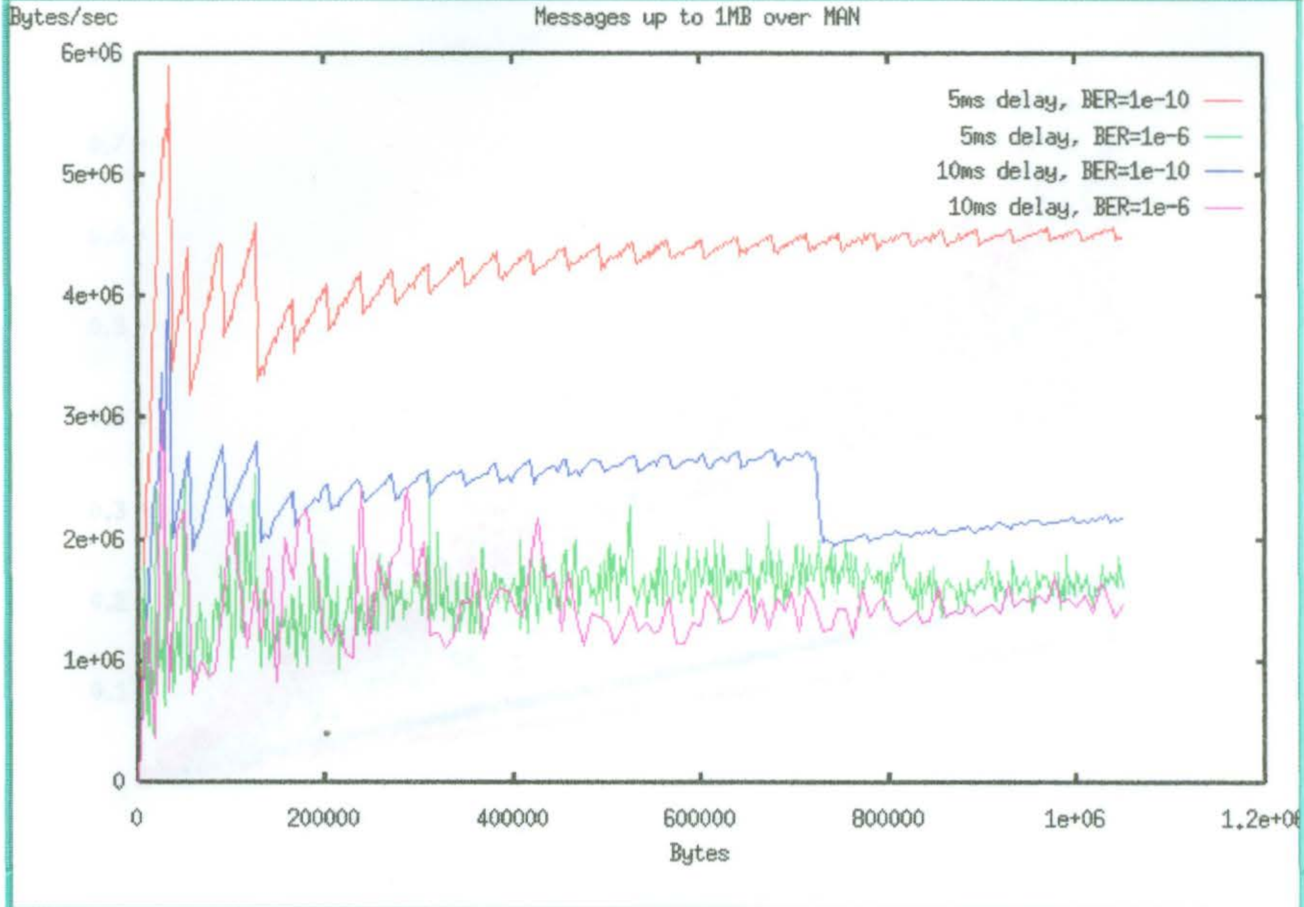
[15] Thomas Sterling, Donald J. Becker, John E. Dorband, Daniel Savarese, Udaya A. Ranawake, and Charles V. Packer. Beowulf: A parallel workstation for scientific computation. In *Proceedings, International Conference on Parallel Processing*, 1995.

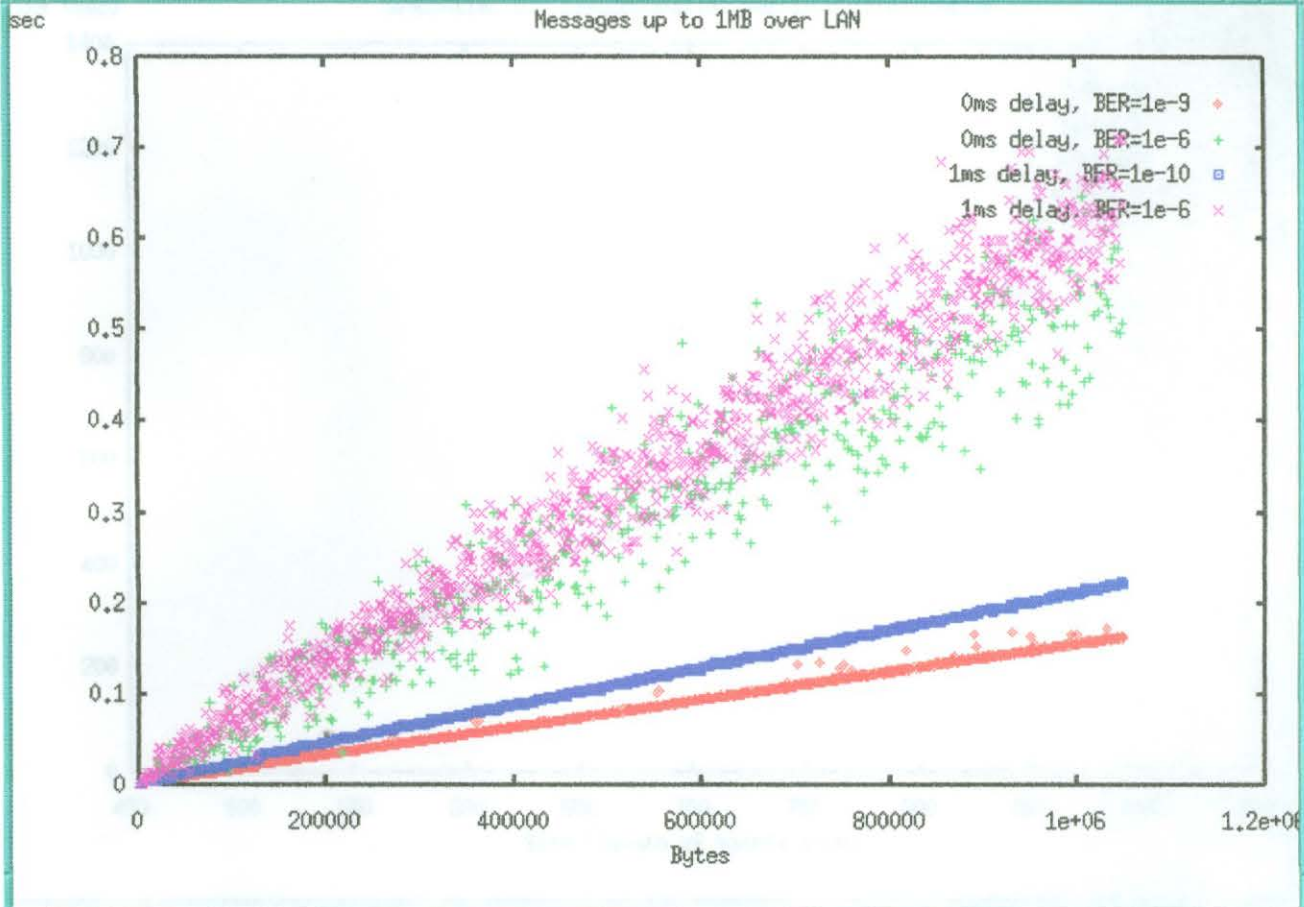
[16] Tim Strayer. Xpress Transport Protocol specification. Technical Report XTP 95-20, XTP Forum, 1995.

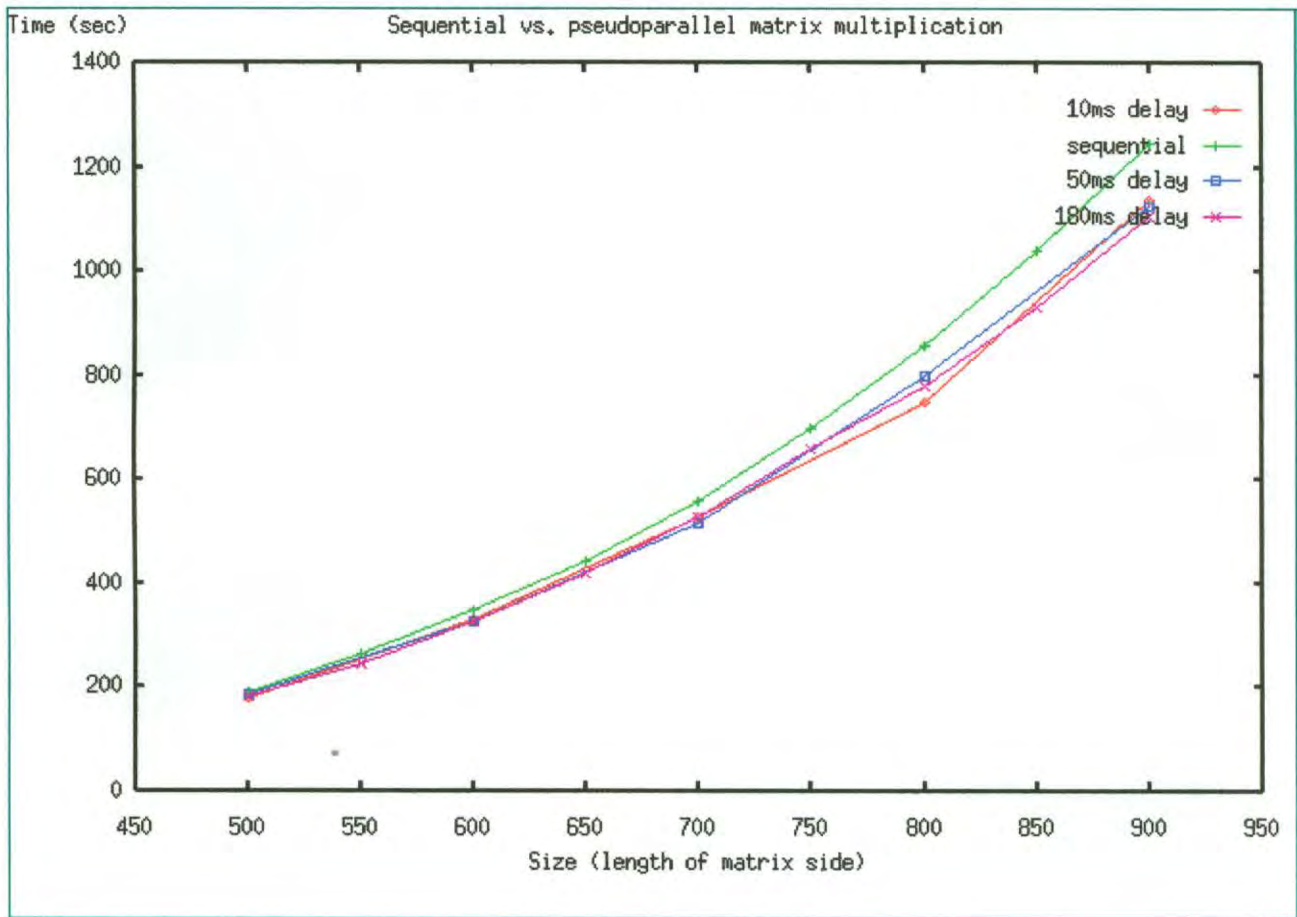
[17] Andrew S. Tannenbaum. *Distributed Operating Systems.* Prentice Hall, 1995.

[18] University of California, Berkley, http://setiathome.ssl.berkeley.edu. *SETI@home: Search for Extraterrestrial Intelligence at Home.*


Bytes/sec
Messages up to 1MB over LAN
1.4e+07
1.2e+07
1e+07
8e+06
6e+06
4e+06
2e+06
0
0
200000
400000
600000
800000
1e+06
1.2e+06
Bytes
0ms delay, BER=1e-9
0ms delay, BER=1e-6
1ms delay, BER=1e-10
1ms delay, BER=1e-6


Bytes/sec
Messages up to 1MB over MAN
5ms delay, BER=1e-10
5ms delay, BER=1e-6
10ms delay, BER=1e-10
10ms delay, BER=1e-6
6e+06
5e+06
4e+06
3e+06
2e+06
1e+06
0
0
200000
400000
600000
800000
1e+06
Bytes


Messages up to 1MB over LAN
sec
0.8
0.7
0.6
0.5
0.4
0.3
0.2
0.1
0
0
200000
400000
600000
800000
1e+06
1.2e+06
Bytes
0ms delay, BER=1e-9
0ms delay, BER=1e-6
1ms delay, BER=1e-10
1ms delay, BER=1e-6


Time (sec)
Sequential vs. pseudoparallel matrix multiplication
1400
1200
1000
800
600
400
200
0
10ms delay
sequential
50ms delay
180ms delay
450
500
550
600
650
700
750
800
850
900
950
Size (length of matrix side)