International Journal of Information, Control and Computer Sciences
ISSN: 2517-9942
Vol:1, No:7, 2007

Network Based High Performance Computing

Karanjeet Singh Kahlon, Gurvinder Singh, and Arjan Singh

Abstract**—**In the past few years there is a change in the view of high performance applications and parallel computing. Initially such applications were targeted towards dedicated parallel machines. Recently trend is changing towards building meta-applications composed of several modules that exploit heterogeneous platforms and employ hybrid forms of parallelism. The aim of this paper is to propose a model of virtual parallel computing. Virtual parallel computing system provides a flexible object oriented software framework that makes it easy for programmers to write various parallel applications.

Keywords**—**Applet, Efficiency, Java, LAN

I. Introduction

THE power of Internet and Intranet can be used for integrating remote and heterogeneous computer into a single global computing facility for parallel and collaborative work. To gain control over the resources of Internet-based computers for parallel computing has introduced new difficulties and problems that have never been addressed by parallel computing in LAN (local area network) environment [3].

Some of the difficulties are the heterogeneity of the participating systems, difficulties in administering distributed applications, security concerns of users, and matching of applications and users. The proposed virtual parallel computing system examines the aspect of high-performance computing.

The scope of the virtual parallel computing system is in the possibility of carrying out computations that require very large computational power with the cooperation of processors on LAN. Execution time and result of the computation are the only parameters of interest. Another simplification is by dealing mostly with the use of CPU-time of the processors and less with other resources like memory. Finally, focus is on very large computations.

II. RELATED WORK

The use of local area networks as a parallel computing platform has been explored for many years. Numerous research projects have aimed at this goal. A report [11] indicates that the number of registered Internet Protocol (IP) [24] addresses has been growing at 50%-80% per year.

Other reports [15], [27], [16], [14], [2] indicate that most networked machines in typical commercial organizations and universities are underutilized and mostly idle. Given this wealth of unused computing power, it is not surprising that several projects [5], [6], [23], [22], [21], [30] have used Internet to execute programs that are too compute intensive even for a supercomputer. For example, in discovering the world's largest known prime number [31], Prime Net reported a sustained throughput of more than 200 billion point instructions per second, the equivalent of seven fully-equipped Cray T916 supercomputers at peak performance [29]. In another case, during the first successful brute force crack of a DES-encrypted message, it was observed that within a single 24-hour period nearly 14,000 machines joined the computation [4]; this is a tremendous amount of computing power.

There are many software systems [8], [19], [28], [12], [20] for developing and executing programs using idle CPU cycles in networks of workstations. A comprehensive end-to-end solution for virtual parallel computing on the LAN must address the concerns of programmers, users, and clients. Implementing virtual parallel computing systems involves many interesting and challenging technical questions like speedup, flexibility, maximum utilization, simplicity, accessibility, applicability and reliability and problems that are open areas for research.

III. Idea Of Virtual Parallel Computing

Virtual parallel computing is a variation on the idea of parallel computing. It uses network of many separate computers as if they were one large parallel machine, or Parallel computer. Parallel computing systems today primarily take the form of networks of workstations, or NOWs [16]. This allows people to pool together existing and mostly idle workstations in their own local or institution-wide networks. It uses them to do parallel processing without having to purchase an expensive supercomputer [7]. Global-scale NOWs, use computers geographically distributed around the world and communicate through the Internet. They have been used with great success to solve large parallel problems as far back as the early 1990's [26], [13] and until recently [9], [32], [25], [18].

Manuscript received August 20, 2005
Karanjeet S. Kahlon, Gurvinder Singh, Arjan Singh are with Department of Computer Science & Engineering, Guru Nanak Dev University, Amritsar, India (phone 91-183-2258802-09, Email: karanvkahlon@yahoo.com, gsbawa71@yahoo.com, bhinderas@rediffmail.com)

IV. Performance Metrics

The parallel computing literature is rich in examples of parallel algorithms that exhibit unbounded parallelism. Each, of course, requires that workload and number of processors increase indefinitely. Perhaps the oldest and most quoted observation about limits to attainable speedup is Amdahl's Law [1]. This law makes the observation that if one is to solve a problem which contains both a serial component i.e., portion of computing that cannot be parallelized and a parallel component i.e., portion that can be run on a P processor parallel computer with speedup P then the observed speedup will be

$$\frac{s+p}{s+p/P} \tag{1}$$

Where, s and p are respectively the times needed to execute the serial and parallel components on a single processor (in general, s and p will be functions of some characteristics of the instance being solved, e.g., the size of the instance). The definition of speedup used by Amdahl corresponds to that of relative speedup. Serial component remains a constant fraction of the total serial work even as the workload increases.

When dealing with problems for which $c \le s/(s+p) \le 1$ for some constant c, the attainable speedup is bounded even in the face of increasing workloads and number of processors. From Amdahl's law, one gets

$$\text{Speed up} = \frac{s+p}{s+p/P} = \frac{1}{\frac{s}{s+p}+\frac{p}{P(s+p)}} \le \frac{1}{c+\frac{p}{P(s+p)}} \le \frac{1}{c} \tag{2}$$

So, if 99% of the workload is parallelizable, then the maximum speedup attainable is 100; if 99.9% is parallelizable, the maximum attainable speedup is 1000; and if only 99.99% is parallelizable, the speedup cannot exceed 10,000. This reasoning was used by Amdahl to conclude that parallel computers could not deliver significant speedup in practice as to do this the serial component would have to be very small. Gustafson's law [10] can be stated as

$$P - \frac{s}{p}(P-1) \tag{3}$$

It is proposed to use this formula for calculating the speedup and compares the speedup obtained with the Amdahl's law. Efficiency is a performance metric closely related to speedup. It is the ratio of speedup and the number of processors P.

$$\text{Efficiency} = \frac{speedup}{P} \tag{4}$$

V. System Overview

The basic function of the virtual parallel computing system is to provide any programmer on the Intranet with a simple virtual parallel computer. This virtual machine is implemented by utilizing all processors on the network that care to participate at any given moment. The system is implemented in Java and relies on its ubiquitous applet mechanism for enabling wide scale safe participation of remote processors.

There are three types of entities in the virtual parallel computing system

- The computational extensive application program which is to be parallellized written in Java and is divided into sub-tasks and stored in a database in a queue. These sub-tasks will be transferred to the client machine for execution.
- The server program which interacts with clients and waits on a well known port for clients to connect on its local ports, once the connection is established sub-tasks are transferred to client machine and results are stored back in sever after execution.
- An applet program available on the web page which when downloaded using the Internet browser on the client side establishes the connection with server for execution of sub-task on its CPU.

The basic architecture of the proposed system is shown in figure 1. The application program achieves its parallelism by concurrently spawning off many sub-computations. These sub-computations are stored in a queue, which then forwards them to connected clients. The clients execute the sub-computations and return their results to the server, which forwards them back to the application program. A set of computers on a local area network is connected using TCP/IP protocol and interacts with each other by stream socket connections. TCP/IP sockets are used to implement reliable, bi-directional, persistent, point-to-point, and stream-based connections between machines on the network.

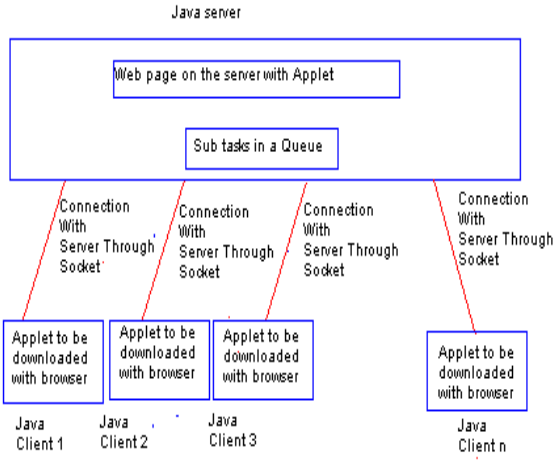

Fig. 1 Basic architecture of the system

VI. EXPERIMENTAL SETUP

For the sake of consistency in the measurement of the experimental results, a homogeneous collection of 20 Pentium PCs 1.5 GHz with 128M RAM, running Windows 98 OS, is used to form the Intranet. All the machines are connected to the each other via 10/100Mbps links. An integration problem with different size of interval was taken as application problem and was divided into different number of sub tasks. The sub tasks were stored in a queue at server. As clients connect to the server at different intervals these sub tasks are allocated to them for execution. The time to complete the whole computation with different numbers of client computer was recorded. The data obtained from these experiments is analyzed for speed up and efficiency of the system developed.

A. Server

Establishing a server in Java requires five steps. Step 1 is to create a Server-Socket object. A call to the Server Socket constructor such as

ServerSocket s = new ServerSocket (port, queueLength);

Registers an available port number and specifies a maximum number of clients that can request connections to the server i.e., the queue Length. If the queue is full, client connections are automatically refused. The preceding statement establishes the port where the server waits for connections from clients also known as binding the server to the port. Each client will ask to connect to the server on this port.

Each client connection is managed with a Socket object. Once the Server Socket is established in Step 2, the server listens indefinitely or blocks for an attempt by a client to connect. This is accomplished with a call to the Server Socket accept method as in

Socket connection = s.accept ();

which returns a Socket object when a connection is established. Step 3 is to get the Output Stream and Input Stream objects that enable the server to communicate with the client. The server sends information to the client via an Output Stream object. The server receives information from the client via an Input Stream object. To obtain the streams, the server invokes method getOutputStream on the Socket to get a reference to the Output Stream associated with the Socket and invokes method getInputStream on the Socket to get a reference to the Input Stream associated with the Socket.

Step 4 is the processing phase in which the server and the client communicate via the Input Stream and Output Stream objects. In Step 5, when the transmission is complete, the server closes the connection by invoking the close method on the Socket. Speedup is one of the major criteria for measuring the performance of a parallel system, With above information for the job executed the speedup and efficiency which is proposed to be calculated by using Amdahl's law [1] and Gustafson's law [10].

B. Client

Establishing a client in Java requires four steps. In Step 1, a Socket is created to connect to the server. The connection to the server is established using a call to the Socket constructor with two arguments the server's Internet address and the port number as in

Socket connection = new Socket (serverAddress, port);

If the connection attempt is successful, this statement returns a Socket. A connection attempt that fails throws an instance of a subclass of IOException; so many programs simply catch IOException.

In Step 2, Socket methods getInputStream and getOUtputStream are used to get references to the Socket's associated Input Stream and Output Stream, respectively. Input Stream method read can be used to input

VII. RESULTS AND ANALYSIS

Experiments were repeated a number of times to verify the consistency in the results.

The whole computation was conducted using three application programs with different parameters as follows.

First application program was an integration problem with range from 0 to 150 named Cat-A; the second was from 0 to 300 named Cat-B and third was from 0 to 450 named Cat-C. Further each problem like Cat-A was executed in four sub categories with increasing size of computation

Named A1SizeU, A1SizeV, A1SizeW and A1SizeX. Each Problem Cat-A, Cat-B and Cat-C were executed by dividing the range into interval of size 1, 2, unto 5.

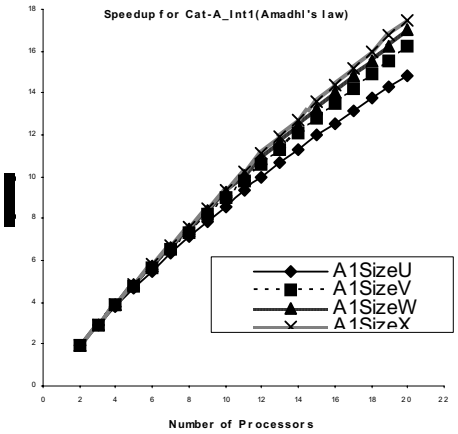

Fig. 2 Speedup for Amdahl's law

The Figure 2 shows the speedup obtained according to Amdahl's law for four jobs a1sizeU, a1sizeV, a1sizeW and a1sizeX for the category a and with area size of one. Starting from two processors the jobs ran in 8.306, 12.137, 16.079 and 19.817 seconds. This corresponds to a speedup of 1.967, 1.975, 1.981 and 1.985 respectively which indicates that as the amount of computation in a job is increased there is a change in the speedup from 1.967 to 1.985, a change of around .9 percent.

The same jobs on twenty processors ran in .256, 12.136, 16.07 and 19.842. This corresponds to a speedup of 14.868, 16.196, 16.987 and 17.486 respectively. This indicates that as there is an increase in the amount of computation in a job there is a change in the speedup from 14.868 to 17.486 a change of 14.97 percent.

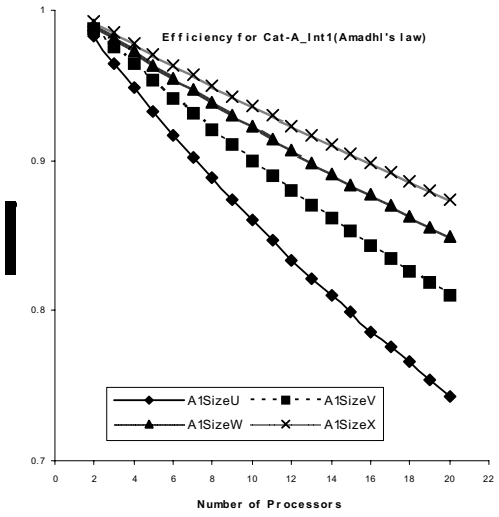

Fig. 3 Efficiency of the system

The Figure 3 shows the efficiency of the system. According to Amdahl's law for four jobs a1sizeU, a1sizeV, a1sizeW and a1sizeX for the category A and with area size of one. The efficiency is 98.23%, 98.8%. 99.089% and 99.24% respectively for two processors, which indicates that as the computation is increased in, jobs the change in efficiency if from 98.23% to 99.24% which is a change of one percent. The efficiency is 74.34%, 81%. 84.9% and 87.4% respectively for twenty processors, which indicates that as the computation is increased in, jobs the change in efficiency if from 74.34% to 87.4% which is a change of 13%.

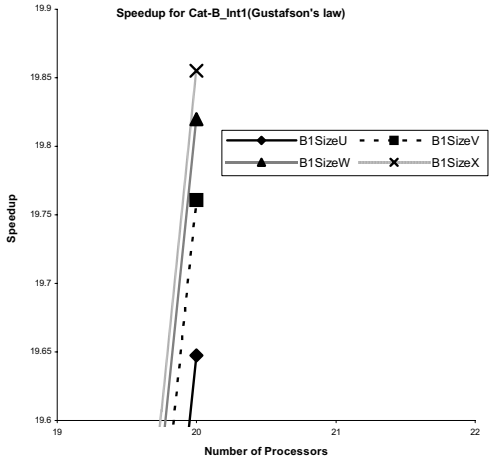

Fig. 4 Speedup for Amdahl's law

The figure 4 shows the speedup obtained according to Gustafson's law for four jobs a1sizeU, a1sizeV, a1sizeW and a1sizeX for the category A and with area size of one with twenty processors. This graph corresponds to table 1 which shows that Starting from two processors the jobs ran in 8.306, 12.137, 16.079 and 19.817 seconds. This corresponds to a speedup of 1.981, 1.987, 1.990 and 1.992 respectively which indicates that as the amount of computation in a job is increased there is a change in the speedup from 1.967 to 1.985, a change of around .55 percent. Also the table 1 indicates that there is a linear increase in the speedup corresponding to the number of processor i.e. for two processors the speedup is 1.981, for three processors the speedup is 2.963 and so on for twenty processors the speedup is 19.65. In graph 6.3 the speedup is shown for twenty processors the jobs ran in 8.256, 12.136, 16.070 and 19.842 seconds. The speedup obtained is 19.65, 19.76, 19.82 and 19.85 respectively, which indicates the change from 19.65 to 19.85, a change of 1.007 percent.

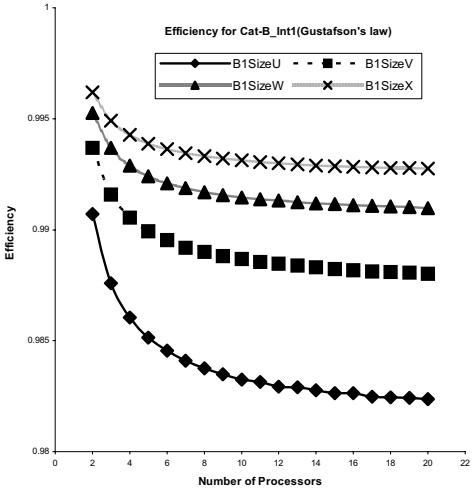

Fig. 5 Efficiency of the system

The Figure 5 shows the efficiency of the system according to Gustafson's law for four jobs a1sizeU, a1sizeV, a1sizeW and a1sizeX for the category a and with area size of one. The efficiency is 99.1%, 99.3%. 99.589% and 99.62% respectively for two processors, which indicates that as the computation is increased in, jobs the change in efficiency if from 99.1% to 99.62% which is a change of .5%. The efficiency is 98.27%, 98.82%, 99.11% and 99.28% respectively which indicates that as the computation is increased in jobs the change in efficiency if from 98.27% to 99.28% which is a change of 1.01%.

The above analysis indicates that according to Amdahl's law increase in the number of processor along with the increase in amount of computation gives better speedup as well as efficiency. Whereas according to Gustafson's law there is a

linear increase in speedup as well as efficiency of the system corresponding to the increase in the number of processors.

VIII. Conclusion

The proposed Virtual Parallel Computing system represents an environment for distributed computing. Large number of heterogeneous machines with different speeds and configuration can participate to execute parallel jobs. Many networked computers in parallel work at the same time on different parts of the same problem. A LAN-based parallel computing environment using Java programming language has been created to perform parallel computing. Using inexpensive personal computers and workstations to solve complex problems has increasing appeal. Virtual parallel computing supports some of the key functionality like Speedup, Ease of use, Security, Platform Independence and Code Mobility, Maximum utilization, Robustness, Openness and scalability.

References

[1] Distributed. Net: Bovine, Project Bovine. Available at *http://www.distributed. Net/rc5.*
[2] Distributed. Net: Monarch, Project Monarch. Available at *http://www.distributed.net/des*.
[3] Fields: 1993, S. Fields. Hunting for Wasted Computing Power: New Software for Computing Net- works Puts Idle PC's to Work. Research Sampler. University of Wisconsin Madison. 1993. *URL: http://www.cs.wisc.edu/condor/doc/WiscIdea.html*.
[4] Freeman: 1995, N. Carriero, E. Freeman, D. Gelernter, and D. Kaminsky. Adaptive parallelism and Piranha. Computer, 1995.
[5] Gibbs: 1997, W. Gibbs. Cyber View. Scientific American, May 1997.
[6] Gustafson: 1988, J. Gustafson, Reevaluating Amdahl's Law, CACM, 31, 5, 532-533, 1988.
[7] Internet Domain Survey: 1998 Network Wizards. Internet Domain Survey, July 1998. Available at *http: //www.nw.com/zone/WWW/report.html.*
[8] Kedem: 1995, A. Baratloo, P. Dasgupta, and Z. M. Kedem. Calypso: A novel software system for fault-tolerant parallel processing on distributed platforms. In Proceedings of International Symposium on High-Performance Distributed Computing (HPDC), 1995.
[9] Levy: 1996, S. Levy. Wisecrackers. Wired, issue 4.03, Mar. 1996. URL: http://www.wired.com/ wired/archive/4.03/crackers.html .
[10] Livny: 1991, M. Mutka and M. Livny. The available capacity of a privately owned workstation environment. In Performance Evaluation, 1991.
[11] Nichols: 1987, D. Nichols. Using idle workstations in a shared computing environment. In Proceedings of SOSP, 1987.
[12] Ousterhout: 1991, F. Douglis and J. Ousterhout. Transparent process migration: Design alternatives and the Sprite implementation. Software Practice and Experience, 1991.
[13] Patterson: 1995, T.E. Anderson, D.E. Culler, D. A. Patterson, and the NOW Team. A case for networks of workstations: NOW. IEEE Micro, Feb. 1995. *URL: http://now.cs.berkeley. edu/Case/case.html*
[14] Pearson: URL, K. Pearson. Internet based Distributed Computing Projects. *URL: http://www. nyx.net/"kpearson/distrib.html* .
[15] Prouty:1995, J. Casas, D. Clark, R. Konuru, S. Otto, R. Prouty, and J. Walope. MPVM: A migration transparent version of PVM. Computing Systems, 1995.
[16] Pruyne: 1996, J. Pruyne and M. Livny. Interfacing condor and pvm to harness the cycles of workstation clusters. Journal on Future Generations of Computer Systems, 1996.
[17] RSA Data Security: 97, RSA Laboratories Secret-Key Challenge. Available at *http://www.rsa.com/rsalabs/97challenge*.
[18] RSA Data Security: des2, RSA Laboratories DES Challenge II. Available at http: //www.rsa.com/rsalabs/des2.
[19] RSA Data Security: URL, RSA Data Security RSA Factoring Challenge.
[20] Salus: 1995, P. Salus. Casting the Net: From Arpanet to Internet and Beyond. AddisonWesley, 1995.
[21] SETI@Home, URL: *http://www.computer.org/cise/articles/seti.htm*
[22] Strumpen: 1995, V. Strumpen. Coupling Hundreds of Workstations for Parallel Molecular Sequence Analysis. Software: Practice and Experience, 25(3), 1995, pages 291—304.
[23] Theimer: 1989, Theimer and K. Lantz. Finding idle machines in a workstation-based distributed system. IEEE Transactions on Software Engineering, 1989.
[24] Vesseur: 1994, L. Dikken, F. van Der Linden, J. Vesseur, and P. Sloot. DynamicPVM: Dynamic load balancing on parallel systems. In Proceedings High-Performance Computing and Networking, 1994.
[25] Woltman :1998, G. Woltman. Internet PrimetNet server. Available at http://www. Entropia.com/primenet/status_htm, 1998.
[26] Woltman: URL, George Woltman. Great Internet Mersenne prime search. Available at http: //www.mersenne.org/.
[27] Woltman: 1998(GIMPS), G. Woltman. GIMPS discovers 37th. Known Mersenne prime. Available at http://www.mersenne.org/3021377.htm, 1998.
[28] Wulf: 1993, W. Wulf. The Collaborators Opportunity. Science. Aug. 1993.