Characterizing NAS Benchmark Performance on Shared Heterogeneous Networks

Jaspal Subhlok **Shreenivasa Venkataramaiah** **Amitoj Singh**

Department of Computer Science
University of Houston
Houston, TX 77204
{jaspal, shreeni, amitoj}@cs.uh.edu

Abstract

The goal of this research is to develop performance profiles of parallel and distributed applications in order to predict their execution time under different network conditions. This paper measures the resource requirements of the NAS benchmark programs and characterizes their performance in a shared heterogeneous environment. The programs in the benchmark suite were executed on a controlled testbed and their usage of CPU, bandwidth, and memory were measured. The performance of the benchmark programs was also measured under controlled sharing of CPU and bandwidth. The results are used to characterize the behavior of the NAS benchmark programs with resource sharing. The paper demonstrates that the core system activity of a program can be accurately measured by passive probing, and that this measured system activity is the key to the prediction of program performance when resources must be shared. Our methods rely on system level measurements alone, and therefore, application knowledge or access to the source code, is not required. Hence, the techniques apply across programming languages and models. This paper is an important step towards building an automated framework to infer execution characteristics and estimate performance on shared networks. Such a framework has an important role in resource selection in shared clusters and grid computing environments.

1 Introduction

Shared networks, varying from workstation clusters to distributed computation grids, are an increasingly important platform for parallel and distributed computing. Performance of an application strongly depends on the dynamically changing availability of resources in such distributed computation environments. Understanding and quantifying the relationship between the performance of a particular application and available resources, i.e., how will the application perform under given network conditions, is important for resource selection and for achieving good and predictable performance. The broad goal of this research is automatic development of performance profiles of applications to estimate their execution behavior under different network conditions.

This research is motivated by the problem of resource selection in shared heterogeneous environments. The basic resource selection problem in a shared computation network can be stated as follows: "What is the best set of nodes and links on the network for the execution of a given application under current network conditions?". A solution to this problem requires measurement and reporting of network status, computing the application performance profile for shared execution, and mapping the application to the network based on the performance profile and the network status. In recent years, significant progress has been made in several of these areas. Systems to measure and predict the availability of resources on a network have been developed, some examples being NWS[18] and Remos[10]. Research has addressed algorithms and systems to select nodes and schedule application classes onto networks [2, 4, 13] but these efforts assume that the applications fit a known simple profile. In practice, applications show diverse structures that can be difficult to quantify. The goal of our research is to automatically develop application profiles to estimate performance in different resource availability scenarios. We believe that this is a critical *missing piece* in successfully tackling the larger problem of automatic resource selection.

This paper is based on experiments with the NAS Parallel Benchmark suite [1]. The basic result of this paper is to characterize the performance of the NAS Parallel benchmarks on shared networks. The paper also establishes system level measurements on a controlled networking testbed as a basis for estimating program performance

Appears in the 11th Heterogeneous Computing Workshop (HCW 02)

on a shared network with limited resource availability. Our approach is to measure the core execution parameters of a program, such as the amount and pattern of data exchanged and CPU utilization of application processes, which are independent of the execution environment. These measurements are the basis of the estimation of the expected performance of the program in heterogeneous or shared environments. We present measurements of the performance of the NAS benchmark programs under limited availability of resources to establish the connection between the core execution parameters and the performance with shared or limited resources. All measurements are made by system level probing, hence no program instrumentation is necessary and there is no dependence on the programming model with which an application was developed. The results establish the different patterns of resource requirements and their relationship to the performance for NAS benchmarks in a shared environment. The paper develops a basis for performance modeling for shared heterogeneous networks.

2 Resource utilization by NAS benchmarks

This section presents measurements and analysis of CPU, bandwidth, and memory usage by the programs in the NAS parallel benchmark suite.

2.1 Experimental setup

All our experimental results are based on the MPI implementations of the NAS benchmarks. The codes used are EP (Embarrassingly Parallel), BT (Block Tridiagonal solver), CG (Conjugate Gradient), IS (Integer Sort), LU (LU solver), MG (Multigrid), and SP (Pentadiagonal solver). All programs are in Fortran 77, except IS, which is a C program. Experiments were performed on a testbed of 500 MHz, dual CPU, Pentium 2 nodes running FreeBSD and MPICH implementation of MPI. These nodes were connected by 100Mbps ethernet links and a full crossbar switch. All benchmarks were compiled with class A size for 4 nodes and executed on 4 nodes. `g77` and `gcc` (Fortran 77 and C compilers from GNU) were used for compilation.

For the experiments described in this section, the programs were executed on a dedicated testbed, i.e., with no other activity on the testbed. The CPU and memory utilization were measured with probes based on the Unix *top* utility. The bandwidth usage was measured by employing *tcpdump* to record the timestamped headers of all TCP packets from the nodes and by using SNMP (Simple Network Management Protocol) queries. Results presented in this paper are based on *tcpdump* although both approaches yield virtually identical numbers as expected. The number of network packets was measured with *iptraf* utility.

In the next section we present results with competing CPU loads and limited bandwidth on network links. External loads on processors were simulated with dummy compute bound processes. The *dummynet* toolkit was employed to control the effective bandwidth on network links, e.g., to reduce the bandwidth on a 100Mbps link to 10Mbps. Dummynet works by intercepting selected network packets and passing them through *queues* to simulate the effects of bandwidth limitations, propagation delays, packet loss, etc. The above discussion applies to experiments with FreeBSD which we selected for presenting results in this paper. The entire suite of experiments was also performed with Linux. The main difference in the methodology with Linux was the use of *NISTNet* which provides the same functionality as dummynet for our purposes. A summary of the results on resource usage by NAS benchmarks is presented in Figure 1.

2.2 Processor utilization

We observe from Figure 1 that most of the benchmark programs show a CPU utilization just under 100% with a relatively small difference between the utilization on different nodes. This indicates that most of these applications are compute intensive, load balanced, and do not spend a lot of time blocked for communication or I/O. The main exception is MG (Multigrid benchmark) with an average CPU utilization around 60%. CG, and IS show average CPU utilization of 78% and 85%, respectively, which is somewhat lower than the other benchmarks. The difference in the CPU utilization for different nodes is very small for most of the benchmarks. The relative exceptions are CG and IS, which show differences around 20% and 10%, respectively, between the nodes with the highest and lowest CPU utilization. Low processor utilization implies that the performance is expected to be more robust with CPU sharing since competing applications may utilize some of the unused CPU cycles. Also, if there is a significant difference between the CPU utilization of different nodes, nodes with lower CPU utilization are likely to be better candidates for CPU sharing. We will address these issues in the next section.

2.3 Bandwidth usage

Figure 1 shows statistics on the number of bytes and number of packets exchanged between pairs of executing nodes as well as the total communication traffic generated by the programs. Links between pairs of nodes that exchange a nontrivial amount of data were designated *busy links*. For some applications all links are busy, while for others only

Benchmark	Computation			Communication					Memory
	Percentage CPU Utilization			Busy Links	Avg. Bandwidth util. on busy links (Mbps)	Total Network Traffic (Mbps)	Packets on the Network		Resident Set Size (MBytes)
	Max	Min	Average				Rate (pkts/sec)	Avg. Pkt Size (Bytes)	
EP	98.8	94.7	96.9	None	0	0	0	0	2.11
BT	98.6	94.9	96.7	All	1.61	9.73	1302	957	76.80
CG	87.1	66.4	78.0	Chain	15.16	45.70	6229	939	1.63
IS	90.3	79.6	84.6	All	9.24	5552	6900	1030	31.74
LU	95.6	93.4	94.8	Ring	1.68	6.90	1205	734	14.33
MG	59.8	58.6	59.6	Ring	4.63	18.75	2537	946	116.73
SP	98.9	92.1	94.8	All	4.28	25.73	3400	969	25.6

Chain : Busy links are {node0 -node1, node1 - node2, node2 - node3}
All : All links are busy
Ring : Busy links are {node0 - node1, node1 - node3, node3 -node2, node2 - node0}

Figure 1: CPU utilization and network traffic generated by NAS benchmarks during execution on a cluster

certain links are busy. This exposes the dominant communication patterns in the programs. BT, IS and SP exhibit an all-all communication pattern, CG a one dimensional nearest neighbor chain pattern, LU and MG a ring pattern, while EP shows negligible communication. Figure 1 also shows a wide variation in the bandwidth consumed by the programs. Interestingly, within every benchmark, the bandwidth consumed is roughly equal on all busy links, and in either direction of a busy link, and therefore only one value for link bandwidth is shown. Based on these observations, the fundamental communication patterns of this benchmark suite are highlighted in Figure 2.

Another interesting observation is that the average packet size varies from 734 bytes for LU to 1030 bytes for IS. This seems to indicate that LU sends smaller messages and IS sends larger messages, which is a known fact [15]. However, we should be careful in drawing conclusions about message sizes from packet sizes, since larger messages are broken up into small packets by the TCP and IP communication layers, and small packets are also generated to acknowledge the receipt of data in TCP transport. Indeed the average message size in IS is reported to be almost 1000 times larger than that in LU, but the average packet size in IS is only around 40% larger than that in LU. In related work, we have developed techniques to reconstruct application level message sequence from network measurements.

2.4 Memory usage

Memory usage per node for the NAS benchmark programs is tabulated in the last column in Figure 1. The amount of memory utilized varies from around 2 Mbytes for EP and CG benchmarks to around 117 MBytes for MG benchmark.

3 Performance of NAS benchmarks on shared networks

We shall present and analyze the measured performance of NAS benchmarks under the following simple network sharing scenarios:

1. Two CPU intensive synthetic jobs were run on one of the nodes concurrently with the benchmarks. The experiment was repeated with loads on different nodes. Since each node has a dual CPU, this implies that the benchmark program nominally gets 2/3rds (or 67%) percent of one CPU, as compared to 100% of one CPU on the dedicated testbed. (Since there is only one thread per node, at most 1 CPU can be used even without any other load)

2. Two CPU intensive synthetic loads were run on each of the four nodes concurrently with the benchmarks. This implies that nominally 2/3rd of 1 CPU was available to the benchmark programs on each node.

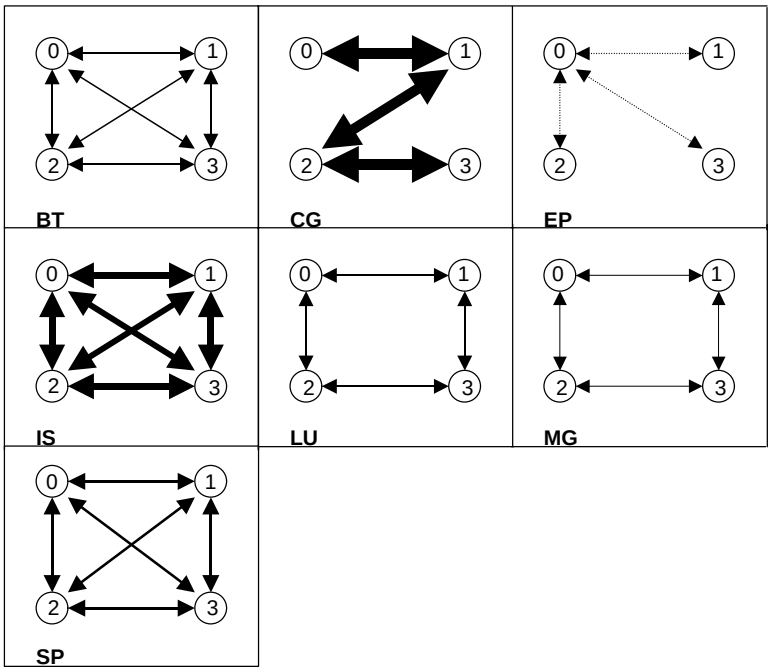

Figure 2: Dominant communication patterns of NAS Parallel Benchmark suite derived from traffic measurements. The width of the line is indicative of the communication bandwidth. The dotted line for EP indicates that the dominant communication pattern is associated with negligible bandwidth.

3. Available capacity on one of the busy links was artificially reduced to 10Mbps from 100Mbps.

4. Available capacity on each of the busy links was artificially reduced to 10Mbps from 100Mbps.

5. One of the communication links was saturated with a synthetic data stream generated by the *netperf* utility, concurrent with the benchmark execution.

6. Different amounts of memory were made available to the programs.

The results from these experiments are tabulated in Figure 3. We omit detailed results from the last two items, that is, execution with a synthetic data stream and with limited amount of memory, but include a brief discussion of the main observations from those experiments.

3.1 Performance with computation loads

Figure 3 includes a summary of the execution times obtained with competing loads. Recall that we nominally expect the execution time to increase by 50% since the CPU availability has nominally reduced from 1 full CPU to 2/3 CPU. Figure 4 graphically compares the percentage increase in execution time for the cases when competing loads are placed on the node with the least busy CPU, the node with the most busy CPU, (i.e., nodes with the highest and lowest CPU utilization during dedicated execution on the testbed) and on all nodes. Presence of competing loads slows down the computation on those nodes and there is also the subtle but important effect of slowing down of the entire computation due to synchronization. Hence the execution speed is determined by a complex interaction between multiple factors. In this paper we observe the performance patterns and point out the main contributing factors.

We observe from Figure 4 that MG, CG, and IS, the benchmarks that stood out earlier for relatively low CPU utilization, also stand out as the ones with the smallest increase in computation time (around 20%) in the presence of one competing load on the least busy node. The reason is that the competing loads partly utilize the unused CPU cycles and therefore have a relatively small impact on the execution time. For CG, the execution is significantly slower with competing loads on the most busy CPU as compared to competing loads being on the least busy CPU. The table at the bottom of Figure 4 points out the reason. CG is the only benchmark with a relatively large difference between the CPU utilization on the most busy and the least busy node.

We now focus on the difference in execution time when

Benchmark	Reference execution time with no competing load or traffic (seconds)	Execution with load on one CPU				Percentage increase in execution time when all nodes are loaded	Percentage increase in execution time with bandwidth limited to 10 Mbps	
		Most busy node is loaded		Least busy node is loaded			On one busy link	On all links
		Percentage CPU Utilization	Percentage increase in exec. time	Percentage CPU Utilization	Percentage increase in exec. time			
EP	102	98.8	48.0	94.7	45.1	50.9	0	0
BT	896	98.6	50.4	94.9	46.2	88.7	11.0	50.5
CG	25	87.1	48.0	66.4	20.0	116.0	116.6	254.1
IS	39	90.3	28.2	80.0	20.5	100.0	77.0	448.7
LU	554	95.6	63.0	94.0	56.5	107.0	8.2	21.8
MG	71	59.8	18.3	58.6	28.1	50.7	40.5	83.7
SP	608	98.9	49.6	92.1	48.8	125.0	35.0	136.6

Figure 3: Performance of NAS benchmarks under different network sharing scenarios

having competing CPU loads on a single node versus having competing CPU loads on all nodes. Figure 4 shows that for the EP benchmark program, the increase in the execution time is approximately around 50% and does not depend strongly on which nodes have competing loads and whether only some or all nodes have competing loads. This is expected since EP does not have any significant communication or synchronization and the program completion time is simply determined by the slowest executing node.

Execution times of CG, IS and SP increase the most with competing loads on all 4 nodes in comparison with execution with competing loads on just one node. The slowdown of an application beyond that of individual CPUs is due to the combination of communication, synchronization and lack of gang scheduling. That is, when one CPU is unavailable due to sharing, all other CPUs may become idle waiting for messages, and each CPU can separately cause this behavior at different times. We point out that CG, IS and SP have the three highest aggregate communication bandwidth demands, which partially explains the higher slowdown, but a detailed analysis of the synchronization patterns is beyond the scope of this paper.

3.2 Performance with low capacity links

Figure 5 shows the impact on the execution time when one of the busy communication links is assigned a 10Mbps capacity rather than the 100Mbps available on the network. The figure also plots the bandwidth utilization on the corresponding links during normal execution. Similarly, Figure 6 shows the increase in the execution time with all links reduced in capacity to 10Mbps and the total network bandwidth used by the application for comparison.

We observe a striking correlation between the bandwidth utilized on a link and the performance if that link is slowed down, as well as the total bandwidth utilization and the performance if all communication is slowed down. The impact of a slowdown in communication can be amplified by synchronization constraints in a parallel program. However, our results indicate that the bandwidth usage is the dominant factor and the effect of other factors is largely uniform across programs in the benchmark. The implication is that the bandwidth utilization is a key indicator of application performance with slow or congested links.

3.3 Performance with congested links and limited memory

We have omitted detailed results on performance with one or more links congested with competing traffic and for execution with limited memory. The reason is brevity and the fact that they did not introduce any new insight. Overall, the results for performance with congested links support the results with limited capacity links. Our measurements of execution time with limited memory were also predictable, at least on the surface. As the total available memory was reduced, there was no impact on the execution time until the available memory approached the mem-

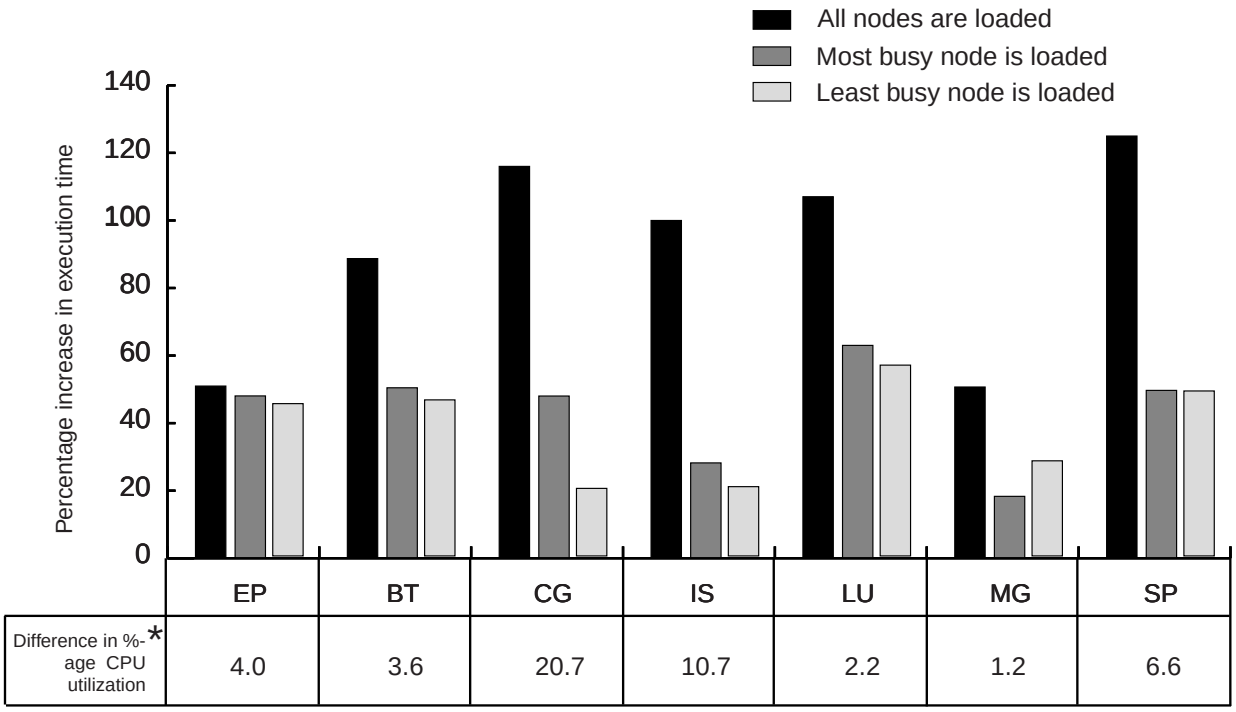

Figure 4: Comparison of execution time with competing compute loads on the node with the most busy CPU, the node with the least busy CPU, and all 4 nodes

ory that the programs needed as shown in Figure 1. When the available memory was reduced further, the execution times increased very rapidly to the point that several of the benchmarks could not be executed in a reasonable amount of time. The conclusion is simply that these programs need the amount of memory that is there requirement in order to execute in a reasonable fashion.

4 Related Work

This research is in the context of shared metacomputing environments pioneered by Globus [7] and Legion [8]. These systems provide support for a wide range of functions, such as resource location and reservation, authentication, and remote process creation mechanisms. Our research is in the broad area of resource selection and management in shared clusters and metacomputing environments.

A number of resource management systems support the selection of computation resources, some examples being Condor [9] and LSF(Load Sharing Facility). However, the selection of communication resources can be just as important and that introduces several new challenges. A number of systems have been developed to measure and forecast network and CPU availability [5, 10, 12, 18]. The main goal of our research is to be able to predict application behavior once a forecast of the CPU and network availability is known. Hence, this research complements network measurement and prediction research. An alternate approach is integrated network measurement and adaptation systems such as [4, 16] that are designed for a particular class of applications.

Several projects have addressed application scheduling on shared networks [2, 13, 17]. These projects address specific classes of applications and assume a simple, well defined structure and resource requirements for their application class. The focus of this research is to quantitatively measure the resource requirements of applications so that more precise algorithms can be used to match the application needs and available resources.

The pattern of usage of MPI in NAS benchmarks has been reported in [15] based on instrumenting the MPI library. A key feature of our approach is that all mea-

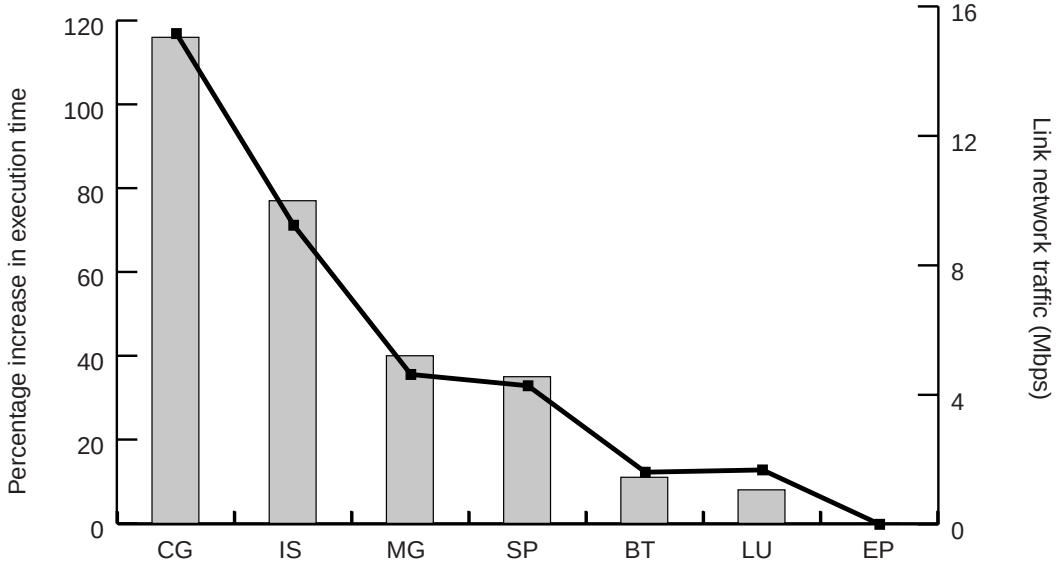

Figure 5: Execution time with the capacity of one busy communication link reduced to 10Mbps from 100 Mbps. Also shown is the bandwidth utilization on the busy link under normal computation

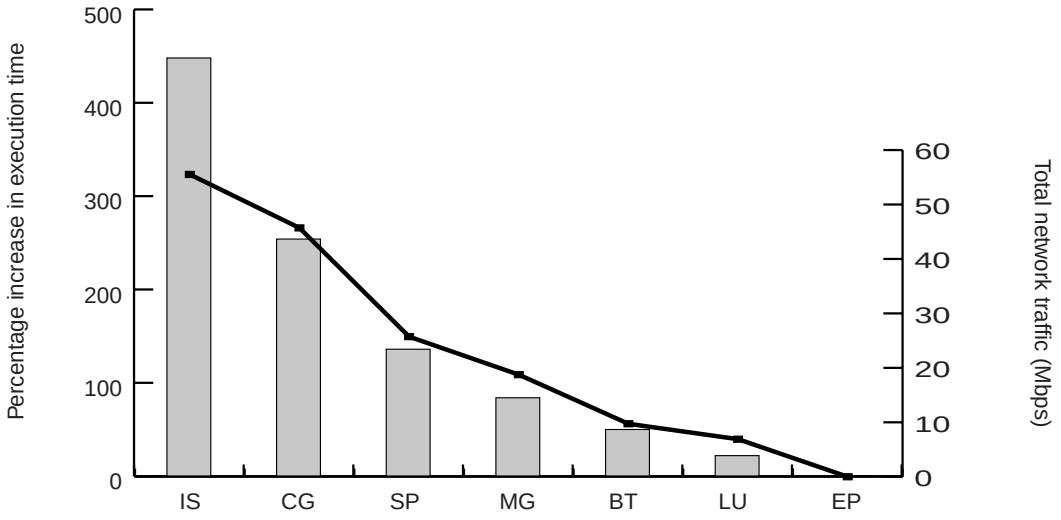

Figure 6: Execution time with the capacity of all communication links reduced to 10Mbps from 100 Mbps. Also shown is the total volume of network traffic generated by the application

surements are made at the system level and hence no instrumentation is necessary. This research also relates to task scheduling with CPU and communication constraints [3, 14] and performance prediction [6, 11].

5 Concluding remarks

We have measured the performance and system activity generated by the MPI implementations of NAS benchmarks in controlled environments and highlighted the key features that determine their performance on shared heterogeneous networks. In this process we have demonstrated that simple passive measurements during execution of an application on a testbed provide valuable information about the application structure and its performance in shared computing environments. In particular, we show that:

- CPU utilization and network bandwidth measurements allow us to characterize an application in terms of its communication and load distribution structure and suggest strategies for placement of application nodes on a busy network.
- CPU utilization measurements of an application on a testbed are a key indicator of application performance when some or all of the available nodes have competing jobs.
- Bandwidth utilized by an application on a testbed is a key indicator of application performance on a busy network. A higher bandwidth utilization is closely correlated to reduced performance when limited bandwidth is available.

We believe that this paper contributes to understanding how system level measurements can be used to characterize applications and estimate their execution time on shared networks. This approach is fairly general since it does not require access to the source code or knowledge of the code. The paper makes a fundamental contribution towards deriving models for predicting application performance in the presence of sharing of computation and network resources. However, more research is necessary before quantitative performance models for shared networks can be built automatically. Perhaps the most important problem that has to be solved is deriving the synchronization structure of a program and its implication on performance in this context.

This paper focuses on the methodology for building performance profiles for applications as a component of a resource selection on shared networks. A complete resource selection framework would also include a network measurement and prediction system and algorithms to map performance profiles to network resources based on current network conditions. Ongoing research is addressing the integration of these components.

6 Acknowledgments

This research was supported in part by the Los Alamos National Laboratory Computer Science Institute (LACSI) through LANL contract number 03891-99-23 as part of the prime contract (W-7405-ENG-36) between the DOE and the Regents of the University of California. Support was also provided by the Texas Advanced Technology Program under grant number 003652-0424 and the University of Houston's Texas Learning and Computation Center.

We also wish to thank other members of our research group, in particular Srikanth Goteti and Mala Ghanesh, for their contributions to this research.

References

[1] BAILEY, D., HARRIS, T., SAPHIR, W., VAN DER WIJNGAART, R., WOO, A., AND YARROW, M. The NAS Parallel Benchmarks 2.0. Tech. Rep. 95-020, NASA Ames Research Center, December 1995.

[2] BERMAN, F., WOLSKI, R., FIGUEIRA, S., SCHOPF, J., AND SHAO, G. Application-level scheduling on distributed heterogeneous networks. In *Proceedings of Supercomputing '96* (Pittsburgh, PA, November 1996).

[3] BHATT, P., PRASANNA, V., AND RAGHAVENDRA, C. Adaptive communication algorithms for distributed heterogrneous systems. In *Seventh IEEE Symposium on High-Performance Distributed Computing* (Chicago, IL, July 1998).

[4] BOLLIGER, J., AND GROSS, T. A framework-based approach to the development of network-aware applications. *IEEE Trans. Softw. Eng. 24*, 5 (May 1998), 376 – 390.

[5] DINDA, P. Statistical properties of host load in a distributed environment. In *Fourth Workshop on Languages, Compilers, and Run-time Systems for Scalable Computers* (Pittsburgh, PA, May 1998).

[6] FAHRINGER, T., BASKO, R., AND ZIMA, H. Automatic performance prediction to support parallelization of Fortran programs for massively parallel systems. In *Proceedings of the 1992 International Conference on Supercomputing* (Washington, DC, July 1992), pp. 347–56.

[7] FOSTER, I., AND KESSELMAN, K. Globus: A metacomputing infrastructure toolkit. *Journal of Supercomputer Applications 11*, 2 (1997), 115–128.

[8] GRIMSHAW, A., AND WULF, W. The Legion vision of a worldwide virtual computer. *Communications of the ACM 40*, 1 (January 1997).

[9] LITZKOW, M., LIVNY, M., AND MUTKA, M. Condor — A hunter of idle workstations. In *Proceedings of the Eighth Conference on Distributed Computing Systems* (San Jose, California, June 1988).

[10] LOWEKAMP, B., MILLER, N., SUTHERLAND, D., GROSS, T., STEENKISTE, P., AND SUBHLOK, J. A resource query interface for network-aware applications. In *Seventh IEEE Symposium on High-Performance Distributed Computing* (Chicago, IL, July 1998).

[11] SCHOPF, J., AND BERMAN, F. Performance prediction in production environments. In *12th International Parallel Processing Symposium* (Orlando, FL, April 1998), pp. 647–653.

[12] STEMM, M., SESHAN, S., AND KATZ, R. Spand: Shared passive network performance discovery. In *USENIX Symposium on Internet Technologies and Systems* (Monterey, CA, june 1997).

[13] SUBHLOK, J., LIEU, P., AND LOWEKAMP, B. Automatic node selection for high performance applications on networks. In *Proceedings of the Seventh ACM SIGPLAN Symposium on Principles and Practice of Parallel Programming* (Atlanta, GA, May 1999), pp. 163–172.

[14] SUBHLOK, J., AND VONDRAN, G. Optimal latency–throughput tradeoffs for data parallel pipelines. In *Eighth Annual ACM Symposium on Parallel Algorithms and Architectures* (Padua, Italy, June 1996), pp. 62–71.

[15] TABE, T., AND STOUT, Q. The use of the MPI communication library in the NAS Parallel Benchmark. Tech. Rep. CSE-TR-386-99, Department of Computer Science, University of Michigan, Nov 1999.

[16] TANGMUNARUNKIT, H., AND STEENKISTE, P. Network-aware distributed computing: A case study. In *Second Workshop on Runtime Systems for Parallel Programming (RTSPP)* (Orlando, March 1998).

[17] WEISMANN, J. Metascheduling: A scheduling model for metacomputing systems. In *Seventh IEEE Symposium on High-Performance Distributed Computing* (Chicago, IL, July 1998).

[18] WOLSKI, R., SPRING, N., AND PETERSON, C. Implementing a performance forecasting system for metacomputing: The Network Weather Service. In *Proceedings of Supercomputing '97* (San Jose, CA, Nov 1997).

Biographies

Jaspal Subhlok received a B.Tech. degree in Computer Science and Engineering from the Indian Institute of Technology, Kharagpur, India in 1984, and a Ph.D. degree in computer science from Rice University, Houston in 1990. Between 1990 and 1998 he served as a member of the research faculty in the School of Computer Science at Carnegie Mellon University, Pittsburgh. He is currently an Associate Professor of Computer Science at the University of Houston.

Dr Subhlok's technical areas of interest are compilers, tools and runtime systems, particularly in the context of parallel and distributed computing. His research involves design of algorithms and systems to solve a variety of problems in programming and runtime support for parallel and networked systems. The focus of his current research is on network-aware distributed computing and distributed web caching. His earlier projects included development and standardization of integrated task and data parallelism in the context of High Performance Fortran, algorithms and tools for automatic mapping of mixed task and data parallel programs, and validation of job scheduling strategies with supercomputer workloads.

Shreenivasa Venkataramaiah is an M.S. candidate in the Department of Computer Science at the University of Houston. His research interests include, distributed application characterization in shared heterogeneous network environments, analysis of application execution patterns by monitoring resource usage, variations in these patterns in different multi-processing environments, and development of software tools for such monitoring and analysis. He received his B.E. degree in Computer Science in 1997 from Bangalore University, Bangalore, India. He has also worked in the industry in the field of networking and embedded programming.

Amitoj Singh is an M.S candidate in the Department of Computer Science at the University of Houston. His current research focuses on characterizing parallel applications based on resource usage, benchmarking and evaluating parallel and distributed applications and developing software tools for analysis of high performance applications. He is also interested in real-time parallel computing. He received his B.S degree in 1996 in Computer Engineering from South Gujarat University, Surat, India.