Performance Surface Prediction for WAN-Based Clusters
†

Mark J. Clement, Glenn M. Judd, Joy L. Peterson,
Bryan S. Morse, J. Kelly Flanagan
Computer Science Department
Brigham Young University
Provo, Utah 84602-6576
(801) 378-7608
FAX (801) 378-7775
{clement,glenn,jpeterso,morse,kelly}@cs.byu.edu

Abstract

The last five years have been a period of exponential growth in the number of machines connected to the Internet and the speed at which these machines communicate. The infrastructure is now in place to consider a nationwide cluster of workstations as a viable parallel processing platform. In order to achieve acceptable performance on this kind of a machine, performance prediction tools must provide information on where to place computational objects. Incorrect object placement can result in poor performance and congestion in the network. This research develops a new paradigm for predicting performance in the Wide Area Network (WAN) based cluster arena. Statistical samples of the performance of clusters and applications are used to build characteristic surfaces. These surfaces are then used to provide guidance in placement of new applications. This prediction method is intended to minimize both the execution time of the application and the impact of the application on the nationwide virtual machine. Performance prediction tools are an important prerequisite to effectively utilizing WAN based clusters.

† This research was supported by Sprint Corporation.

1 Introduction

With Internet backbone performance increasing and the number of machines connected to the Internet growing exponentially, we see WAN based clusters as the next frontier for parallel computing. If all of the machines on the Internet were available for parallel tasks, it would appear as a large distributed memory machine with latency and bandwidth characteristics that vary according to geographical position and network connectivity. Most machines built five years from now will have four to eight processors connected through shared memory. A cluster of these machines connected through a high speed LAN provides additional processing power with more complex communication characteristics. It may be advantageous to utilize several clusters on a campus or in a city for certain applications. All of the machines in a state or nation may be effectively utilized by some parallel tasks. In order to manage this kind of a parallel platform, new paradigms must be developed to abstract away much of the complexity of these systems. Performance surfaces have been developed to combine factors contributing to parallel performance into a multidimensional surface which then can be manipulated to generate performance predictions. This paradigm provides the degree of abstraction necessary for this environment and appears to be accurate enough to make placement decisions for parallel applications.

1.1 LEGION

The LEGION system has been developed at the University of Virginia as a framework for implementing this kind of nationwide virtual machine [10]. This research uses an instrumented version of LEGION to gather performance data. LEGION provides a model of persistent objects that exist in a global name space across all machines in the cluster. A method can be called in another object through a directory of all objects that exist on the nationwide machine. A distributed file system is also provided by the system. The MENTAT [9] programming language is currently supported by LEGION. MENTAT provides the following support for parallelism:

- Dependency analysis to determine loops that can be distributed across multiple processors.
- Explicitly parallel loop constructs
- Non-blocking method invocation
- Futures

LEGION is similar to CORBA and DCE in terms of distributed computing capabilities, but is focused on providing high performance parallel capabilities as well. The combination of object oriented distributed computing capabilities and high performance parallel constructs makes LEGION attractive from a software engineering perspective.

The parallel support which LEGION offers has been shown to be usable by many parallel application developers [10]. Object placement is one of the big open problems in efficiently executing applications in this environment. LEGION offers limited support for object placement by allowing an application to communicate performance characteristics to the runtime system. A framework has been developed to allow the runtime system to query the hardware to determine placement for application objects. With the application and hardware characterizations, the runtime system should be able make good object mapping decisions.

The actual implementation of the object placement code in LEGION is quite simplistic. The user must make most of the decisions and provide most of the performance information himself. The performance surfaces generated by this research were designed to use the LEGION object placement framework to provide more efficient mappings of applications to subclusters in a nationwide network. The support is provided in an automated fashion so that little user intervention is necessary.

1.2 Performance Surfaces

In order to deal with the complexity of this problem, this research develops the *performance surface* paradigm for parallel architectures. This is an extension of prior work which was was focussed on characterizing caches[5].

Platform performance surfaces are probabilistic descriptions of the service times (e.g., latency) for a given system as a function of certain request parameters. In other words, "what is the probability that a request with these parameters will be serviced at (or within) time t?" Examination of these performance surfaces can reveal interactions between system components that might not be readily identified by the examination of individual components.

Application surfaces are likewise probabilistic descriptions of the service-time requirements (e.g., message granularity) for a set of request parameters. For example, "what is the probability that this system will issue requests having these request parameters and needing a response within time g?"

If the same set of request parameters is used to generate these surfaces, they can be compared to match the probability of each type of request with the probability that that request will be serviced within the time required. A high degree of correspondence between these surfaces suggests a high degree of efficiency on that system for that application; a low degree of correspondence suggests poor performance. This then gives us a means of characterizing or predicting the performance of the particular application on a given platform without actually running the application on that platform.

The LEGION runtime system contains an object placement framework for comparing known application characteristics with shared memory, LAN cluster, campus and WAN performance surfaces in order to determine where to place objects for optimum performance. These surfaces can also be time varying in order to account for changes in the utilization of workstations.

Workstation clusters will continue in their popularity as parallel platforms for several economic and technical reasons. This research will contribute to the success of WAN based cluster computing by providing automated object placement for parallel applications.

2 Related Work

Several different approaches have been taken in predicting performance for parallel systems. This research is unique in its usage of performance surfaces and the abstraction they provide. The performance surface paradigm allows the runtime system, operating system and hardware implementations to be represented by one data structure. Analytical models which attempt to analyze all of these systems are often too complex to be useful.

Several analysis techniques have been investigated which incorporate information from the source code or from an instrumentation run of a target application into an analytical model [1, 2, 4, 6, 8]. These methods combine effects from all basic blocks in an application into a total count of the number of critical operations performed. The cost of these operations is then estimated using a training set of benchmark programs, or by using specifications provided by system manufacturers. This research is similar in that it uses information from previous runs of the application in order to formulate a probabilistic surface. The data gathering requirements are much lower for surface analysis and the prediction techniques are much different as well.

Lost cycle analysis attempts to fit a curve to the number of communications and computations using multiple runs of an application [7]. Several runs are made to determine the coefficient for each variable in the model and the resultant equations are used to predict performance on scalable applications. If the system has a significant number of effects, this curve fitting can be inaccurate. The probabilistic methods used in this research can use results from several runs of an object to build a surface model for an application.

The HINT benchmark [15, 14] has been proposed as a way of generating information about CPU and memory performance on different platforms. The machine surface generated in this research is more focused on an analysis of the network, but uses some of the same benchmarking methods.

The PVM process migration strategy used for MpPVM[3] exploits WAN based clusters. The selection of a destination node for a migrating task can be difficult in a large WAN network. The performance surface analysis described here could also be used to make informed decisions for migration.

Theoretical work has been performed to develop efficient data distribution algorithms for distributed memory machines[11]. Graph partitioning algorithms have also been developed to assign tasks to processors based on thier communication patterns[12]. These methods require information which is not generally available from an application and are too computationally complex for scheduling tasks. Performance Surface analysis uses information which is easily obtainable and utilizes an abstraction which enables scheduling decisions to be efficiently.

3 Performance Surface Analysis

An ideal performance prediction function would return the service time for each possible service request. By multiplying the frequency of occurrence for each type of request by the corresponding service time, the overall performance of a system can be estimated. In a sense, this is analogous to the notion of a transfer function in signal processing: a function that indicates how specific components (in this case, service requests) "transfer" to the output (in this case, service time). However, such transfer functions are useful only for linear, time-invariant systems, and most computing systems are not such. The service time for a particular service request isn't a time-invariant function—factors such as caching can cause the service time to vary depending on the order of service requests. A complete analysis of these interactions is often infeasible.

One way to simplify this analysis is to present the system with a large, random set of service requests and measure the *distribution* of response times. That is, the system measures the probability of a particular service request being serviced within a particular time—a fuzzy estimation of the service time[1].

3.1 Formal Definitions

If the distribution of service times t is measured for various service requests that can be described by a set of n characteristics $\overline{r} = (r_1, r_2, \ldots, r_n)$, the following conditional probability is obtained:

$$\mathrm{Mach}(t, \overline{r}) = p(t|\overline{r}) \qquad (1)$$

These conditional probabilities can be viewed either as a set of one-dimensional probability distribution functions $p_{\overline{r}}(t)$ for each type of service request $\overline{r}$ or as

[1]The use here of the term "fuzzy" doesn't mean "approximate". It is meant in the same sense as fuzzy logic—a real-valued characteristic function ("probability") for a particular variable having a particular value.

an $n+1$-dimensional function of both $\overline{r}$ and t: $p(t, \overline{r})$. This latter conceptualization is what we call a *performance surface*.

Similar probabilistic measurements can be made of the frequency of each type of service request and the corresponding timing requirements or granularity g in a target application (what we call an *application surface*):

$$\mathrm{App}(g, \overline{r}) = p(g, \overline{r}) \tag{2}$$

The most critical requirement is whether requests are serviced at *or before* the granularity of the requests. In order to satisfy this requirement, the conditional probabilities in the performance surfaces are integrated to yield cumulative distribution functions for each request granularity g (in other words, "all requests serviced within time g or less"):

$$\mathrm{CPerf}(g, \overline{r}) = \int_0^g \mathrm{Mach}(t, \overline{r}) \; dt \tag{3}$$

The application surface $\mathrm{App}(g, \overline{r})$ is then multiplied by the cumulative performance surface $\mathrm{CPerf}(g, \overline{r})$ to give us the probability that a particular request $\overline{r}$ was issued with timing requirements g **and** was serviced within that time. We call this the *efficiency surface*:

$$\mathrm{Eff}(g, \overline{r}) = \mathrm{App}(g, \overline{r}) \;\; \mathrm{CPerf}(g, \overline{r}) \tag{4}$$

A metric for measuring the overall efficiency of a system is given by integrating the volume under the entire efficiency surface $\mathrm{Eff}(g, \overline{r})$:

$$\mathrm{TotalEff} = \int_{-\infty}^{\infty} \int_0^{\infty} \mathrm{Eff}(g, \overline{r}) \;\; dg \;\; d\overline{r} \tag{5}$$

4 Methods

The performance surface analysis system was implemented in order to validate the technique and to allow for data to be gathered. The machine surface was generated using a MENTAT application. Application surfaces were generated through running the application on an instrumented version of the MENTAT runtime system. The initial implementation focuses on network effects. Future work will extend the implementation to account for multiple architectural parameters.

4.1 Machine Surface Generation

The machine surface $Mach(t, \overline{r}) = p(t \mid \overline{r})$ is generated with a MENTAT application that causes various sizes of messages to be sent between nodes. The

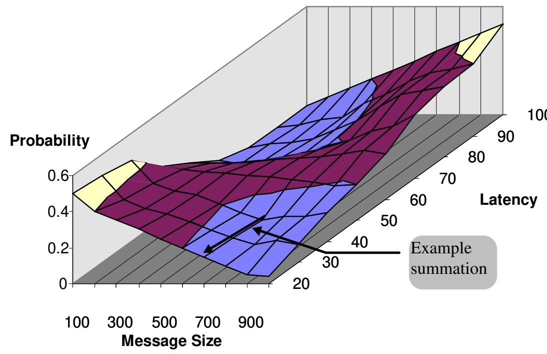

Figure 1: An example of a machine surface.

Pablo performance analysis system[13] gathers similar information and an attempt was made here to avoid perterbation caused by the instrumentation. Figure 1 provides an example of the shape a machine surface could have. Each point on the surface is the probability that a message of a particular size will take the given time to complete. The probabilities across the time axis for a given message size sum to 100%. The general shape of the surface should express that small messages will have a higher probability of taking a smaller amount of time. For large messages, the probability curve should be shifted to indicate the higher probability of a longer message latency.

Because the machine surface is generated using a MENTAT program, compiler, runtime system, operating system and hardware features are all included in the resultant surface representation. Each message size is a probability distribution function for the time taken for that size of message. The points along each of the grid lines in Figure 1 sum to 100%.

4.2 Application Surface Generation

The performance surface for the application $App(g, \overline{r}) = p(g, \overline{r})$ is generated by recording the time required to handle a remote procedure call for a given argument (message) size during a run of the application. The frequency and time for each message size are recorded in the application surface. The plot represents a two dimensional probability distribution function with all of the points on the surface summing to 100%. Figure 2 gives an example of what an application surface could look like. In this case, there are only a few message sizes that have non-zero usage frequencies. The slices of the surface for message sizes which have higher frequencies have higher magnitudes and

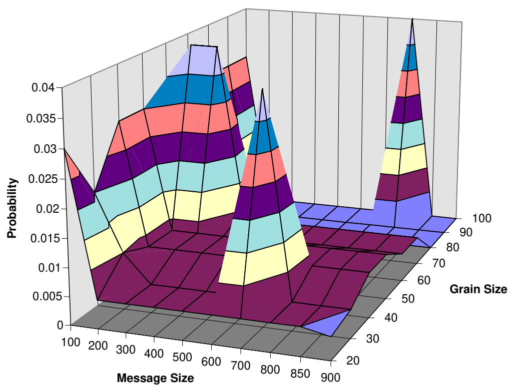

Figure 2: An example of an application surface.

will contribute more to the predicted efficiency than message sizes that are seldom used in the application.

Application data is gathered each time the application is run and combined with previous data to characterize the grain size of the application. Applications with shorter remote procedure call times (smaller grain size) will have a higher probability of requiring a low latency network connection for efficient execution. Given the machine surface, performance surface analysis can generate the probability that a particular message will complete before the grain size of the application has expired. If asynchronous message passing is used, then this corresponds to determining if the communication can be hidden behind an overlapped computation. As long as the communication completes before the computation has finished, there will be little message passing overhead.

Another way of looking at the application surface is to consider it's efficiency. If half of the time is spent in communications (the grain size for that application is equal to the message passing time), then the efficiency will be approximately 50%. If the communication completes in less than the grain size for that message size, then efficiency will be greater than 50%. By combining the application surface with machine surfaces for several candidate clusters, relative efficiency surfaces can be generated. These surfaces can be compared to determine the best placement for the application.

4.3 Relative Efficiency Surfaces

The relative efficiency surface $\mathrm{Eff}(g, \overline{r}) = p(g, \overline{r}) \int p(t \mid \overline{r}) dt$ represents the predicted efficiency of the application on a candidate cluster for different message sizes. By calculating the volume under the surface, a single comparative efficiency metric (TotalEff) can be generated for a cluster. This metric $\mathrm{TotalEff} = \int \int_0^\infty \mathrm{Eff}(g, \overline{r}) dg d\overline{r}$, which ranges between zero and one, can be compared with the metric for other clusters to select the best placement for an application. Each point on the relative efficiency surface is calculated by taking the magnitude at that same point on the application surface and multiplying that value by the sum of all probabilities in the machine surface (for that message size) with lower times. This corresponds to determining if the message will complete before the grain size of the application for that message size has finished.

For example, the application surface in Figure 2 has a peak at a message size of 600 bytes and a grain size of 40msec. In order to determine the value at that same message size and latency point for the relative efficiency surface, the probabilities along the line highlighted in the example summation in Figure 1 are summed and multiplied by the value in the application surface plot (.04).

5 Experimental Results

Several experiments have been performed to validate the performance surface paradigm. Most of the work to date has been performed on cluster of HP9000 workstations connected with 10Mbps Ethernet. Experiments were performed on a local cluster as well as a collection of machines in Utah and Oregon.

Figure 4 shows the machine surface for the Ethernet cluster. MENTAT objects were distributed on each of the processors. Remote procedure calls were made with different argument sizes to objects on different nodes in the MENTAT machine and the message sizes and the time to invoke the method were recorded. The two ridges in the plot are generated because of the non-homogeneous placement of the MENTAT objects. When two objects that reside on the same physical processor communicate, a lower transfer time is achieved. If the cluster included machines connected with ATM, fast Ethernet or other network technologies, the surface would have additional ridges. Figure 3 shows the raw data used to build the machine surface for the LAN. The ridge corresponding to other

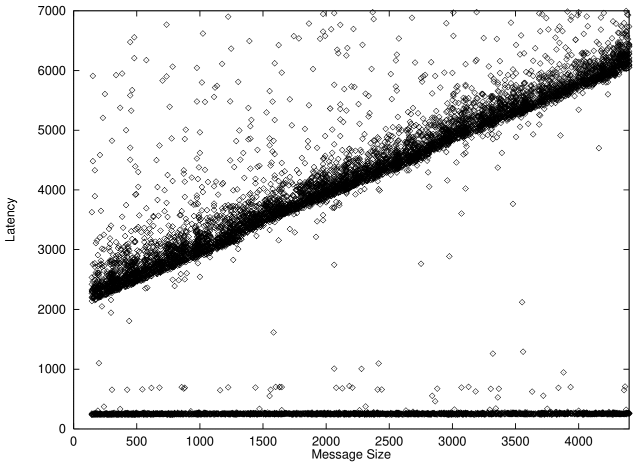

Figure 3: Raw data used to generate machine surface for the LAN. The latency is in μsec and the message size is given in bytes.

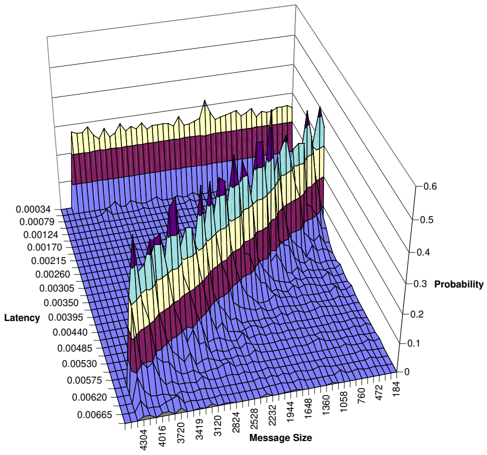

Figure 4: Machine surface for an ethernet cluster. The time scale is in msec and the message size is in bytes.

machines on the LAN can be used to calculate the message startup cost and bandwidth achieved. The latency is approximately $2msec$ and the bandwidth is nearly $8Mbps$.

Figure 6 shows the machine surface for the WAN cluster. The two ridges are much farther apart due to the difference in communication speeds between local and remote machines. There is also more variance due to contention with other internet applications. Figure 5 shows the raw data used to build the machine surface for the WAN. The latency term is approximately $60msec$ and the achieved bandwidth is $900Kbps$ over the T1 line.

Figure 8 shows an application surface for a gaussian elimination application. The small ridge near the message size axis corresponds to extremely small grain sizes for all message lengths, and will contribute to poor efficiency in the application. The programmer should make an effort to eliminate these fine grain size messages in order to improve performance. Application surfaces can assist programmers in tuning their code so that it is portable to several different clusters. Figure 7 contains the raw data used in preparing the surface. Gaussian elimination communicates using a wide range of message sizes due to the characteristics of the algorithm. Other algorithms, such as matrix multiplication will use only a few message sizes during their execution.

Figure 9 shows the efficiency surface for gaussian elimination on the local Ethernet cluster. The part of the ridge near the origin in the application surface was not reflected in the efficiency surface because it required lower latencies than were available on Ethernet. The efficiency metric TotalEff = 91% in this case indicates that the ethernet cluster will have good performance for this cluster. Figure 10 shows the efficiency surface for a WAN cluster consisting of machines in Oregon and Utah. The volume of the surface is significantly reduced by the decreased network speed. The efficiency metric TotalEff = 38% indicates that this application should be placed on the Ethernet cluster. By comparing efficiency metric values, correct object placement may be obtained. It is interesting to note that the execution time was approximately three times as long on the WAN cluster as it was on the local machines. This is corresponds to the ratio of the two TotalEff values. Future work will focus on building models which will allow execution time predictions to occur.

6 Future Direction

This paper describes some promising initial results and future research will investigate refinements to the general methodology. Many additional experiments will be performed to validate the accuracy of the modeling technique. The efficiency function will also be modified as additional dimensions are added to the surfaces. We have seen several interesting features in the machine surfaces, and we intend to analyze the runtime system and operating system in order to determine the cause of these anomalies.

In order for a nationwide virtual machine to be practical, care must be taken to place computations

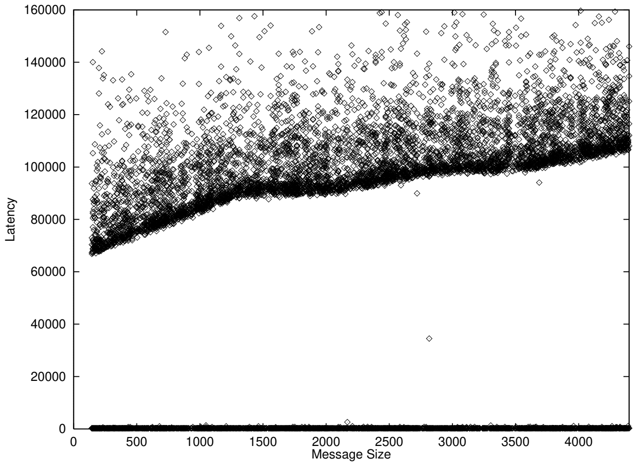

Figure 5: Raw data used to generate machine surface for the WAN. The latency is in μsec and the message size is given in bytes.

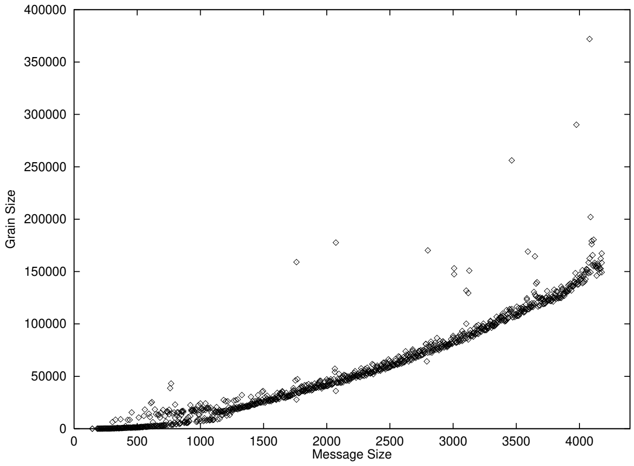

Figure 7: Raw data used to generate application surface.

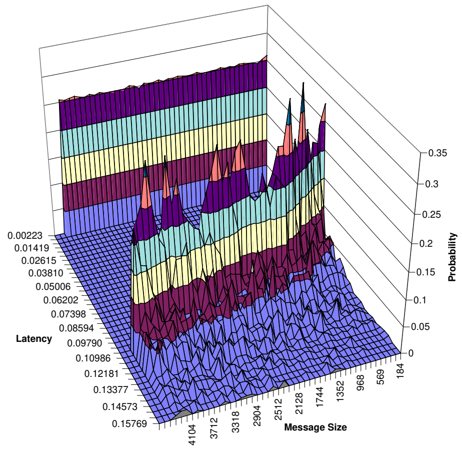

Figure 6: Machine surface for the WAN cluster including machines at BYU in Utah and at Oregon State University. The time scale is in msec and the message size is in bytes.

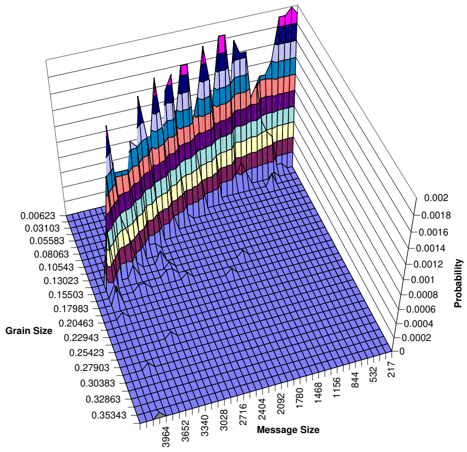

Figure 8: Application surface for Gaussian elimination. The time scale is in msec and the message size axis is in bytes.

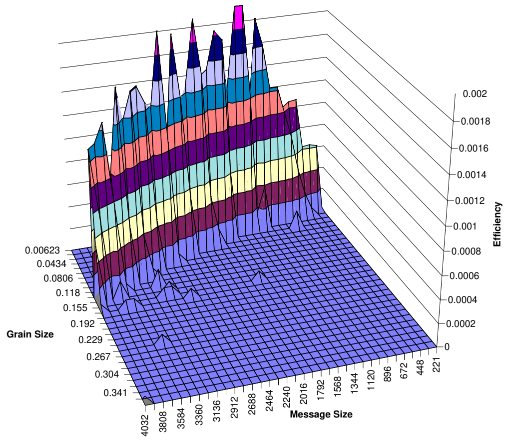

Figure 9: Efficiency surface for Gaussian elimination on the Ethernet cluster.

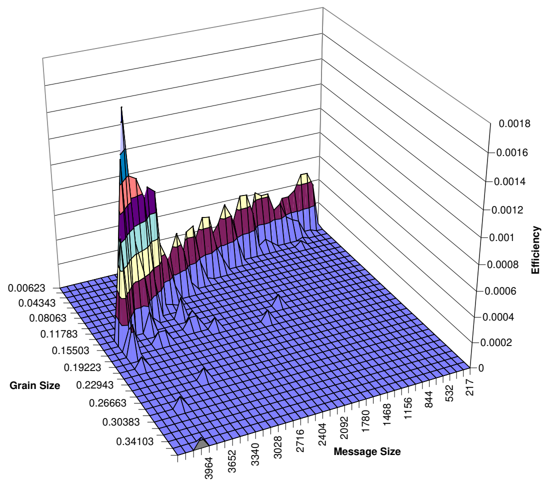

Figure 10: Efficiency surface for Gaussian elimination on a WAN cluster with approximately 1Mbps network connections.

so that they do not degrade overall system performance. Future work will focus on the effects of object placement on the performance of the machine as a whole. This analysis can also be performed using performance surfaces.

The logical performance surface is actually dynamically varying over time and future research will investigate ways of updating the machine surface periodically. This can be done through recomputing the surface occasionally, or by using the performance results from programs with known application surfaces to adjust the machine surface.

7 Conclusions

Parallel programming has always been viewed as a daunting project. Achieving acceptable performance can be difficult on even the simplest of machines. The complexities present in programming for a nationwide virtual machine are sufficient to overwhelm even the most sophisticated parallel programmer. The performance surface paradigm described here can abstract away much of the complexity found in this environment. The surfaces can be used to automatically place objects on clusters where they will run more efficiently. Anomalies in the surfaces can point out problems in the application, runtime system, or network. When the complexities of this computing surface have been dealt with, users can begin to use the untapped computing resources available in WAN based clusters.

8 Acknowledgment

We would like to thank Dr. Mike Quinn and the Parallel Processing group at Oregon State University for the use of their workstation cluster.

References

[1] M. Annaratone and R. Ruhl. Balancing interprocessor communication and computation on torus-connected multicomputers running compiler-parallelized code. In *Proceedings SHPCC 92*, pages 358–365, March 1992.

[2] V. Balasunderam, G. Fox, K. Kennedy, and U. Kremer. A static performance estimator

to guide data partitioning decisions. *SIGPLAN Notices*, 26(7):213 – 223, July 1991.

[3] K. Chanchio and X. H. Sun. Mppvm: A software system for non-dedicated heterogeneous computing. In *Proceedings of the International Conference on Parallel Processing*, 1996.

[4] M. J. Clement and M. J. Quinn. Analytical performance prediction on multicomputers. In *Proceedings of Supercomputing '93*, pages 886–905, November 1993.

[5] N. Crockett, X.-H. Tu, J. K. Flanagan, and F. Sorenson. A stochastic disk i/o simulation technique. In *Proceedings of the Winter Simulation Conference*, 1997.

[6] M. Crovella, R. Vianchini, T. LeBlanc, E. Markatos, and R. Wisniewski. Using communication-to-computation ratio in parallel program design and performance prediction. In *Proceedings of the Fourth IEEE Symposium on Parallel and Distributed Processing*, pages 238–245, December 1992.

[7] M. E. Crovella and T. J. LeBlanc. The search for lost cycles: A new approach to parallel program performance evaluation. In *Proceedings of Supercomputing '94*, November 1994.

[8] T. Fahringer and H. P. Zima. A static parameter based performance prediction tool for parallel programs. Technical Report ACPC/TR 93-1, University of Vienna Department of Computer Science, January 1993.

[9] A. S. Grimshaw, A. Ferrari, and E. West. *Parallel Programming Using C++*. The MIT Press, 1996.

[10] A. S. Grimshaw and W. A. Wulf. Legion – a view from 50,000 feet. In *Proceedings of the Fifth IEEE International Symposium on High Performance Distributed Computing*, 1996.

[11] P. Lee. Efficient algorithms for data distribution on distributed memory parallel computers. *IEEE Transactions on Parallel and Distributed Systems*, 8(8):825–839, August 1997.

[12] C.-W. Ou and S. Ranka. Parallel incremental graph partitioning. *IEEE Transactions on Parallel and Distributed Systems*, 8(8):884–896, August 1997.

[13] D. A. Reed, R. A. Aydt, R. J. Noe, P. C. Roth, K. A. Shields, B. Schwartz, and L. F. Tavera. Scalable performance analysis: The pable performance analysis environment. In *Proceedings of the Scalable Parallel Libraries Conference*, 1993.

[14] Q. O. Snell and J. L. Gustafson. Hint: A new way to measure computer performance. In *Proceedings of the 28th Hawaii International Conference on System Sciences, HICSS-28*, January 1995.

[15] Q. O. Snell and J. L. Gustafson. An analytical model of the hint performance metric. In *Proceedings of Supercomputing 1996*, November 1996.