A Slowdown Model for Applications Executing on Time-Shared Clusters of Workstations

Silvia M. Figueira and Francine Berman, *Member*, *IEEE*

Abstract—Distributed applications executing on clustered environments typically share resources (computers and network links) with other applications. In such systems, application execution may be retarded by the competition for these shared resources. In this paper, we define a model that calculates the slowdown imposed on applications in time-shared multi-user clusters. Our model focuses on three kinds of slowdown: *local slowdown*, which synthesizes the effect of contention for CPU in a single workstation; *communication slowdown*, which synthesizes the effect of contention for the workstations and network links on communication costs; and *aggregate slowdown*, which determines the effect of contention on a parallel task caused by other applications executing on the entire cluster, i.e., on the nodes used by the parallel application. We verify empirically that this model provides an accurate estimate of application performance for a set of compute-intensive parallel applications on different clusters with a variety of emulated loads.

Index Terms—Data-parallel applications, time-shared clusters of workstations, networks of workstations, application slowdown, performance prediction.

1 Introduction

In the last decade, networks of workstations have emerged as powerful platforms for executing parallel applications. To achieve performance, the allocation of application tasks to machines requires a realistic prediction of application behavior in the system. In time-shared systems, additional applications can cause *contention*, which dramatically affects the availability and capability of resources and can seriously retard application execution. If an effective predictive model can be developed for time-shared multi-user environments, applications can be allocated in a way that promotes execution performance.

To illustrate the effects of contention, consider an application with two tasks, A and B, mapped to a simple heterogeneous cluster with two machines, M_1 and M_2. Task A must execute before Task B, which depends on data generated by Task A. Tables 1 and 2 contain the times to execute each task of the application on each machine of the platform in dedicated mode as well as the times to transfer data between the machines (including data conversion) in dedicated mode when the tasks do not execute on the same machine. In this dedicated environment, both tasks should be assigned to machine M_1 and the application would execute in 16 time units.

However, if M_1 is time-shared and there are other applications competing for its CPU, the times in M_1's column of Table 1 will change. Table 3 illustrates the case in which the other applications are CPU-bound and slow Tasks A and B on M_1 by a factor of 3. In this setting, Task A should be assigned to machine M_2, whereas Task B should be assigned to machine M_1. Since communication is required, the application would execute in 38 time units, 10 units less than if the two tasks were executed on machine M_1.

In another setting, the additional applications on M_1 could be computing and also transferring data to M_2. Tables 3 and 4 illustrate the case in which both the computation on M_1 and the communication between M_1 and M_2 are slowed by a factor of 3. In this case, both tasks should be assigned to machine M_1 even though Task A executes faster on M_2. This is because the gain obtained by executing Task A on machine M_2 will be outweighed by the slowed communication between the machines.

This simple example illustrates both the importance of employing a performance-efficient mapping strategy and the fact that contention dramatically affects application performance and must be considered for efficient allocation.

Despite the fact that contention effects may dramatically influence the mapping of tasks to machines and, more importantly, affect the performance of applications, there has been little discussion on how to predict these effects. In recent literature, *machine workload* has been used to parameterize the distribution of tasks to workstations in a network. However, many allocation strategies do not consider load characteristics in the measurement of performance (e.g., [1], [2], [4], [6]). When load characteristics are considered, the system environment for each workstation may be severely limited ([12], [21]). We believe that a model to predict contention effects on application performance must reflect both system characteristics and workload behavior. The dependence on workload behavior reflects the fact that competing applications utilize many types of

- *S.M. Figueira is with the Department of Computer Engineering, Santa Clara University, Santa Clara, CA 95053-0566. E-mail: sfigueira@scu.edu.*
- *F. Berman is with the Department of Computer Science and Engineering, University of California, San Diego, La Jolla, CA 92093-0114. E-mail: berman@cs.ucsd.edu.*

Manuscript received 16 Feb. 1999; revised 17 Nov. 1999; accepted 29 June 2000.
For information on obtaining reprints of this article, please send e-mail to: tpds@computer.org, and reference IEEECS Log Number 109217.

TABLE 1
Dedicated Execution Times

	M_1	M_2
A	12	18
B	4	30

TABLE 2
Dedicated Communication Times

	$M_1 \rightarrow M_2$	$M_2 \rightarrow M_1$
$A \rightarrow B$	7	8

TABLE 3
Nondedicated Execution Times

	M_1	M_2
A	36	18
B	12	30

TABLE 4
Nondedicated Communication Times

	$M_1 \rightarrow M_2$	$M_2 \rightarrow M_1$
$A \rightarrow B$	21	24

resources, and the requirements for these resources vary during the execution of each application.

In [10], we proposed a contention model based on system characteristics and workload behavior for two-machine platforms. In this paper, we extend this work to develop a contention model for application performance on clusters of workstations. Our model measures three kinds of contention experienced by a single application: the *local slowdown*, which reflects the effect of contention for CPU in a single workstation; the *communication slowdown*, which reflects the effect of contention for the workstations and network links on communication costs; and the *aggregate slowdown*, which determines the effect of contention caused by other applications executing on the entire cluster. These measures, in the form of *contention factors*, can be used to adjust the computation time and communication cost predictions of an application to reflect the system load. The *local slowdown factor* is used to adjust the predicted time to execute a serial computation on a workstation. The *communication slowdown factor* is used to adjust the predicted time to communicate between two machines. Finally, the *aggregate slowdown factor* is used to adjust the predicted time to execute a single parallel application on a set of machines.

The adjusted predictions can be used to rank candidate schedules and allocate application tasks more effectively. The ranking and allocation tasks can be performed by automatic schedulers, which use performance prediction models, such as the one proposed, to find dynamically the best platform to execute an application.

Note that the local slowdown may be used in the calculation of the aggregate slowdown to provide a local slowdown factor for each workstation in the cluster. Also, the slowdown factors defined may be used together to predict the execution time of a targeted distributed application formed by parallel and/or sequential tasks that communicate, i.e., exchange data with one another.

This paper is organized as follows: Section 2 introduces the slowdown model. Section 3 determines the environment and presents the assumptions made, the experiments performed, and the platforms used. In Sections 4, 5, and 6, we define and demonstrate local slowdown factors, communication slowdown factors, and aggregate slowdown factors, respectively. Section 7 concludes with a summary and a discussion of future work.

2 The Slowdown Model

In a distributed heterogeneous system, application tasks may be assigned to distinct execution sites to promote concurrent execution and/or to take advantage of distributed data sources, aggregate memory, or I/O bandwidth. Scheduling decisions are based on two factors: The cost to execute application tasks on appropriate machines and the communication costs to transfer data between communicating tasks that do not execute on the same machine.[1] In order to define candidate schedules for an application, these costs must be estimated.

It is very common that users and/or application writers know the execution time of their applications when in dedicated mode. Experiments show that the time to execute an application in dedicated mode and the time to execute the same application under contention are directly proportional by a factor that we identify as the slowdown caused by contention for resources. Based on these facts, the contention models developed in the following sections provide slowdown factors, which determine, according to the load on the system, how much slower an application will compute or communicate. The slowdown factor can be multiplied by the execution time in dedicated mode to produce a realistic performance prediction. For example, if X_{ded} is the time to execute a task in dedicated mode, and X is the time to execute a task in a multi-user system, then we define *sd* to be the slowdown factor, where

$$X = X_{ded} \times sd. \tag{1}$$

In summary, slowdown factors can be used for predicting the performance of an application in a multi-user system, by helping adjust predictions in dedicated mode to accommodate for the load in the cluster. Such predictive models are critical for efficient scheduling [3].

Since slowdown factors depend on system characteristics and load behavior, they are always calculated at runtime. In our model, each slowdown factor reflects the current condition of the cluster and is recalculated when new applications arrive or leave. Since the slowdown factors are

1. In this paper, we assume that communication costs to transfer data between communicating tasks executing on the same machine are negligible.

always calculated at runtime, they must be efficient to compute relative to how often applications enter and leave the system.

3 The Environment

We define the *targeted application* as the application for which the slowdown is being calculated. Our targeted applications are large CPU-bound scientific applications. The computational environment is a cluster of workstations, shared by CPU-bound serial and/or parallel distributed applications, which execute for the entire duration of the targeted application. We define *competing applications* as the applications executing on the cluster and interfering with the targeted application. Each node in the cluster has one CPU. We assume that contention due to CPU-bound applications sharing resources provides the major source of slowdown. For this model, we consider the effects due to system overhead to be minimal. Developing an accurate contention model for CPU-bound applications provides a fundamental building block for contention models for a more heterogeneous job mix in which competing applications may also be non-CPU bound.

We assume that each workstation schedules its processes locally and independently using a round robin mechanism. Note that priority-based mechanisms, usually employed by workstations' operating systems, are reduced to round robin when the executing applications are CPU-bound ([8], [16]).

The models developed here do not take into account priorities, i.e., they assume that all applications have the same priority. There has been some work in the literature defining priority contracts for applications executing in internet clusters ([7], [15]). Preliminary experiments show that the models proposed can be easily extended to accommodate for different contracts. However, these results are beyond the scope of this paper.

We assume that the memory in each node fits the working set of all the applications executing on the node and that no delay is imposed by swapping. Our model can be extended to include more varied memory access costs in a straightforward manner.

We verify our models with experiments performed in two types of clusters: *dedicated-network clusters*, defined as homogeneous workstations interconnected by a fast dedicated LAN (such as the DEC Alpha Farm), and *nondedicated-network clusters*, defined as heterogeneous workstations interconnected by slower and possibly heterogeneous network links (such as a laboratory network). In a dedicated-network cluster, all the traffic in the network is generated by the machines in the cluster, while in a nondedicated-network cluster, the traffic in the communication links may have been generated by other machines as well. Also, machines in a nondedicated-network cluster may be located in different networks. The dedicated-network cluster used in our experiments was a DEC Alpha Farm located at the San Diego Supercomputer Center. This Alpha-Farm is a cluster of eight DEC Alpha 3000/400 workstations connected via a dedicated GIGAswitch. The nondedicated-network cluster used consisted of two DEC Alphas 3000/800 (located at the Computer Systems Laboratory at UCSD) connected by Ethernet and two IBM RS-6000s (located at the Parallel Computation Laboratory at UCSD) also connected by Ethernet. These workstations were part of two nondedicatednetworks.

Nondedicated-network clusters generally deliver less performance to the application than dedicated-network clusters. The basic differences between the two environments are:

- Less bandwidth is typically available in nondedicated-network clusters.
- In nondedicated-network clusters, machines may not be all equal.
- Links connecting the machines in a nondedicated-network cluster may not provide uniform bandwidth at any given time.
- Bandwidth between machines in a nondedicated-network cluster may be affected by applications executing on other machines that share network links with the machines in the cluster.

It is important to note that, in nondedicated-network clusters, we use the term *communication link* to refer to a connection between two nodes that may or may not be directly connected by a physical network link.

For this paper, experiments were performed to show the accuracy of the models proposed. The experiments were executed in the following steps:

1. Execute the targeted application in dedicated mode and measure the *dedicated time*.
2. Execute the targeted application under contention, i.e., together with a synthetically generated competing load, and obtain the *measured time*.
3. Compute the slowdown factor according to the competing load.
4. Multiply the dedicated time with the slowdown factor to obtain the *modeled time*.
5. Compare the modeled time with the measured time. The error will give an idea of the accuracy of the model.

We have verified the model with various load conditions. In the experiments performed, contention was imposed by a load generator that emulates competing applications. These applications execute in one or more machines and alternate computation and communication cycles. Note that the contention is the same throughout the execution of the targeted application. The emulator was parameterized so that the duration of the computation cycle, the direction of the communication, and the number of messages and message size in a communication burst could be varied. We ran experiments for a broad range of these parameters to measure the model's accuracy in different scenarios.

4 Local Slowdown

In this section, we present a contention measure, the local slowdown factor, which is used to adjust the computation time of a sequential computation to accommodate for the system load. The local slowdown factor reflects how much

slower the targeted application is going to execute on a workstation given the load from the competing applications. Local slowdown synthesizes the contention for CPU in the workstation and embeds computation and communication activity on that workstation.

For the local slowdown model, we consider each node in the cluster as an individual machine and calculate the slowdown on computation as sd_a, where a is a node in the cluster.

Since applications executing on a machine may be computing or communicating with other machines and experiments show that computation and communication activities impact the targeted application differently, the slowdown model needs to deal with these activities separately. The model also needs to reflect the number of computation and communication activities affecting the targeted application at the same time. This can be accomplished by using the relative amounts of computation and communication performed by each competing application (this is usually known or can be estimated with sensors) to calculate the probability that any combination of applications will be computing and/or communicating at the same time. For example, if there are two competing applications executing and one computes 60 percent of the time and communicates 40 percent of the time while the other computes 70 percent of the time and communicates 30 percent of the time, the probability that one computation activity interferes with the targeted application execution is $0.60 \times 0.30 + 0.70 \times 0.40$ while the probability that two computation activities interfere with the targeted application execution is 0.60×0.70. These probabilities are used in the model to represent the amounts of computation and communication of the competing applications.

The delay imposed by competing applications computing on a node reflects the fact that CPU time is evenly split among the competing processes. (Recall that we assume that the node scheduling policy is round robin.) This delay can be calculated based on the number of competing applications computing at the same time. Note that some factors, such as contention for cache, are not included in the model. Experiments show that, for the targeted applications which are CPU-bound applications that operate on large data sets (e.g., large matrices), these effects are not significant since the cache will be updated often, even when there is no contention. An extension of the model could include cache effects for applications that operate differently from the scientific ones.

Experiments show that the delay imposed by communication activity on computation time may vary with the average bandwidth that is available in the links used by the competing applications. Intuitively, if more bandwidth is available, more CPU is used by communicating competing applications and the slowdown imposed by the communication activity is larger. This is an important result and happens because the bandwidth available determines the amount of CPU used for communication by the competing communicating applications and, consequently, the delay imposed by the related communication activity.

To illustrate, let a and b be two nodes of the DEC Alpha Farm. Fig. 1 shows the difference between the delay imposed on computation on node a by one competing application communicating from node a to node b when different bandwidths are available. It shows curves for execution of an SOR benchmark with different problem sizes (given by $N \times N$) subject to no contention and subject to contention when the bandwidth available is 1.55 and 3.21 Mwords/second, respectively. The competing communicating application is the same for both experiments. It transfers data at different rates from node a to node b. It is clear that, with more bandwidth available, the communicating process uses more CPU, and the delay it imposes on the SOR benchmark is larger, i.e., the SOR takes longer to execute. The bandwidth available was calculated with a ping-pong benchmark that transfers 1K messages with 1K words each to another machine and receives one word back.

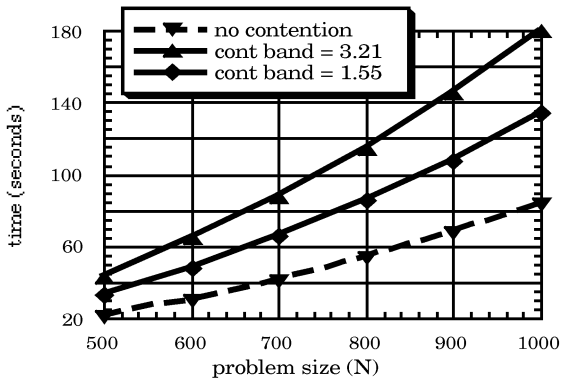

Fig. 1. Time to execute the SOR benchmark on one node of the DEC Alpha Farm in dedicated mode and competing for the CPU with one communication-bound process when different bandwidths are available.

The delay imposed by competing communicating applications depends also on a number of factors such as the communicating machines, the direction of the transferences, and the size of the messages being transferred. However, experiments show that all these factors are embedded in the bandwidth available between the communicating machines. Hence, we focus on just this bandwidth measure and we define one of the parameters in our model as the *available bandwidth* which is the amount of data (in Mwords/sec) that can be transferred between two machines given the load in the communicating links and on the machines themselves. The available bandwidth varies with the load in the system and can be measured with a benchmark that sends a number of messages from one machine to another and receives one word back. In some cases, the size of the messages used in this benchmark may affect the accuracy of the model, as we show later.

We calculate the slowdown imposed on a task executing on node a as:

$$sd_a = 1 + \sum_{i=1}^{p_a} (pp_{a,\,i} \times i) + \sum_{i=1}^{p_a} (pm_{a,\,i} \times delay_{comm}^{i,\,k}), \qquad (2)$$

where

- p_a = the number of additional applications executing on node a,
- $pp_{a,i}$ = the probability that i applications will compute on node a at the same time,
- $pm_{a,i}$ = the probability that i applications on node a will try to communicate at the same time, and
- $delay_{comm}^{i,k}$ = the delay imposed on computation by i communicating applications when the available bandwidth is k Mwords/second.

In the slowdown calculation, the first summation accounts for competing applications using CPU cycles. The second summation accounts for competing communicating applications. In the first summation, the probability $pp_{a,i}$ that each number of applications will be competing for the CPU at the same time is multiplied by the respective number of applications. This is intuitive, since CPU cycles are evenly split among all CPU-bound applications on the node. In the second summation, the probability $pm_{a,i}$ that each number of applications will be communicating at the same time is multiplied by the delay imposed by the respective number of competing applications communicating at the same time. This delay is a function of the available bandwidth, which is a key parameter of the model.

The values $pp_{a,i}$ and $pm_{a,i}$ are calculated based on the percentages of computation and communication associated with each competing application executing on node a. The slowdown factor is calculated at runtime and, therefore, it is important to guarantee that its calculation is efficient so that it does not impose too much overhead on a scheduler. Although the values $pp_{a,i}$ and $pm_{a,i}$ change with the load on the system and are obtained at runtime, they can be calculated using dynamic programming in time $O(p_a^2)$. Since p_a is generally small and the overall calculation of the slowdown takes $O(p_a^2)$ time, the overhead imposed by its calculation is negligible.

The value $delay_{comm}^{i,k}$ is the average delay imposed on computation by i communication-intensive competing applications executing on the node. It is calculated as the average delay imposed on a CPU-bound synthetic application by i contention generators that transfer messages to another node when the bandwidth is k Mwords/second. Since the delay imposed by receiving and sending applications is roughly the same, we use $delay_{comm}^{i,k}$ in both situations.

Experiments [9] determined $delay_{comm}^{i,k}$ as a set of functions of k, each of which is indexed by a different value of *i*. The curves can be determined just once for each platform, without imposing any additional overhead on the scheduling process at runtime. Each function is determined with a two-piece linear regression on the numbers obtained with a benchmark. This benchmark measures the delay imposed on a CPU-bound synthetic application by i communication-bound competing applications that communicate with other nodes under decreasing bandwidth. Note that the synthetic application used is independent of the targeted application. For these experiments, the application used was a sequential SOR. At runtime, the value for k—the average bandwidth on the links used by competing applications—can be predicted with statistical methods, e.g., using the Network Weather Service [19], [20], and used to determine $delay_{comm}^{i,k}$.

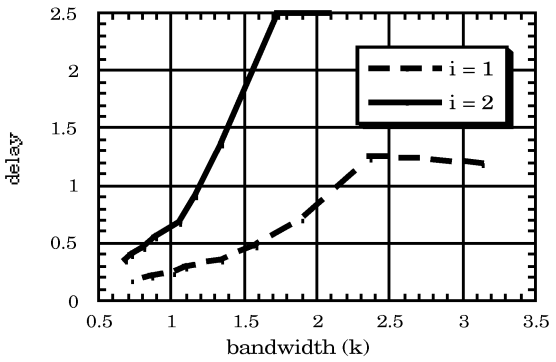

Fig. 2. Each curve in the graph provides the value for the respective delay, which varies with the bandwidth (k).

Fig. 2 shows the curves obtained for $delay_{comm}^{i,k}$ in the DEC Alpha Farm, when $i = 1$ and $i = 2$. In this case, $delay_{comm}^{i,k}$ is obtained by the following formulas, which were obtained with linear regression:

- For $i = 1$ and $k < 2.37$, $delay_{comm}^{i,k} = -0.20 + 0.49 \times k$.
- For $i = 1$ and $k \geq 2.37$, $delay_{comm}^{i,k} = 1.38 - 0.06 \times k$.
- For $i = 2$ and $k < 1.74$, $delay_{comm}^{i,k} = -0.50 + 1.37 \times k$.
- For $i = 2$ and $k \geq 1.74$, $delay_{comm}^{i,k} = 2.48$.

In low-bandwidth systems, $delay_{comm}^{i,k}$ may be approximated by just one curve or by just one value. In the first case, one curve is enough because the delay does not vary with i. In the second case, one value may be enough because the range of the delay imposed by communication is small.

To verify (2), we ran a comprehensive set of experiments, in which we varied all the components that could affect the accuracy of the model, such as platform, available bandwidth, number and characteristics of the competing applications (such as the amount of communication and computation), and burst size for the communicating applications.

In the following subsections, we show experiments that verify the model presented here for local slowdown, sd_a, for both dedicated-network clusters and nondedicated-network clusters. To validate the local slowdown models, we have used scientific computations on systems in which contention is generated by a variety of emulated loads. This guarantees equivalent conditions for comparative experiments. We demonstrate our model on serial versions of three benchmarks commonly used in scientific applications: An SOR benchmark [5], which solves Laplace's equation and was developed using PVM [17]; a Matrix Multiplication benchmark which was developed using KeLP [11] and MPI [13]; and a Multigrid benchmark [5], which was also developed using KeLP and MPI. We have verified the model with a broad range of load conditions, which were determined by the parameters for the contention generator.

4.1 Experiments with Dedicated-Network Clusters

To illustrate (2), Fig. 3 shows the modeled and measured times for executing the SOR benchmark on one node of

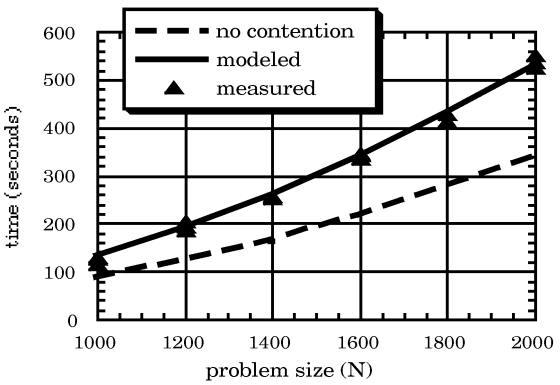

Fig. 3. Time to execute the SOR benchmark on one node of the DEC Alpha Farm in dedicated mode and competing with one application.

the DEC Alpha Farm, parameterized by problem size ($N \times N$). In this experiment, one competing application is executing on the same node. It alternates computation and communication cycles and communicates with another node 45 percent of the time. The available bandwidth, $k = 0.45$ Mwords/second, was measured with a benchmark and $delay_{comm}^{i,k}$ was calculated as shown in the previous section. In this case, $i = 1$, $k < 2.37$, and

$$delay_{comm}^{i,k} = -0.20 + 0.49 \times k = 0.02.$$

Using (2), the slowdown is calculated as:

$$1 + 0.55 \times 1 + 0.45 \times 0.02.$$

The slowdown, 1.56, is used to calculate the modeled time, which is obtained by multiplying the time to execute the targeted application in dedicated mode and the slowdown factor. Modeled time is compared with measured time to determine the accuracy of the predictions. In this example, our predictions were within an average error of 3 percent of the actual measurements.

To validate the model represented in (2), we ran a serial version of each benchmark with diverse configurations of one, two, and three instances of the contention generator. These configurations were defined by the contention generator parameters. Table 5 shows experiments with a serial version of the Multigrid application and two competing applications. Table 6 shows experiments with a serial version of the Matrix Multiply application. The targeted application executes on node a. The competing applications also execute on node a and alternate computation and communication with other applications executing on node b. The competing applications send a burst of messages, then compute for a while, then send another

TABLE 5
Experiments with Multigrid

Burst Sizes (number of messages / message size in words) per Competing Application	Available Bandwidth (Mwords / second)	Average Error
10 / 1000, 10 / 1000	2.75	2%
1000 / 1000, 1000 / 1000	2.75	1%
5000 / 1000, 5000 / 1000	2.75	1%
100 / 1000, 100 / 1000	2.12	11%
1000 / 1000, 1000 / 1000	2.12	9%
5000 / 1000, 5000 / 1000	2.12	4%

TABLE 6
Experiments with Matrix Multiply

Burst size(s) (number of messages / message size in words) per Competing Application	Average Bandwidth (Mwords / second)	Average error
4000 / 4000	1.51	5%
2000 / 2000, 2000 / 2000	3.18	1%
2000 / 2000, 3000 / 3000 4000 / 4000	3.89	9%
2000 / 1000, 2000 / 1000 2000 / 1000	2.45	6%

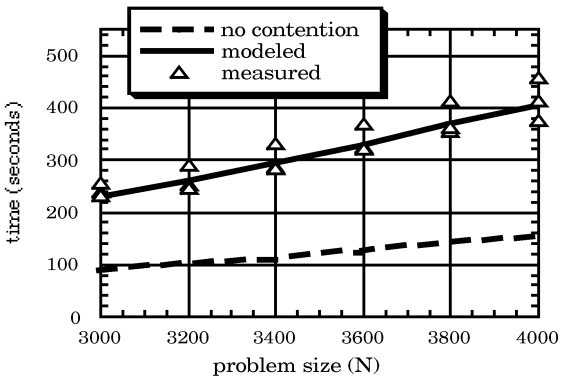

Fig. 4. Time to execute the SOR algorithm on a DEC Alpha, in dedicated and nondedicated mode, competing with two extra applications. The contending applications communicate 24 percent of the time and compute the remaining time.

burst of messages, and so on. Each row in the tables corresponds to one execution of the targeted application with five different problem sizes.

In these sets of experiments, we want to show that the available bandwidth is the correct measure to be used in this model. As the tables show, we vary the burst size used by the competing applications and the available bandwidth on the links used by the competing applications. We vary the available bandwidth by creating an extra load on node b. The available bandwidth was measured with the ping-pong benchmark and $delay_{comm}^{i,k}$ was calculated according to the curves shown in Fig. 2 (which were obtained using the SOR benchmark). In Table 5, we show experiments with three different sets of competing applications, each with two different available bandwidths. In Table 6, we show experiments with four different available bandwidths, each with a different set of competing applications. The competing applications are defined by the burst sizes. The average error reflects the difference between modeled and measured times for each set of experiments. The modeled times were calculated using (1) and (2). More information on the experiments can be found in [9].

4.2 Experiments with Nondedicated-Network Clusters

For nondedicated-network clusters, $delay_{comm}^{i,k}$ can be approximated by a constant and does not need to vary with i and k. This is because the bandwidth is lower and the range of the delay imposed by communication is small. We measured $delay_{comm}^{i,k} = 0.25$ for the nondedicated-network cluster, and used this value in the experiments presented in this section.

Fig. 4 shows the modeled and measured times for executing the SOR application on a DEC Alpha (in the nondedicated-network cluster) in dedicated and nondedicated modes, parameterized by problem size (which is $N \times N$). In this experiment, two more applications are executing on the machine. They alternate computation and communication cycles and communicate with another machine 24 percent of the time. In this platform, $delay_{comm}^{i,k} = 0.25$. Using (2), the slowdown is calculated as:

$$1 + 0.58 \times 2 + 0.36 \times 1 + 0.42 \times 0.25 = 2.6.$$

The slowdown, 2.6, is used to calculate the modeled time, which is obtained by multiplying the time to execute the targeted application in dedicated mode and the slowdown factor. Modeled time is compared with measured time to determine the accuracy of the predictions. In this example, our predictions were within an average error of 6 percent of the actual measurements.

To validate the model described in (2), we ran a serial version of the SOR application and other applications with diverse configurations of one, two, and three instances of the contention generator. These configurations were defined by the contention generator parameters. Table 7 presents experiments performed with a serial version of the SOR application when there are competing applications on the same node. Each row of the table corresponds to one execution of the benchmark (on a DEC Alpha) with six different problem sizes. As the table shows, we vary the computation/communication ratio of the competing applications and the burst size used by the competing applications. The available bandwidth varies according to the competing applications that are executing. The modeled time is compared with measured time to give the average error. The modeled time is calculated as the time to execute the SOR application, in dedicated time, times the slowdown, which was calculated according to (2). These experiments show that, in this platform, using $delay_{comm}^{i,k} = 0.25$ independently of the available bandwidth is enough to obtain reasonable accuracy.

4.3 Evaluation of the Local Slowdown Model

The model developed for slowdown on computation in (2) was based on knowledge of the relative amounts of computation and communication for each application. We assume that the time to execute the targeted application is longer than the computation/communication cycles of the competing applications and will be affected by both computation and communication activities. If the targeted application is fast in comparison with the duration of these cycles, i.e., its time to execute is close to the duration of one computation/communication cycle, the accuracy of the model decreases. The absolute error may also increase if the competing applications exhibit mutually nonrandom behavior, e.g., if the computation and communication cycles of the competing applications are either totally synchronized or totally nonsynchronized with one another.

Figs. 5 and 6 illustrate the difference in accuracy obtained in the prediction of the time to execute the SOR benchmark serially on a DEC Alpha (in the nondedicated-network cluster) in two settings. In both settings, the SOR competes for the CPU with one competing application that alternates computation and communication. In Fig. 5, the time to execute one computation/communication cycle of the competing application was 24.2 seconds. In this case, the time to execute the algorithm was longer than one computation/communication cycle of the competing application and the average error was 5 percent. In Fig. 6, the time to execute one computation/communication cycle of the competing

TABLE 7
Experiments with Computation

Amount(s) of Time Spent in Communication per Competing Application	Burst Size(s) (number of messages / message size in words) per Competing Application	Average Error
24%	1000 / 2000	5%
49%	2000 / 1000	9%
58%	1000 / 3000	9%
44%	5000 / 1000	7%
15%	5000 / 100	8%
14%	100 / 5000	3%
49%, 49%	2000 / 1000, 2000 / 1000	6%
15%, 14%	5000 / 100, 100 / 5000	8%
38%, 15%	1000 / 2000, 5000 / 100	13%
48%, 48%	1000 / 2000, 1000 / 2000	3%
48%, 21%	1000 / 2000, 5000 / 100	18%
58%, 32%	1000 / 3000, 1000 / 1000	15%
49%, 49%, 49%	2000 / 1000, 2000 / 1000, 2000 / 1000	6%
58%, 32%, 49%	1000 / 3000, 1000 / 1000, 2000 / 1000	22%

application was 288.0 seconds. In this case, the time to execute the same algorithm was shorter than one computation/communication cycle of the competing application, and the variation of the actual times to execute the algorithm causes the average error to expand to 17 percent. This indicates that an important next step in our research is an expansion of the program characteristics of both the targeted and competing applications.

5 Communication Slowdown

In this section, we present a contention measure, the *communication slowdown factor*, $sd_{a \to b}$, to adjust the communication costs between nodes a and b according to the system load. The value $sd_{a \to b}$ is a slowdown factor dependent on the load on machine a, the load on machine b, and the traffic on the communication link between the two machines.

There are accurate ways of measuring communication times when the communication pattern (number of

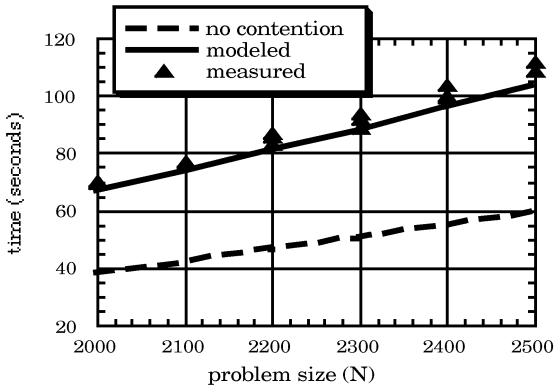

Fig. 5. Time to execute the SOR benchmark in dedicated mode and competing with one application that has a 24.2-second computation/communication cycle.

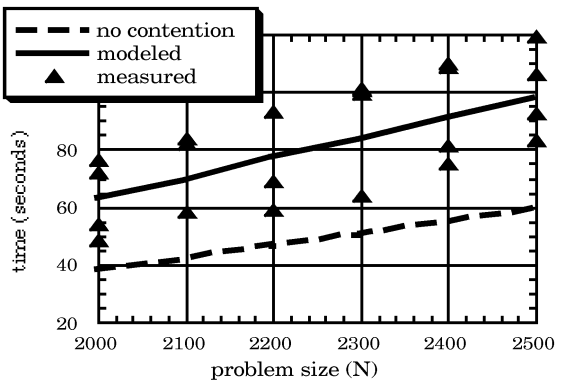

Fig. 6. Time to execute the SOR benchmark in dedicated mode and competing with one application that has a 288.0-second computation/communication cycle.

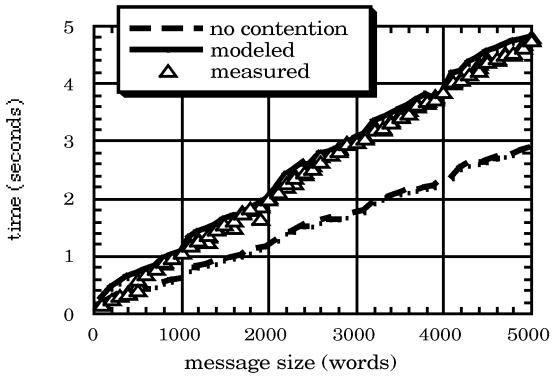

Fig. 7. Time to transfer bursts with 1,000 messages each between two nodes, in dedicated mode and under decreased bandwidth.

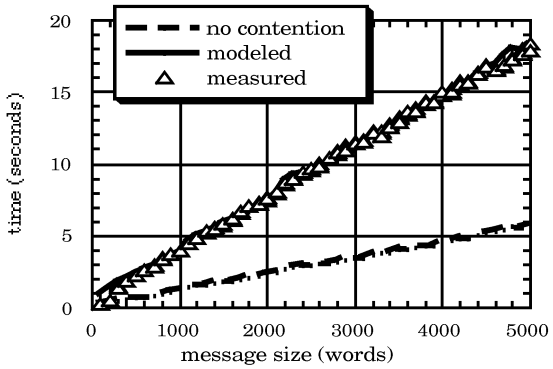

Fig. 8. Time to transfer bursts with 2,000 messages each between two nodes, in dedicated mode and under decreased bandwidth.

messages and message sizes) is known [10]. However, we need a model to describe how to calculate the time to communicate under contention when the only known information is the time to communicate in dedicated mode.

Our experiments indicate that various factors affect communication costs. The number of communicating and computing applications executing on the source and destination nodes, the size of the messages being transferred by the communicating applications, the available bandwidth on the links used, and the destination and direction of the communication all affect the amount of delay. However, experiments also show that the available bandwidth between the source and the destination nodes embeds all these other factors. Therefore, we use just the available bandwidth between nodes a and b to calculate the slowdown imposed on communication between the same nodes, as follows:

$$sd_{a\rightarrow b} = \frac{ib_{a\rightarrow b}}{cb_{a\rightarrow b}}, \tag{3}$$

where

- $ib_{a\rightarrow b}$ = the available bandwidth in dedicated mode and
- $cb_{a\rightarrow b}$= the available bandwidth at runtime.

Equation (3) reflects the delays imposed by both computation and communication activities on communication costs. Although these activities do not appear in the formula, they are embedded in the current bandwidth measure, $cb_{a\rightarrow b}$, which accounts for competition for both CPU and the shared communication link. The value for $ib_{a\rightarrow b}$ can be calculated with a benchmark beforehand. The benchmark sends 1,024 messages with 1,024 words each from one node to another and waits for the answer. The value for $cb_{a\rightarrow b}$ can be obtained at runtime with statistical methods employed by performance prediction tools, such as the Network Weather Service ([19], [20]). In our experiments, since the contention in the cluster is generated by a synthetic load, we determined the value for $cb_{a\rightarrow b}$ by measuring the bandwidth at runtime with the same benchmark used to calculate $ib_{a\rightarrow b}$.

To verify (3), we ran a comprehensive set of experiments, in which we varied all the components that could affect the accuracy of the model, such as platform, available bandwidth, number and characteristics of the competing applications, and burst and message sizes for the targeted and competing applications. Some of the experiments are presented below. Note that we ran experiments with different available bandwidths by adding extra competing applications to the system. After changing the bandwidth capacity, the available bandwidth is remeasured with the ping-pong benchmark.

5.1 Experiments with Dedicated-Network Clusters

Fig. 7 illustrates (3). It shows the time for transferring bursts of 1,000 equal-sized messages between two nodes of the DEC Alpha Farm in dedicated mode (available bandwidth = 6.21 Mwords/second) and the modeled and measured times for transferring the same messages between the two nodes when available bandwidth = 3.67 Mwords/second. The available bandwidth in dedicated mode and under contention were measured with a benchmark and used to calculate the slowdown factor according to (3). The slowdown factor is multiplied by the time in dedicated mode (which was measured) to produce the modeled time, which is then compared with the measured time. The error in this experiment was within 9 percent on average.

Fig. 8 also illustrates (3). It shows the time for transferring bursts of 2,000 equal-sized messages between two nodes of the DEC Alpha Farm in dedicated mode (available bandwidth = 6.21 Mwords/second) and the modeled and actual times for transferring the same messages between the two nodes when available bandwidth = 1.95 Mwords/second. The error in this experiment was within 13 percent on average.

To validate (3), we ran a targeted application (which transfers bursts of same-sized messages) with diverse configurations of one, two, and three instances of the contention generator, which represents competing applications. These competing applications affect the available bandwidth, which is measured with the ping-pong benchmark. Table 8 provides a representative set of the experiments performed to verify the equation. Each row of the table corresponds to three executions (one burst is transferred in each execution) of the targeted application for each message size, which grows in steps of 100 words. The bandwidth was altered by the extra load on the node and the available bandwidth was measured. The table shows the available bandwidth in each experiment and the burst size used by the targeted application. The average error reflects the difference between modeled and measured times.

TABLE 8
Experiments with Communication

Burst sizes (number of messages / message size in words)	Average Bandwidth (Mwords / second)	Average Error
100 / 100 – 1000	1.19	368%
100 / 100 – 1000	2.32	151%
100 / 100 – 1000	3.44	78%
100 / 1000 – 3000	1.19	16%
100 / 1000 – 3000	2.32	51%
100 / 1000 – 3000	3.44	23%
1000 / 100 – 1000	1.19	81%
1000 / 100 – 1000	2.32	42%
1000 / 100 – 1000	3.44	33%
1000 / 1000 – 3000	1.19	14%
1000 / 1000 – 3000	2.32	4%
1000 / 1000 – 3000	3.44	2%
5000 / 100 – 1000	1.19	57%
5000 / 100 – 1000	2.32	20%
5000 / 100 – 1000	3.44	16%
5000 / 1000 – 3000	1.19	10%
5000 / 1000 – 3000	2.32	3%
5000 / 1000 – 3000	3.44	2%

According to the results shown in Table 8, the absolute error increases when the burst size is small and the bandwidth is low. This is an important result and happens because, in this high-bandwidth platform, short bursts are not sensitive to bandwidth variations. This is clear in Table 9, which shows the slowdown

TABLE 9
Experiments with Bandwidth Benchmarks

Burst size (number of messages / message size in words) used by the Benchmark	$sd_{a\rightarrow b}$
100 / 100	1.08
100 / 500	0.96
1000 / 500	3.99
1024 / 1024	3.97
5000 / 500	3.96

$$sd_{a\rightarrow b} = (ib_{a\rightarrow b})/(cb_{a\rightarrow b})$$

produced with a ping-pong benchmark that measures the available bandwidth using a specific burst size. For small bursts (the first two lines), the slowdown is close to 1, meaning that there is no slowdown. However, for larger bursts, the slowdown is close to 4. Since the burst size used in the ping-pong benchmark reflects the slowdown produced, a tailored burst size should be used in the ping-pong benchmark in order to produce more accurate predictions.

5.2 Experiments with Nondedicated-Network Clusters

Fig. 9 also illustrates (3). It shows the time for transferring bursts of 1,000 equal-sized messages between the two IBM RS-6000s (located in the same subnet) in dedicated mode (the available bandwidth is 0.91 Mwords/second). It also shows modeled and actual times for transferring the same bursts, when there are competing applications alternating computation and communication cycles on the nodes, and the value for the available bandwidth is 0.33 Mwords/second. The available bandwidth in dedi-

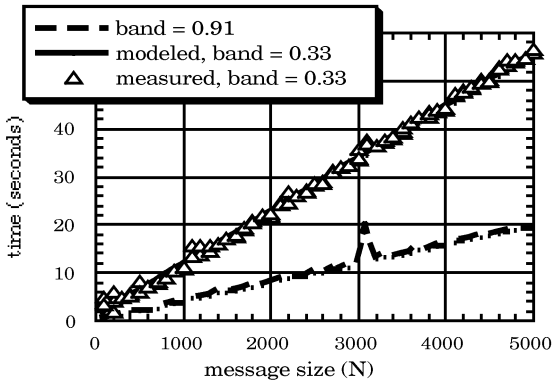

Fig. 9. Time to transfer bursts with 1,000 messages between machines in the same subnet.

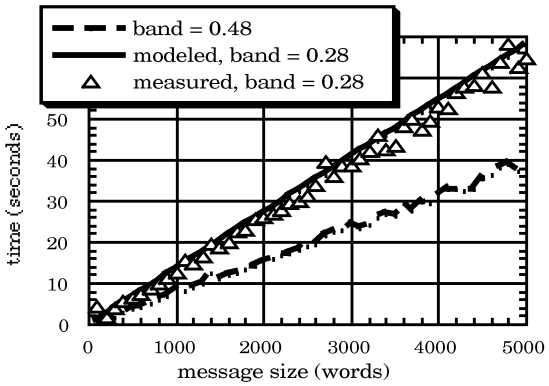

Fig. 10. Time to transfer bursts with 1,000 messages between machines in the different subnets.

cated mode and under contention was measured with a benchmark and used to calculate the slowdown factor ($sd_{a\to b} = 2.76$) according to (3). The slowdown factor is multiplied by the time in dedicated mode (which was measured) to produce the modeled time, which is then compared with the measured time. The average error for modeled versus actual communication cost was within 7 percent. The peak obtained by transferring messages with 3,100 words reflects an inefficiency of the IBM RS-6000 system in dealing with this specific message size.

Fig. 10 shows the time for transferring bursts of 2,000 equal-sized messages between two nodes in different subnets, when $cb_{a\to b} = ib_{a\to b} = 0.48$ Mwords/second, as well as the modeled and actual times for transferring the same bursts when there are additional applications alternating computation and communication cycles on the nodes. The value for $cb_{a\to b}$ is 0.28 Mwords/second, $sd_{a\to b} = 1.71$, and the average error for communication cost modeled with slowdown versus actual communication cost was within 7 percent.

To validate (3), we ran a targeted application (which transfers bursts of same-sized messages) with diverse configurations of one, two, and three instances of the contention generator which act as competing applications. These competing applications affect the available bandwidth, which is measured with a ping-pong benchmark. Table 10 provides a representative set of the experiments performed to verify the equation. Each row of the table corresponds to three executions (one burst is transferred in each execution) of the targeted application for each message size which grows in steps of 100 words. The bandwidth was

TABLE 10
Experiments with Communication

Burst Sizes (number of messages / message size in words)	Average Bandwidth (Mwords / second)	Average Error
10 / 100 – 1000	0.48	23%
10 / 100 – 1000	0.68	43%
10 / 1000 – 3000	0.48	5%
10 / 1000 – 3000	0.68	43%
1000 / 100 – 1000	0.48	19%
1000 / 100 – 1000	0.68	8%
1000 / 1000 – 3000	0.48	2%
1000 / 1000 – 3000	0.68	7%
5000 / 100 – 1000	0.48	15%
5000 / 100 – 1000	0.68	8%
5000 / 1000 – 3000	0.48	1%
5000 / 1000 – 3000	0.68	6%

TABLE 11
Experiments with Bandwidth Benchmarks

Burst size (number of messages / message size in words) used by the Benchmark	$sd_{a\to b}$
10 / 100	5.09
10 / 500	2.26
1000 / 500	2.06
5000 / 500	2.05

altered by extra load on the node and the available bandwidth was measured. The table shows the available bandwidth in each experiment and the burst size used by the targeted application. The average error reflects the difference between modeled and measured times.

Corroborating the results shown in Section 5.1, the absolute error increases when the burst size is small. The error is smaller in this platform than it is in the dedicated-network cluster because the bandwidth is lower and the variation in its range is shorter. Also, the transference of short bursts in this environment is more sensitive to bandwidth variation than the transference of large bursts. This is clear in Table 11, which shows the slowdown $sd_{a\to b} = (ib_{a\to b})/(cb_{a\to b})$ produced with a benchmark that measures the bandwidth using a specific burst size. As for dedicated-network clusters, a benchmark that calculates the bandwidth using a tailored burst size should overcome this problem.

6 Aggregate Slowdown

Local slowdown and communication slowdown provide a measure of the impact of contention on single processors and on the network. However, these slowdown factors are not enough to predict the slowdown imposed on a parallel application executed in a time-shared cluster. In this case, the load in the entire cluster must be combined into one single slowdown factor.

In this section, we will define a contention measure, the aggregate slowdown factor, that determines how the load in the cluster will retard the performance of an individual parallel application. The aggregate slowdown factor is used to adjust the time to execute all the components of a parallel computation. Using this factor, the time to execute a parallel application X on a cluster is given by:

$$X = X_{ded} \times sd, \tag{4}$$

where *sd* is the aggregate slowdown factor (dependent on the aggregate load in the cluster), and X_{ded} is the time to execute the targeted application X on the cluster without interference from competing applications.

Aggregate slowdown factors depend on both the work partitioning policy and the capacity of each node. Node capacity is given by two parameters, each determined for each node in the cluster:

- sd_a = the local slowdown at node a and
- w_a = the weight of node a.

The *local slowdown* at node a, sd_a, was discussed in Section 4. The *weight* of a node a, w_a, reflects its dedicated capacity relative to the other nodes in the cluster. The value for w_a can be calculated as the ratio of the time to execute a task on the slowest node s to the time to execute the same task on node a (as described in [7]). In a homogeneous cluster, $w_a = 1$, for all a. To calculate weights for nodes in a heterogeneous cluster, we execute a serial benchmark in dedicated mode on the different nodes of the cluster and calculate the weight for each node $a \neq s$ as $w_a = t_s/t_a$, such that the machine with the longest time t_s, has weight $w_s = 1$. Note that weights are dependent on the benchmark chosen ([7], [9]).

The following subsections develop aggregate slowdown factors for two work partitioning policies commonly used by high performance parallel applications:

- *load-dependent partitioning*, in which the amount of work allocated to each node is calculated based on its computational capacity, and
- *constraint-based partitioning*, in which work is partitioned among the nodes according to a set of constraints (e.g., memory availability).

Note that we base our model on work partitioning (and not on data partitioning) since the computational effort required by a node is not always proportional to the amount of data the node is assigned. Note also that the model developed does not cover applications for which work partitions vary throughout the execution (e.g., particle simulation). The model does cover these applications when a dynamic rebalance strategy, which takes the load of each node into account, is implemented *as part of the code*.

6.1 Aggregate Slowdown for Load-Dependent Partitioning

In load-dependent partitioning, work is allocated according to the available computational capacity in the cluster. In this case, work is partitioned so that all nodes will finish together (ideally), assuming they all started at the same time. For this partitioning, we determine the aggregate slowdown in terms of the *aggregate capacity* available on the cluster. We define aggregate capacity as the sum of the available computational capacities of the nodes. We define $capacity_a$ as the available computational capacity of node a. It is important to note that the available capacity of each node depends on both its local load and its weight. For instance, for an unloaded node k, $capacity_k = w_k$, and for a loaded node k, $capacity_k = w_k \,/\, sd_k$.

The aggregate slowdown is given by the ratio of the aggregate capacity available in the unloaded (or dedicated) cluster, to the aggregate capacity available in the loaded cluster. We calculate the aggregate slowdown for a cluster formed by n nodes as

$$sd = \frac{aggregate\ capacity_{ded}}{aggregate\ capacity} = \left(\sum_{a=1}^{n} w_a\right) \Big/ \left(\sum_{a=1}^{n} \frac{w_a}{sd_a}\right). \tag{5}$$

Fig. 11 illustrates a load-dependent partitioning. It shows Application A's work partitioning in a four-node heterogeneous cluster. The application has 7 work units which are partitioned among the nodes according to their capacities.

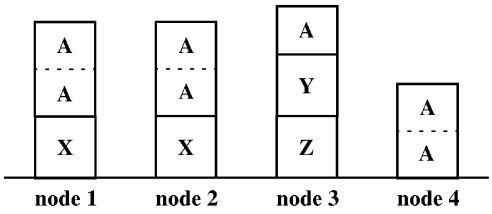

Fig. 11. A cluster formed by 4 heterogeneous nodes and shared by four applications: A, X, Y, and Z. Application A's work is divided among the nodes according to their computational capacity. Each box represents the amount of work associated with the corresponding node.

The dotted lines in the figure determine Application A's work units. If node 4 is twice as slow as the others, then

$$sd = (2 + 2 + 2 + 1)/(2/2 + 2/2 + 2/3 + 1/1) = 7/3.67 = 1.9$$

according to (5). For this example, we assume that $sd_a = 1 +$ the number of applications executing in node a.

6.2 Aggregate Slowdown for Constraint-Based Partitioning

When work is partitioned based on a set of constraints, such as memory availability or data locality, the nodes may not finish at the same time, as with the load-dependent partitioning. In this case, the aggregate slowdown can be calculated based on the slowest node which is determined by the capacity of each node and by the extra amount of work assigned to each node. The extra amount of work is calculated relative to the work assigned in a *uniform* partitioning, in which the work is partitioned evenly among the nodes, as shown in Fig. 12.

The time to execute the part of the application assigned to each node is formed by two components: The amount of work that would be performed if the work partitioning were uniform and the extra work that must be executed by a node because the work partitioning is not uniform. Using this approach, the time to execute the part of the application assigned to node a in the presence of contention is

$$\begin{aligned} time_a &= (time_{a,\,ded} \times sd_a) + (time_{a,\,ded} \times ew_a \times sd_a) \\ &= time_{a,\,ded} \times sd_a \times (1 + ew_a), \end{aligned} \tag{6}$$

where

- $time_{a,\,ded}$ = the time to execute the part of the application assigned to node a, in dedicated mode,
- ew_a = the extra work executed by node a, calculated as

$$ew_a = \frac{f_a - (1/n)}{1/n} = (f_a \times n) - 1, \tag{7}$$

- f_a = the fraction of work assigned to node a, and
- n is the number of nodes in the cluster, i.e., the number of nodes used by the targeted application.

Note that node a can be assigned less work than it would be if the partitioning were uniform. In this case, $f_a < 1/n$ and $ew_a < 0$.

If all nodes begin execution concurrently, the slowest node will determine the time to execute the application

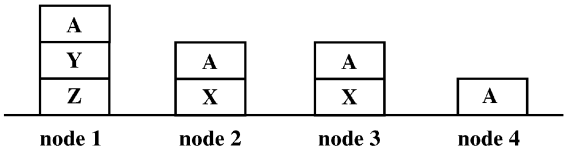

Fig. 12. A cluster formed by 4 nodes and shared by four applications: A, X, Y, and Z. Application A's 4 units of work are divided evenly among the nodes. Each box represents the amount of work associated with the corresponding node.

which is given by (4). From (4) and (6), we can calculate the aggregate slowdown as:

$$sd = \frac{X}{X_{ded}} = \max_a\left\{\frac{(1 + ew_a) \times sd_a}{w_a}\right\} / \max_a\left\{\frac{(1 + ew_a)}{w_a}\right\}, \tag{8}$$

where the numerator represents the time units to execute the application in the presence of contention and the denominator represents the time units to execute the application in dedicated mode. Note that the uniform partitioning policy is characterized by $ew_a = 0$ for all a, and the aggregate slowdown for this policy can also be calculated by (8).

Equation (8) assumes that the targeted application will use the same constraint-based work partitioning (given by ew_a) in determining both X and X_{ded}. This may not be always the case, since the work partitionings used for X and X_{ded} may be determined by different constraints. For example, if the partitioning depends on the memory available on each workstation, which varies according to the load, the partitioning used for the dedicated and nondedicated executions may not be the same. Therefore, we must allow for different constraint-based partitionings to be used for X and X_{ded}. Call these constraint-based partitionings d_1 and d_2. To accommodate for d_1 (for X) and d_2 (for X_{ded}), (8) is modified to

$$sd = \max_a\left\{\frac{(1 + ew_{a,\,1}) \times sd_a}{w_a}\right\} / \max_a\left\{\frac{(1 + ew_{a,\,2})}{w_a}\right\}, \tag{9}$$

where

- $ew_{a,\,y}$ = extra work executed by node a with partitioning d_y ($y = 1, 2$), calculated as

$$ew_{a,\,y} = \frac{f_{a,\,y} - (1/n)}{1/n} = (f_{a,\,y} \times n) - 1 \tag{10}$$

  and
- $f_{a,y}$ = the fraction of work assigned to node a for partitioning d_y ($y = 1, 2$).

In the setting, where dedicated time X_{ded} is given for a uniform work partitioning, (8) can be reduced to

$$sd = \max_a\left\{\frac{(1 + ew_a) \times sd_a}{w_a}\right\} / \max_a\{1/w_a\}, \tag{11}$$

where the numerator represents the time to execute in the presence of contention with a constraint-based partitioning (characterized by ew_a) and the denominator represents the time to execute in dedicated mode with a uniform partitioning.

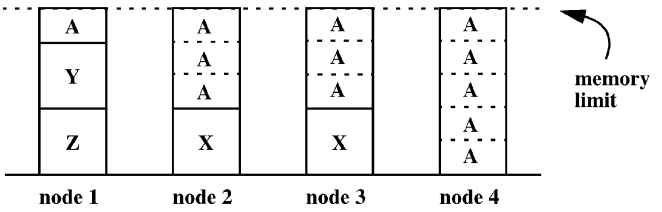

Fig. 13. A cluster formed by 4 nodes and shared by four applications: A, X, Y, and Z. Application A's 12 units of work (which are determined by the dotted lines) are divided among the nodes according to memory availability. Each box represents the amount of work associated with the corresponding node.

Since $\max_a\{1/w_a\} = 1$, for any node a, (11) can be further reduced to

$$sd = \max_a\left\{\frac{(1 + ew_a) \times sd_a}{w_a}\right\}. \quad (12)$$

Fig. 12 illustrates the uniform partitioning. It shows Application A's work partitioning in a four-node cluster. The application has four work units which are partitioned evenly among the nodes. According to the figure, $f_1 = 1/4$, $f_2 = 1/4$, $f_3 = 1/4$, and $f_4 = 1/4$. If the nodes are homogeneous, node 1 determines the time to execute Application A. In this case, according to (12),

$$sd = (1 + 0) \times 3/1 = 3.$$

If the nodes are heterogeneous and node 2 is four times as slow as the others, the time to execute Application A is determined by node 2 and $sd = (1 + 0) \times 2/1 = 2$, also according to (12).

Fig. 13 illustrates the constraint-based partitioning. It shows Application A's work partitioning in a four-node cluster. The application has 12 work units which are partitioned among the nodes according to memory availability. The dotted lines in the figure determine Application A's work units. According to the figure, $f_1 = 1/12$, $f_2 = 3/12$, $f_3 = 3/12$, and $f_4 = 5/12$. If the nodes are homogeneous, nodes 2 and 3 determine the time to execute Application A. In this case, according to (12), $sd = (1 + 0) \times 2/1 = 2$. However, if the nodes are heterogeneous and node 4 is twice as slow as the others, the time to execute Application A is determined by node 4, and $sd = (1 + 0.67) \times 1/1 = 1.67$, also according to (12).

6.3 Experimental Verification

The formulas presented in the previous section provide a measure of the aggregate slowdown. To assess the accuracy of these formulas, we performed a large collection of experiments on a wide range of CPU-bound benchmarks commonly found in high performance scientific applications. The applications used included: Jacobi2D [5], Jacobi3D [5], Red-Black SOR [5], Multigrid [5], Genetic Algorithm [18], and LU Solver [14]. In these experiments, we compared actual execution times with modeled times (dedicated time factored by aggregate slowdown) for applications executing on a cluster of workstations with synthetic loads. The synthetic loads were formed by a combination of serial and/or CPU-bound parallel applications distributed over the cluster. Note that the load does not change during each experiment. In this paper, we show representative experiments for load-dependent and constraint-based scheduling policies. A more complete list of the experiments can be found in [9].

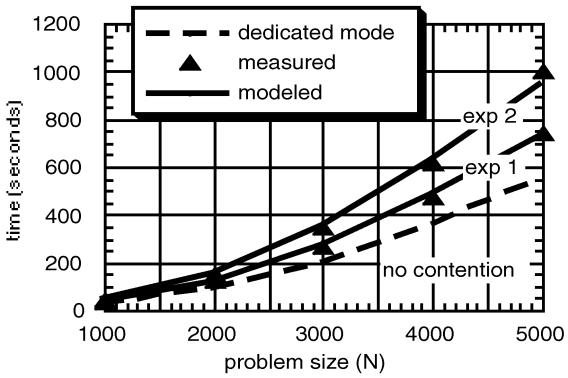

Fig. 14. Time to execute the SOR algorithm on 4 nodes of the DEC Alpha Farm in dedicated mode and with 2 different loads.

The local slowdown for each machine is determined by the synthetic competing load. The weights depend on the targeted application and are calculated for each experiment. These parameters are used in the proper equation to produce the aggregate slowdown factor. This slowdown factor is multiplied by the time to execute in dedicated mode (which was measured) to produce the modeled time. The error is obtained by comparing modeled and measured times.

All of our experiments show the modeled execution times to be within 15 percent of actual execution times. This demonstrates that, for reasonably accurate dedicated time performance estimates, aggregate slowdown captures contention delays in production systems accurately and can provide an important factor for performance predictions. The discrepancy in absolute error between modeled and actual times is due to a variety of factors. For example, the fact that the round robin scheduling policy assumed for each workstation is not a "perfect" round robin contributes to the error.

The following sections show a representative subset of the experiments performed on the dedicated-network and nondedicated-network clusters described above.

6.3.1 Experiments with Dedicated-Network Clusters

Consider a platform where the nodes are homogeneous and all the weights are equal to 1. Fig. 14 illustrates a representative experiment with the load-dependent work partitioning policy. Shown is a distributed SOR application [5] developed using PVM [17] and executed on 4 nodes of the DEC Alpha Farm with two different loads. In experiment 1 (*exp 1*), there is a CPU-bound parallel application executing on two of the nodes. In this case,

$$aggregate\ capacity = 1/2 + 1/2 + 1 + 1 = 3$$

and

$$sd = 4/3 = 1.33.$$

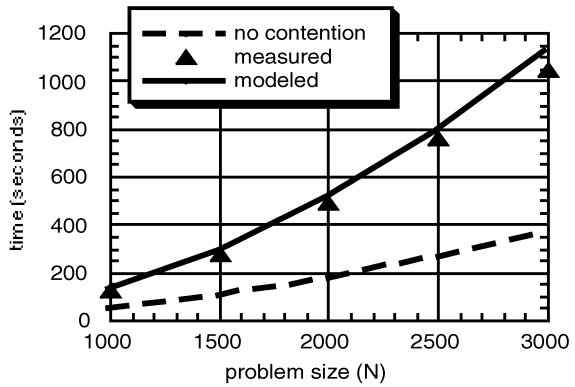

Fig. 15. Time to execute the Multigrid application on 4 nodes of the DEC Alpha Farm, with constraint-based partitioning, in dedicated mode and with contention on the nodes.

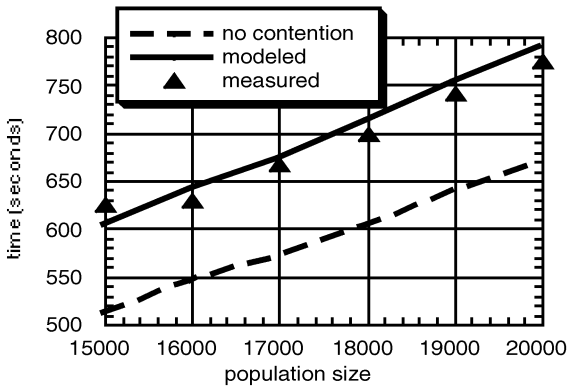

Fig. 16. Times for the Genetic Algorithm executing with load-dependent work partitioning on a nondedicated-network cluster in dedicated mode and under contention.

In experiment 2 (*exp 2*), there is a CPU-bound parallel application executing on two of the nodes, and two serial CPU-bound applications executing on another node, as shown in Fig. 11. In this case,

$$aggregate\ capacity = 1/2 + 1/2 + 1/3 + 1 = 2.33,$$

and the SOR algorithm was slowed by a factor of $4/2.33 = 1.72$.

Fig. 15 represents experiments using the constraint-based work partitioning policy. Shown is a Multigrid application [5] (developed using KeLP [11] and MPI [13]) executed for different problem sizes (given by $N \times N$) on 4 nodes of the DEC Alpha Farm. Two of the nodes (nodes 2 and 3) also host a CPU-bound parallel application, and one of the nodes (node 1) also hosts two serial, CPU-bound applications, as shown in Fig. 13. The blocks of data were divided among the nodes according to a set of constraints resulting in the partitioning shown in Table 12 (column *f*). Table 12 also shows the other parameters (*sd* and *ew*) used to calculate the aggregate slowdown (3.0).

6.3.2 Experiments with Nondedicated-Network Clusters

Fig. 16 represents experiments using a load-dependent work partitioning policy with a Genetic Algorithm application [18] developed using PVM [17]. Shown are modeled and measured times for execution with different problem sizes (given by population size) on four nodes of a nondedicated-network clusters consisting of two DEC Alphas and two IBM RS-6000s. In this experiment, the IBM RS-6000s are also executing a CPU-bound parallel application. The aggregate slowdown is 1.18.

Fig. 17 represents experiments using a constraint-based work partitioning policy. Shown is a representative SOR application [5] developed using PVM [17] and executed for different problem sizes (given by $N \times N$). The platform is a nondedicated-network cluster consisting of two DEC Alphas and two IBM RS-6000s. Data for the dedicated run was partitioned according to a set of constraints resulting in the partitioning shown in Table 13 (column *f*).

In experiment 1 (*exp 1*) there was an additional CPU-bound parallel application executing on the two IBM RS-6000s. Table 14 shows the work partitioning (column *f*) and slowdown (sd) parameters used in this experiment. According to Tables 13 and 14, the aggregate slowdown for this case is $2.67/1.33 = 2.01$.

In experiment 2 (*exp 2*) there was a CPU-bound parallel application executing on the two IBM RS-6000s, another executing on one IBM RS-6000 and one DEC Alpha, and three more CPU-bound serial applications executing on the other Alpha. Table 15 shows the work partitioning (column *f*) and slowdown (*sd*) used in this experiment. According to Tables 13 and 15, the aggregate slowdown for this case is 3.27/1.33 = 2.46. Note that, even though there is one application executing on alpha2, its local slowdown due to load imbalance is 1.33.

7 Summary

It is common for users and/or application designers to know the execution time of their applications when in dedicated mode. Our experiments show that execution time in dedicated mode and execution time under contention are directly proportional by a factor that we identify as the *slowdown* caused by contention for resources. Based on these facts, the contention models developed in this paper provide slowdown factors which determine, according to the load on the system, how much slower an application will compute or communicate. These factors can be multi-

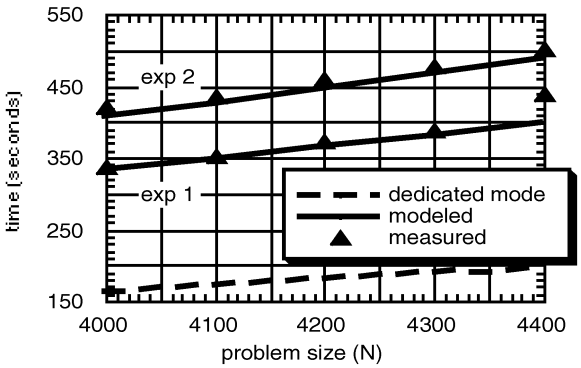

Fig. 17. Times for two experiments with the SOR algorithm executing with constraint-based work partitioning on a nondedicated-network cluster in dedicated mode and with contention.

TABLE 12
Local Slowdown in Each Node of a Dedicated-Network Cluster

Node	sd	f	ew	$(1+ew) \times sd$
1	3	1 / 4 (25%)	0	**3.0**
2	2	1 / 6 (17%)	- 0.33	1.34
3	2	1 / 4 (25%)	0	2.0
4	1	1 / 3 (33%)	0.33	1.33

TABLE 13
Work Partitioning for the Targeted Application in Dedicated Mode

Node	w	f	$(1+ew_i)/w_i$
alpha1	3.07	1 / 6 (17%)	0.22
alpha2	3.07	1 / 3 (33%)	0.43
rs1	1	1 / 6 (17%)	0.66
rs2	1	1 / 3 (33%)	**1.33**

TABLE 14
Data for Experiment 1 in Fig. 17

Node	w	sd	f	$(1+ew_i) \times sd_i/w_i$
alpha1	3.07	1	1 / 6 (17%)	0.22
alpha2	3.07	1	1 / 3 (33%)	0.43
rs1	1	2	1 / 3 (33%)	**2.67**
rs2	1	2	1 / 6 (17%)	1.33

TABLE 15
Data for Experiment 2 in Fig. 17

Node	w	sd	f	$(1+ew_i) \times sd_i/w_i$
alpha1	3.07	4	15 / 55 (27%)	1.42
alpha2	3.07	1.33	15 / 55 (27%)	0.47
rs1	1	3	15 / 55 (27%)	**3.27**
rs2	1	2	10 / 55 (19%)	1.45

plied by the time that an application takes to execute in dedicated mode to produce a realistic performance prediction. The experiments performed show that it is possible to predict application slowdown with reasonable accuracy and to use it to estimate application performance.

In this paper, we modeled three types of slowdown. We modeled slowdown for the time required to compute an application task at a single execution site in the system (*local slowdown*), and for the time required for communication between distinct execution sites (*communication slowdown*). An important parameter for these models proved to be the bandwidth between each pair of machines in the cluster. To determine the effects of contention, we also developed a model to provide the *aggregate slowdown*—the delay on an individual parallel application due to contention from other applications

sharing the cluster. The determination of aggregate slowdown varies with the work partitioning policy and was developed here for two common work partitioning policies (load-dependent and constraint-based). The model is parameterized by the local slowdown present in each node of the cluster and the relative weight of each node in the cluster, all of which contribute substantively to application performance. We performed a set of experiments comparing modeled and actual times on a dedicated system with a synthetic load for a set of benchmarks commonly found in high performance scientific applications. The experiments showed our models to predict relatively accurately—on average within 15 percent—the delay imposed on an individual application due to contention on the cluster.

In developing the models proposed, several important results were observed. These results are often overlooked in other research efforts. They are listed below:

- Competing computation and communication activities affect the targeted application differently and must be taken into account separately. For this reason, the behavior of the competing applications is an important parameter for the local slowdown.
- The bandwidth available in the communication links used by the competing applications affects the slowdown imposed by competing communication activity on the computation of the targeted application.
- Several factors affect the slowdown imposed on communication activity. However, they can be embedded in the available bandwidth, which is the only parameter necessary to calculate the communication slowdown.
- Using a tailored burst size in the benchmark used to measure the available bandwidth leads to more accurate predictions for the communication slowdown.

In this paper, we have made assumptions about the applications, the system, and the usage of the models. Relaxing these assumptions will lead to increased generalization of the models. Considering contention for memory is the first step in generalizing the model. We have assumed that all applications fit in memory, or at least the working set of all the applications, and no delay is imposed by lack of memory. Adding the effects of contention for memory to the models explicitly will expand their scope.

We have also assumed that all the applications executing on a workstation have the same priority and are locally scheduled in a round robin fashion. This may be true for systems dedicated to coarse-grain, CPU-bound applications, in which the largely used priority-based scheduling policy is reduced to round robin. However, in an environment like a nondedicated-network cluster, formed by general-use machines, coarse-grain applications compete with I/O-bound ones, which have higher priority. Including the effects of priority differences in the models will enable their usage in more general platforms.

We developed aggregate slowdown factors considering communication costs to be outweighed by computation costs. This is common among high performance scientific applications. However, for the model to cover a broader range of applications, the effects of contention for the network (i.e., bandwidth variation in the communication links used by the targeted parallel application) must be taken into account.

In addition, some of our models are based on information about the applications executing on the system. Dealing with partial information will allow our models to be used in a wider range of situations in which information is not available for all the competing applications. In this case, the parameters used by the models should be complemented with statistical information about the system usage.

ACKNOWLEDGMENTS

The authors would like to thank their colleagues in the Parallel Computation Laboratory at the University of California, San Diego (UCSD), and in the San Diego Supercomputer Center for their support, especially Stephen Fink, Karan Bhatia, Mike Vildibill, Ken Steube, Cindy Zheng, and Victor Hazlewood. They would also like to thank Professor Joseph Pasquale for the usage of the machines in the Computer Systems Laboratory at UCSD. This paper was supported by the US National Science Foundation, ASC-9701333 and by the Defense Advanced Research Projects Agency, contract number N66001-97-C-8531.

REFERENCES

[1] M. Atallah, C. Black, D. Marinescu, H. Siegel, and T. Casavant, "Models and Algorithms for Coscheduling Compute-Intensive Tasks on a Network of Workstations," *J. Parallel and Distributed Computing,* vol. 16, pp. 319-327, 1992.

[2] A. Beguelin, J. Dongarra, G. Geist, R. Manchek, and V. Sunderam, "Graphical Development Tools for Network-Based Concurrent Supercomputing," *Proc. Supercomputing 91,* pp. 435-444, 1991.

[3] F. Berman, R. Wolski, S. Figueira, J. Schopf, and G. Shao, "Application-Level Scheduling on Distributed Heterogeneous Networks," *Proc. Supercomputing '96,* Nov. 1996.

[4] A. Bricker, M. Litzkow, and M. Livny, "Condor Technical Summary," Technical Report #1069, Computer Science Dept., Univ. of Wisconsin, May 1992.

[5] W.L. Briggs, "A Multigrid Tutorial," *Society for Industrial and Applied Mathematics,* 1987.

[6] H. Dietz, W. Cohen, and B. Grant, "Would You Run It Here. . . or There? (AHS: Automatic Heterogeneous Supercomputing)," *Proc. Int'l Conf. Parallel Processing,* vol. II, pp. 217-221, Aug. 1993.

[7] X. Du and X. Zhang, "Coordinating Parallel Processes on Networks of Workstations," *J. Parallel and Distributed Computing,* vol. 46, no. 2, pp. 125-135, Nov. 1997.

[8] A.C. Dusseau, R.H. Arpaci, and D.E. Culler, "Effective Distributed Scheduling of Parallel Workloads," *Proc. ACM SIGMETRICS '96,* pp. 25-36, May 1996.

[9] S.M. Figueira, "Modeling the Effects of Contention on Application Performance in Multi-User Environments," doctoral dissertation, Computer Science Eng. Dept., Univ. of Calif., San Diego, Dec. 1996.

[10] S.M. Figueira and F. Berman, "Modeling the Effects of Contention on the Performance of Heterogeneous Applications," *Proc. Fifth Int'l Symp. High-Performance Distributed Computing,* pp. 392-401, Aug. 1996.

[11] S.J. Fink, S.B. Baden, and S.R. Kohn, "Flexible Communication Mechanisms for Dynamic Structured Applications," *Proc. Third Int'l Workshop Parallel Algorithms for Irregularly Structured Problems,* Aug. 1996.

[12] S. Leutenegger and X. Sun, "Distributed Computing Feasibility in a Non-Dedicated Homogeneous Distributed System," *Proc. Supercomputing '93,* pp. 143-152, Nov. 1993.

[13] "MPI: A Message-Passing Interface Standard," *Proc. Message-Passing Interface Forum,* June 1995.
[14] D. Bailey, E. Barszcz, J. Barton, D. Browning, R. Carter, L. Dagum, R. Fatoohi, S. Fineberg, P. Frederickson, T. Lasinski, R. Schreiber, H. Simeon, V. Venkatakrishnan, and S. Weeratunga, The NAS Parallel Benchmarks (94), Technical Report RNR-94-007, NASA Ames Research Center Mar. 1994.
[15] A.L. Rosenberg, "Guidelines for Data-Parallel Cycle-Stealing in Networks of Workstations, II: On Maximizing Guaranteed Output," *Proc. Second Merged Symp. IPPS/SPDP,* Apr. 1999.
[16] P. Sobalvarro, S. Pakin, W.E. Weihl, and A.A. Chien, "Dynamic Coscheduling on Workstation Clusters," *Proc. Workshop Job Scheduling Strategies for Parallel Processing,* Mar. 1998.
[17] V. Sunderam, "PVM: A Framework for Parallel Distributed Computing," *Concurrency: Practice and Experience,* vol. 2, no. 4, pp. 315-339, Dec. 1990.
[18] D. Whitley, T. Starkweather, and DUAnn Fuquay, "Scheduling Problems and Traveling Salesman: The Genetic Edge Recombination Operator," *Proc. Int'l Conf. Genetic Algorithms,* 1989.
[19] R. Wolski, N. Spring, and J. Hayes, "The Network Weather Service: A Distributed Resource Performance Forecasting Service for Metacomputing," *J. Future Generation Computer Systems,* 1998.
[20] R. Wolski, "Dynamically Forecasting Network Performance Using the Network Weather Service," *J. Cluster Computing,* vol. 1, no. 1, pp. 119-132, 1998.
[21] X. Zhang and Y. Yan, "A Framework of Performance Prediction of Parallel Computing on Non-Dedicated Heterogeneous Networks of Workstations," *Proc. 1995 Int'l Conf. Parallel Processing,* vol. I, pp. 163-167, 1995.

Silvia M. Figueira received the BS and MS degrees in computer science from the Federal University of Rio de Janeiro (UFRJ), Brazil, in 1988 and 1991, respectively, and the PhD degree in computer science from the University of California, San Diego, in 1997. She was involved in research as a member of the technical staff at the Computing Center of UFRJ (NCE/UFRJ) from 1985 to 1991. Currently, she is an assistant professor of computer engineering at Santa Clara University. Her current research interests are in the area of parallel computation, particularly support for high performance computing in networks of workstations.

Francine Berman is a professor of computer science and engineering and the director of the Grid Computing Laboratory at the University of California, San Diego. Her research interests over the last two decades have focused on parallel and distributed computation and the areas of programming environments, tools, and models that support high performance computing, in particular. She currently co-leads the AppLeS project which is focused on the development of dynamic application scheduling methods and performance technology for multi-user distributed environments. She has served on numerous editorial boards and on program and conference committees in the areas of parallel computing and grid computing. She is involved in the Metasystems Thrust of the National Partnership for Advanced Computational Infrastructure (NPACI) where she provides leadership in application scheduling issues and she is a senior fellow at the San Diego Supercomputer Center, a fellow of the ACM, and a member of the IEEE.