Modeling the Slowdown of Data-Parallel Applications in Homogeneous and Heterogeneous Clusters of Workstations

Silvia M. Figueira* and Francine Berman**

Department of Computer Science and Engineering
University of California, San Diego
La Jolla, CA 92093-0114
{silvia,berman}@cs.ucsd.edu
http://www-cse.ucsd.edu/users/{silvia,berman}

Abstract

Data-parallel applications executing in multi-user clustered environments share resources with other applications. Since this sharing of resources dramatically affects the performance of individual applications, it is critical to estimate its effect, i.e., the application slowdown, in order to predict application behavior. In this paper, we develop a new approach for predicting the slowdown imposed on data-parallel applications executing on homogeneous and heterogeneous clusters of workstations. Our model synthesizes the slowdown on each machine used by an application into a contention measure – the aggregate slowdown factor – used to adjust the execution time of the application to account for the aggregate load. The model is parameterized by the work (or data) partitioning policy employed by the targeted application, the local slowdown (due to contention from other users) present in each node of the cluster, and the relative weight (capacity) associated with each node in the cluster. This model provides a basis for predicting realistic execution times for distributed data-parallel applications in production clustered environments.

1. Introduction

Clusters of workstations have been used effectively as parallel machines for large, data-parallel, scientific applications [3, 4, 6, 16]. Workstations in such clusters are typically time-shared, causing competing applications to split the CPU and memory capacity on each node. The result of such sharing is that the fraction of resources available for a single distributed parallel application is reduced, causing a *slowdown* in the overall application performance. In [1], Arpaci et al. have shown that this slowdown can be significant. When slowdown can be predicted accurately, this information can be used to improve scheduling decisions in shared distributed systems [9].

A number of researchers have investigated how slowdown might be calculated and used to improve performance. Slowdown on a single machine has been used for task scheduling (as shown in [3] and [6]) and to predict performance (as shown in [10], [13] and [22]). Co-scheduling algorithms developed for networks of workstations have taken the slowdown on each machine into account, as shown in [2] and [7]. Weissman [19] has proposed a scheduling model based on resource contention using an *interference paradigm*. The interference measure determines how much slower an application will compute or a parallel application will communicate.

In this paper, we develop a new model for predicting the slowdown imposed on data-parallel applications executing on homogeneous and heterogeneous clusters of workstations. Our model synthesizes the slowdown on each machine used by an application into a contention measure – the *aggregate slowdown factor* – used to adjust the execution time of the application to account for the aggregate load. This factor can be combined with the time to execute the application in dedicated mode (on the same cluster) to provide an estimate of application performance in the presence of contention.

The aggregate slowdown factor is based on both application-specific resource usage and node computational capacity. Node capacity depends both on the dedicated capacity (e.g., CPU speed) of the node and on the load experienced by the node. Note that the dedicated capacity of nodes in a cluster may be non-uniform when the nodes are heterogeneous.

This paper is organized as follows. Section 2 introduces our model and the aggregate slowdown measure. We focus on aggregate slowdown factors for two common work partitioning policies (load-dependent and constraint-based) in Section 3. Section 4 discusses the local

* Supported by a scholarship from CAPES (Brazil).
** Supported in part by NSF contract number ASC-9301788 and ASC-9701333.

slowdown in each node of the cluster. In Sections 5 and 6, we describe our experiments and evaluate the accuracy of the model presented. Section 7 concludes with a summary.

2. The Environment

We define a contention measure, the *aggregate slowdown factor*, that determines how load in the cluster will retard the performance of an individual application. Using this factor, the time X to execute a data-parallel application on a cluster is given by

$$X = X_{ded} \times sd, \quad (1)$$

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 on the cluster without interference from competing applications.

For this paper, we focus on loosely synchronous CPU-bound applications as defined by Fox [12]. These applications are coarse-grained data-parallel scientific applications in which computation and communication phases alternate. Such applications can profit from execution in clustered environments.

The computational environment is a cluster of homogeneous or heterogeneous workstations. Note that the model is independent of the number of nodes in the cluster. We assume that the cluster is shared by CPU-bound serial and/or data-parallel distributed applications, which execute for the entire duration of 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. 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.

For this model, we consider the effects due to system overhead as well as application communication costs to be minimal. Although considering communication costs seemed essential at first, our experiments showed that for representative CPU-bound data-parallel applications the amount of communication was not significant.

In our experiments, we assumed 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, reduce to round-robin when the competing applications are CPU-bound [8].

Finally, 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. The model can be extended to include more varied memory access costs in a straightforward manner.

3. Aggregate Slowdown

Aggregate slowdown is defined to be the performance slowdown of an application due to other applications sharing the cluster. Aggregate slowdown factors must be calculated based both on the work partitioning policy and on 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, is the delay imposed on an application running on node a as a function of other applications that share a's CPU. We show how to calculate the local slowdown 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 data-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 that have a dynamic work partitioning independent of the load, i.e., applications for which work partitioning varies throughout their 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*.

3.1 Load-Dependent Partitioning

In this partitioning, work is allocated according to the available computational capacity of each node 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

$$sd = \frac{aggregate\ capacity_{ded}}{aggregate\ capacity} = \frac{\left(\sum_{a=1}^{n} w_a\right)}{\left(\sum_{a=1}^{n} \frac{w_a}{sd_a}\right)}. \quad (2)$$

Figure 1 illustrates a load-dependent partitioning. The work for application A is partitioned according to the node capacities of a set of heterogeneous nodes, as shown in the figure. If node 4 is twice as slow as the others (i.e., $t_1 = t_2 = t_3 = t_4 / 2$), then $sd = (2 + 2 + 2 + 1) / (2 / 2 + 2 / 2 + 2 / 3 + 1 / 1) = 7 / 3.67 = 1.9$ according to equation (2).

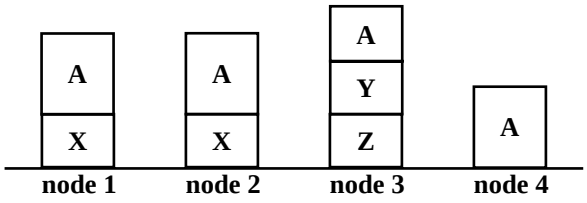

Figure 1: Cluster formed by 4 heterogeneous nodes and shared by three applications: A, X, and Y. Application A's work is divided among the nodes according to their computational capacity. Each box represents the amount of work associated with the corresponding node.

3.2 Constraint-Based Partitioning

When work is partitioned based on a set of constraints, such as memory availability or data locality, the aggregate slowdown can be calculated from the extra amount of work (relative to the work assigned in a *uniform* partitioning, in which the work is partitioned evenly among the nodes) assigned to each node. That is, 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

$$time_a = (time_{a,ded} \times sd_a) + (time_{a,ded} \times ew_a \times sd_a) = time_{a,ded} \times sd_a \times (1 + ew_a), \quad (3)$$

where

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

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

- f_a = fraction of work assigned to node *a*, and
- *n* is the number of nodes in the cluster.

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, which is given by equation (1). From (1) and (3), we can calculate the aggregate slowdown as

$$sd = \frac{X}{X_{ded}} = \frac{\max_a \left\{ \frac{(1 + ew_a) \times sd_a}{w_a} \right\}}{\max_a \left\{ \frac{(1 + ew_a)}{w_a} \right\}}, \quad (4)$$

where the numerator represents the time it takes to execute the application in the presence of contention, and the denominator represents the time it takes to execute the application in dedicated mode.

Note that the uniform partitioning policy, in which each node is assigned an equal amount of work, is characterized by $ew_a = 0$ for all *a*, and the aggregate slowdown for this policy can also be calculated by equation (4).

Equation (4) 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 non-

dedicated 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}), equation (4) is modified to

$$sd = \frac{\max_a \left\{ \frac{(1 + ew_{a,1}) \times sd_a}{w_a} \right\}}{\max_a \left\{ \frac{(1 + ew_{a,2})}{w_a} \right\}}, \quad (5)$$

where

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

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

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

In the setting where the dedicated time X_{ded} is given for a uniform work partitioning, equation (4) can be reduced to

$$sd = \frac{\max_a \left\{ \frac{(1 + ew_a) \times sd_a}{w_a} \right\}}{\max_a \{1/w_a\}}, \quad (6)$$

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.

Since $\max_a \{1/w_a\} = 1$, for any set of nodes, equation (6) can be reduced to

$$sd = \max_a \left\{ \frac{(1 + ew_a) \times sd_a}{w_a} \right\}. \quad (7)$$

Figure 2 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. This happens because nodes 2 and 3 are both assigned 3 work units, which are delayed by a factor of 2, whereas node 4 is assigned 5 work units, which execute in full speed. In this case, according to equation (7), $sd = (1 + 0) \times 2 / 1 = 2$. 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 equation (7).

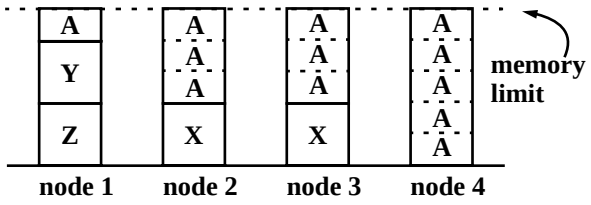

Figure 2: Cluster formed by 4 nodes and shared by three applications: A, X, and Y. Application A's 12 units of work (which are determined by the dotted lines) are divided among the nodes according to memory availability. Each box represents the amount of work associated with the corresponding node.

4. Local Slowdown

The aggregate slowdown sd is calculated based on the local slowdown sd_a on each workstation a of the cluster. The local slowdown on node a is the delay imposed on an application running on node a as a function of other applications that share a's CPU. In [10], we presented a model to calculate the local slowdown based on information (such as the computation/communication ratio) about the applications executing on the system. Since this information may not be always available, in this paper we present a different way of calculating local slowdown based on information provided by the system.

If the scheduling policy implemented by the local scheduler executed on each workstation is reduced to round-robin (which is generally the case when all tasks are CPU-bound and have the same priority), the slowdown imposed by contention on each node a can be calculated simply as

$$sd_a = p_a + 1, \quad (8)$$

where p_a is the number of extra processes executing on node a.

Equation (8) - with some slight variations - has been used for slowdown predictions in [3, 4, 6, 13, 19, 22]. However, it assumes that the competing applications executing on the nodes of the cluster are well balanced and executing at full speed (without idle intervals). This may not be always the case.

When the load is not well-balanced and idle cycles occur in a node, the local slowdown used in the aggregate slowdown calculation must be a function of the fraction of the *busy* (as opposed to idle) time of the competing applications. In this case, the slowdown imposed by contention on node a should be modeled pessimistically as

$$sd_a = 1 + \sum_{i=1}^{p_a} busy_{a,i}, \quad (9)$$

where $busy_{a,i}$ is the fraction of time in which application i is busy on node a, i.e., application i is either computing within its own time slice, communicating, or waiting for its time slice.

The value for $busy_{a,i}$ depends on the effects of the load in the cluster, including the targeted application. To calculate $busy_{a,i}$, we use the CPU-usage associated with application i on node a, and we have

$$busy_{a,i} \approx \min\{1,\ CPU_{a,i} \times (p_a + 1)\}, \quad (10)$$

where $CPU_{a,i}$ is the fraction of a's CPU used by application i. Note that $CPU_{a,i}$ is a fraction provided by the operating system. This fraction can also be predicted by tools such as the Network Weather Service [21].

According to equation (10), application i is busy all the time ($busy_{a,i} = 1$) when its CPU-usage corresponds to at least $1 / (p_a + 1)$. When application i's CPU-usage is less than $1 / (p_a + 1)$, it does not use the CPU all the time, and the product of its CPU-usage and the number of applications in the system gives the fraction of time in which application i is busy. Note that, since the targeted application will affect the amount of time application i is busy, $p_a + 1$ (not just p_a) is used.

Equation (10) is actually an upper bound on the time that application i is busy. It considers the worst case in which applications' computation cycles are synchronized (i.e., all applications compete for the CPU at the same time) and the targeted application neither has nor induces idle cycles in the competing applications. When computation cycles are not synchronized, applications will compete for the CPU less often. Also, when the targeted application either has or induces idle cycles in the competing applications, these applications may stay busy less time. In these cases, the slowdown on the node will decrease.

5. Experiments

The formulas presented in Section 3 provide a model for the aggregate slowdown. To assess the accuracy of this model, we performed a large collection of experiments on a wide range of CPU-bound benchmarks commonly found in high-performance scientific applications. Some examples of the applications used were: Jacobi2D [5], Jacobi3D [5], Red-Black SOR [5], Multigrid [5], Genetic Algorithm [20], and LU Solver [15]. 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 CPU-bound synthetic loads. The synthetic loads were formed by a combination of CPU-bound serial and/or data-parallel applications distributed over the cluster. In this paper, we show representative experiments for each scheduling policy. On all graphs, we show actual times in contentious (multi-user) environment, predicted times using our slowdown model, and dedicated (single-user) actual times for comparison. A more complete list of the experiments can be found in [9].

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

Our experiments were performed on two platforms: a cluster of homogeneous nodes, represented by a DEC Alpha-Farm located at the San Diego Supercomputer Center, and a cluster of heterogeneous nodes formed by two DEC Alphas (located in the Computer Systems Laboratory at UCSD) and two IBM RS-6000s (located in the Parallel Computation Laboratory, also at UCSD). In the DEC Alpha-Farm, the nodes are connected by a GIGAswitch, which is dedicated to the nodes. In the heterogeneous cluster, the nodes are part of different Ethernet networks.

In this section, we show a representative subset of the experiments performed on the homogeneous and heterogeneous clusters described above.

5.1 Experiments on the Homogeneous Cluster

In the first set of experiments, we targeted clusters of uniform workstations.

Figure 3 illustrates two representative experiments with the load-dependent work partitioning policy. Shown is a distributed SOR application [5] developed using PVM [18] and executed on 4 nodes of the DEC Alpha-Farm with two different loads. The parameters for the experiments are shown in Table 1. In experiment 1 (*exp 1*), there is a CPU-bound data-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. In experiment 2 (*exp 2*), there is a CPU-bound data-parallel application executing on two of the nodes, and two serial CPU-bound applications executing on another node, as shown in Figure 1. 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.

Table 1: Parameters for the Experiments in Figure 3

Node	*sd* for exp 1	*sd* for exp 2
1	2	2
2	2	2
3	1	3
4	1	1

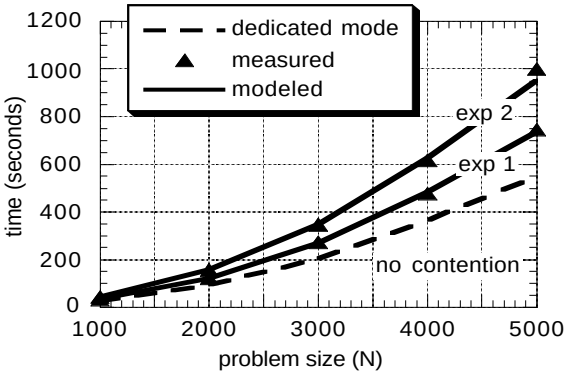

Figure 3: Time to execute the SOR algorithm on 4 nodes of the DEC Alpha-Farm in dedicated mode and with 2 different loads.

Figure 4 represents experiments using the constraint-based work partitioning policy. Shown is a Multigrid application [5] (developed using KeLP [11] and MPI [14]) 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 data-parallel application, and one of the nodes (node 1) also hosts two serial, CPU-bound applications, as shown in Figure 2. The blocks of data were divided among the nodes according to a set of constraints resulting in the partitioning shown in Table 2 (column *f*). Table 2 also shows the other parameters (*sd* and *ew*) used to calculate the aggregate slowdown (3.0). Note that the dedicated time is given for a uniform work partitioning.

Table 2: Parameters for the Experiment in Figure 4

Node	*sd*	*f* (%)	*ew*	$(1 + ew) \times sd$
1	3	25	0	3.00
2	2	17	- 0.33	1.34
3	2	25	0	2.00
4	1	33	0.33	1.33

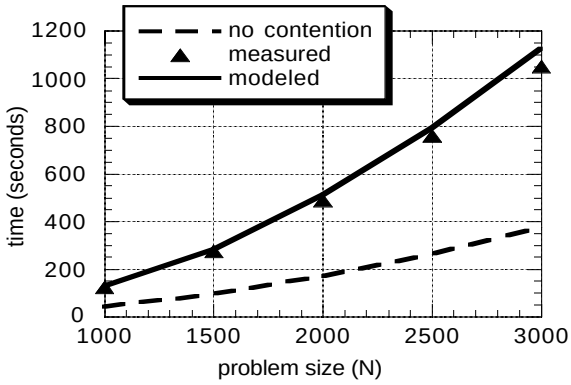

Figure 4: Time to execute the Multigrid application on 4 nodes of the DEC Alpha-Farm, with constraint-based partitioning, in dedicated mode and with contention on the nodes.

Figure 5 and Figure 6 also represent experiments using the constraint-based work partitioning policy. They present examples of a Jacobi3D algorithm [5] (developed using KeLP [11] and MPI [14]) executing for different problem sizes (given by $N \times N$). The dedicated time was given for a uniform work partitioning. Figure 5 shows modeled and measured times for execution on 4 nodes where other applications are also executing. In experiment 1 (*exp 1*), there was a CPU-bound parallel application executing on two of the nodes as shown in Table 3, imposing a slowdown of 2 on the execution of the Jacobi3D algorithm. In experiment 2 (*exp 2*), there was a CPU-bound parallel application executing on two of the nodes, one of which was also shared by a serial CPU-bound application, as shown in Table 4. In this case, the Jacobi3D algorithm was slowed by a factor of 3.

Table 3: Parameters for Experiment 1 in Figure 5

Node	*sd*	*f* (%)	*ew*	$(1 + ew) \times sd$
1	2	25	0	2
2	2	25	0	2
3	1	25	0	1
4	1	25	0	1

Table 4: Parameters for Experiment 2 in Figure 5

Node	*sd*	*f (%)*	*ew*	$(1 + ew) \times sd$
1	3	25	0	3
2	2	25	0	2
3	1	25	0	1
4	1	25	0	1

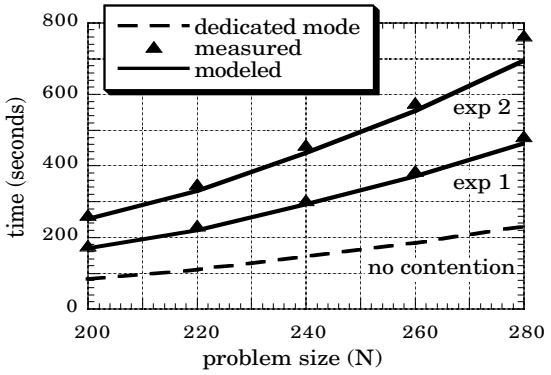

Figure 5: Time to execute the Jacobi3D algorithm on 4 nodes of the DEC Alpha-Farm in dedicated mode and with 2 different loads.

Figure 6 shows an example of the execution of the Jacobi3D benchmark for different problem sizes (given by $N \times N$) on 4 nodes, in which the CPU-bound applications described by Table 5 are also executing. The fractions in the table represent the amount of time the application is busy on the respective node. The work partitioning was constraint-based. The aggregate slowdown factor is 2.375, even though there are two applications executing on node 4. This is explained by the imbalance due to the work partitioning of the competing applications. In particular, this imbalance causes application 2 to be idle part of the time, increasing node 4's computational capacity. Note that the dedicated time parameter is given for a uniform work partitioning.

Table 5: Busy Fractions for the Experiment in Figure 6

Node	Parallel Application 1	Parallel Application 2
1		
2	2 / 8	
3		1
4	1	3 / 8

Table 6: Parameters for the Experiment in Figure 6

Node	sd	f (%)	ew	$(1 + ew) \times sd$
1	1.000	25	0	1.000
2	1.250	25	0	1.250
3	2.000	25	0	2.000
4	2.375	25	0	2.375

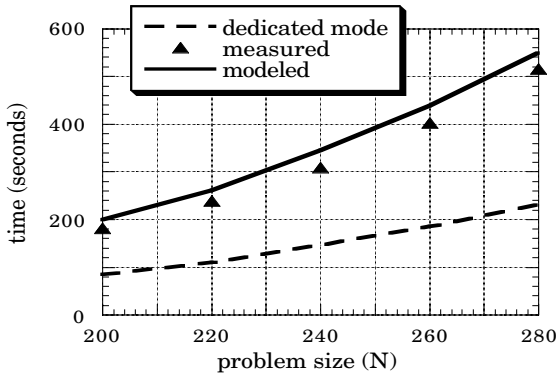

Figure 6: Time to execute the Jacobi3D algorithm on 4 nodes of the DEC Alpha-Farm in dedicated mode and together with the load described in Table 5.

5.2 Experiments on the Heterogeneous Cluster

We now relax the constraints that all the nodes in the cluster are uniform and communication links within the cluster are dedicated to the cluster itself.

Figure 7 shows an example of the execution of the SOR benchmark [5] (developed using PVM [18]) for different problem sizes (given by $N \times N$) on the 4-node heterogeneous cluster. The work was divided among the machines according to their capacity and loads. The ambient load was formed by one CPU-bound data-parallel application executing on the DEC Alphas. Table 7 describes the values used to calculate the aggregate slowdown for this situation, which is 8.14 / 5.08 = 1.6. Note that the time to execute the SOR benchmark for a 4000×4000 matrix is shorter than the time estimated by the model because, for this problem size, the round-robin scheduling policy (assumed on each workstation) is not a "perfect" round-robin, and the SOR benchmark gets higher priority.

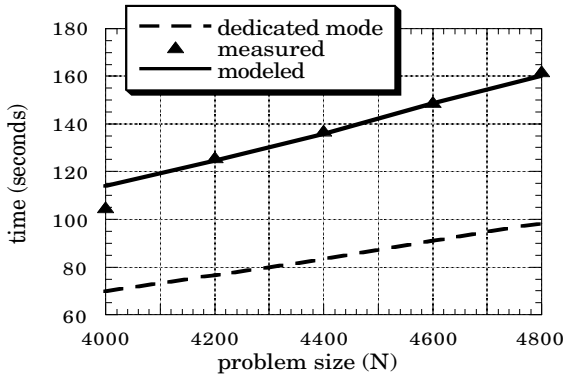

Figure 7: Times for the SOR algorithm executing with load-dependent work partitioning on a heterogeneous cluster in dedicated mode and under contention.

Table 7: Parameters for the Experiment in Figure 7

Node	*w*	*sd*
$alpha_1$	3.07	2
$alpha_2$	3.07	2
rs_1	1.00	1
rs_2	1.00	1

Figure 8 represents experiments using a load-dependent work partitioning policy with a Genetic Algorithm application [20] developed using PVM [18]. Shown are modeled and measured times for execution with different problem sizes (given by population size) on four nodes of the heterogeneous cluster. In this experiment, the IBM RS-6000s are also executing a CPU-bound data-parallel application. The parameters for the experiment are shown in Table 7. The aggregate slowdown is 1.18.

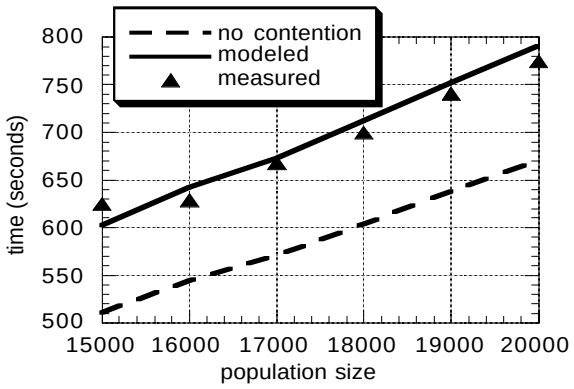

Figure 8: Times for the Genetic Algorithm executing with load-dependent work partitioning on a heterogeneous cluster in dedicated mode and under contention.

Table 8: Parameters for the Experiment in Figure 8

Node	*w*	*sd*
$alpha_1$	2.3	1
$alpha_2$	2.3	1
rs_1	1.0	2
rs_2	1.0	2

Figure 9 represents experiments using a constraint-based work partitioning policy. Shown is a representative SOR application [5] developed using PVM [18] and executed for different problem sizes (given by $N \times N$). The platform is a heterogeneous 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 9 (column *f*).

Table 9: Parameters for Experiments in Figure 9

Node	*w*	*f* (%)	$(1 + ew_i) / w_i$
$alpha_1$	3.07	17	0.22
$alpha_2$	3.07	33	0.43
rs_1	1.00	17	0.66
rs_2	1.00	33	1.33

In experiment 1 (*exp 1*) there was an additional CPU-bound data-parallel application executing on the 2 IBM RS-6000s. Table 10 shows the work partitioning (column *f*) and slowdown (*sd*) parameters used in this experiment. According to Table 9 and Table 10, the aggregate slowdown for this case is 2.67 / 1.33 = 2.01.

Table 10: Parameters for Experiment 1 in Figure 9

Node	*w*	*sd*	*f* (%)	$(1 + ew_i) \times sd_i / w_i$
$alpha_1$	3.07	1	17	0.22
$alpha_2$	3.07	1	33	0.43
rs_1	1.00	2	33	2.67
rs_2	1.00	2	17	1.33

In experiment 2 (*exp 2*) there was a CPU-bound data-parallel application executing on the 2 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. The fractions in Table 11 represent the amount of time the application is busy on the respective node. Table 12 shows the work partitioning (column *f*) and slowdown (column *sd*) used in this experiment. According to Table 9 and Table 12, the aggregate slowdown for this case is 3.27 / 1.33 = 2.46. Note that, even though there is one application executing on $alpha_2$, its local slowdown due to load imbalance is 1.33.

Table 11: Busy Fractions for Experiment 2 in Figure 9

Node	Parallel Appl. 1	Parallel Appl. 2	Serial Appl. 3	Serial Appl. 4	Serial Appl. 5
$alpha_1$			1	1	1
$alpha_2$		1 / 3			
rs_1	1	1			
rs_2	1				

Table 12: Parameters for Experiment 2 in Figure 9

Node	w	sd	f(%)	$(1 + ew_i) \times sd_i / w_i$
$alpha_1$	3.07	4.00	27	1.42
$alpha_2$	3.07	1.33	27	0.47
rs_1	1.00	3.00	27	3.27
rs_2	1.00	2.00	19	1.45

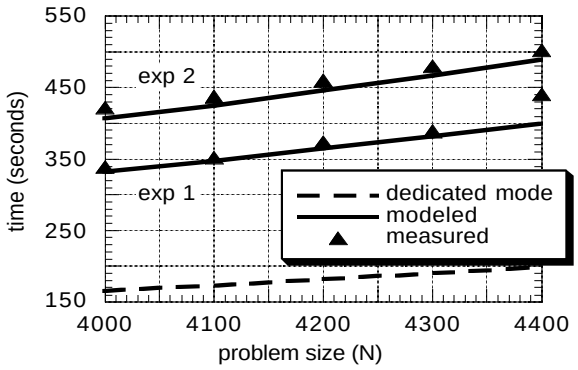

Figure 9: Times for two experiments with the SOR algorithm executing with constraint-based work partitioning on a heterogeneous cluster in dedicated mode and with contention.

Figure 10 also represents experiments using the constraint-based work partitioning policy. It presents examples of a Jacobi2D benchmark [5] (developed using KeLP [11] and MPI [14]) executing for different problem sizes (given by $N \times N$). The graph shows modeled and measured times for execution on the 4 nodes. One of the DEC Alphas and one of the IBM RS-6000s are also used to execute a well-balanced CPU-bound data-parallel task, as shown in Table 13, causing the aggregate slowdown to be 2. Note that the time estimated by the model is a little higher than the measured time because the round-robin scheduling policy (assumed on each workstation) is not a "perfect" round-robin and, in this case, Jacobi2D gets higher priority and executes faster than expected by the model.

Table 13: Parameters for the Experiment in Figure 10

Node	w	sd	f(%)	$(1 + ew_i) \times sd_i / w_i$
$alpha_1$	1.00	1	25	1.00
$alpha_2$	1.00	2	25	2.00
rs_1	1.84	2	25	1.09
rs_2	1.84	1	25	0.54

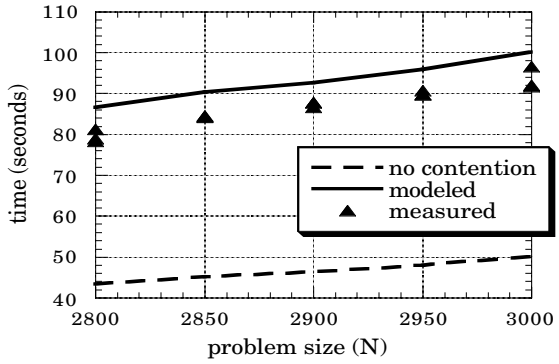

Figure 10: Times for the Jacobi2D benchmark executing with constraint-based work partitioning on the heterogeneous cluster in dedicated mode and with contention.

Figure 11 also presents examples of the Jacobi2D benchmark executing for different problem sizes (given by $N \times N$). The graph shows modeled and measured times for execution on the 4 nodes. The work partitioning was constraint-based. The contention is generated by one CPU-bound parallel application executing on the IBM RS6000s, a second one executing on the DEC Alphas, and a third one executing on one IBM RS-6000 and one DEC Alpha. This scenario is represented in Table 14. The aggregate slowdown is 3 due to the load in the most heavily loaded DEC Alpha, as shown in Table 15.

Table 14: Node Usage for the Experiment in Figure 11

Node	Parallel Application 1	Parallel Application 2	Parallel Application 3
$alpha_1$		✔	
$alpha_2$		✔	✔
rs_1	✔		✔
rs_2	✔		

Table 15: Parameters for the Experiment in Figure 11

Node	w	sd	f (%)	$(1 + ew_i) \times sd_i / w_i$
$alpha_1$	1.00	2	25	2.00
$alpha_2$	1.00	3	25	3.00
rs_1	1.84	3	25	1.63
rs_2	1.84	2	25	1.09

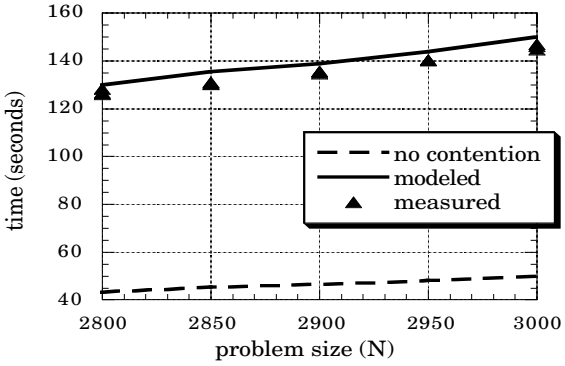

Figure 11: Times for the Jacobi2D benchmark executing with constraint-based work partitioning on the heterogeneous cluster in dedicated mode and with contention.

6. Evaluation of the Model

The models presented were developed based on the amount of busy time of the competing applications. They assume that the time to execute the targeted application is longer than the busy/idle cycles of the competing applications. If the target application is fast in comparison with the duration of these cycles, i.e., the time to execute the application is close to (or smaller than) the duration of one busy/idle cycle, the accuracy of the models decreases.

Figure 12 and Figure 13 illustrate the difference in accuracy obtained in the prediction of the time to execute the SOR benchmark on the heterogeneous cluster in two situations. In both situations, the SOR competes for the cluster with one CPU-bound data-parallel application that executes on one DEC Alpha and one IBM RS-6000. In Figure 12, the time to execute one busy/idle cycle of the competing application was 15.82 seconds. In this case, the time to execute the algorithm was longer than one busy/idle cycle of the competing application, and the average error was 2%. In Figure 13, the time to execute one busy/idle cycle of the competing application was 223.76 seconds. In this case, the time to execute the same algorithm was shorter than one busy/idle cycle of the competing application, and the variation of the actual times to execute the algorithm causes the average error to be within 12%.

Note that in Figure 13, for problem size 4400×4400, one execution of the benchmark with contention was faster than the execution in dedicated mode. This phenomenon is explained by a variation in the execution time in dedicated mode [17] caused by traffic in the network, which is nondedicated in the heterogeneous cluster. In our experiments, this variation is not significant because the amount of communication is small.

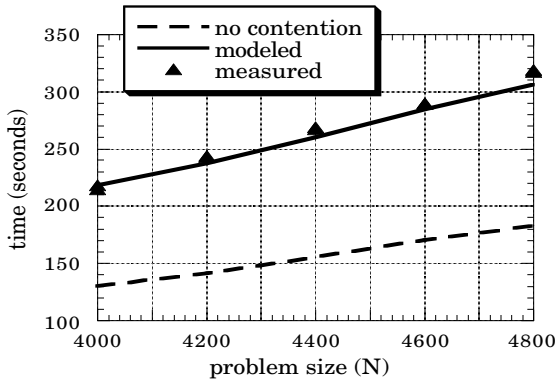

Figure 12: Time to execute the SOR benchmark in dedicated mode and competing with one application that has a 15.82-second busy/idle cycle.

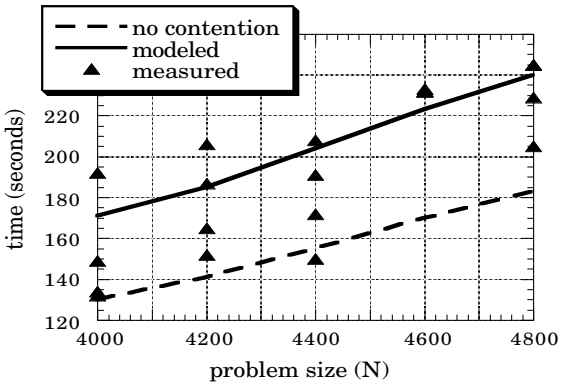

Figure 13: Time to execute the SOR benchmark in dedicated mode and competing with one application that has a 223.76-second busy/idle cycle.

7. Summary

In this paper, we have presented a model to predict contention effects in clustered environments. This model provides a basis for predicting realistic execution times for applications on clusters of workstations, a fundamental component of performance-efficient scheduling. The model is parameterized by the policy used to partition application work, 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.

To determine the effects of contention, we developed a measure of **aggregate slowdown** – the delay on an individual 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). We performed a set of experiments comparing modeled and actual times on a dedicated system with 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% – the delay imposed on an individual application due to contention on the cluster. Since the effect caused by contention in a time-shared environment can be large, as shown by our experiments, the aggregate slowdown provides a critical component in the accurate prediction of performance for data-parallel programs on multi-user clusters of workstations.

Acknowledgments

We would like to thank our colleagues in the UCSD Parallel Computation Laboratory and in the San Diego Supercomputer Center for their support, specially Stephen Fink, Karan Bhatia, Mike Vildibill, Ken Steube, Cindy Zheng, and Victor Hazlewood. We would also like to thank Professor Joseph Pasquale for the usage of the machines in the Computer Systems Laboratory at UCSD.

References

[1] R. H. Arpaci, A. C. Dusseau, A. M. Vahdat, L. T. Liu, T. E. Anderson, and D. A. Patterson, "The Interaction of Parallel and Sequential Workloads on a Network of Workstations", in *Proceedings of SIGMETRICS'95/PERFORMANCE'95 Joint International Conference on Measurement and Modeling of Computer Systems*, pp. 267-278, May 1995.

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

[3] A. Beguelin, J. Dongarra, G. Geist, R. Manchek, and V. Sunderam, "Graphical Development Tools for Network-Based Concurrent Supercomputing", in *Proceedings of Supercomputing 91*, pp. 435–444.

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

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

[6] H. Dietz, W. Cohen, and B. Grant, "Would you run it here...or there? (AHS: Automatic Heterogeneous Supercomputing)", in *Proceedings of the International Conference on Parallel Processing*, vol. II, pp. 217–221, August 1993.

[7] X. Du and X. Zhang, "Coordinating Parallel Processes on Networks of Workstations", Technical Report, High Performance Computing and Software Lab, University of Texas at San Antonio, August 1996.

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

[9] S. M. Figueira, "Modeling the Effects of Contention on Application Performance in Multi-User Environments," Ph.D. Dissertation, CSE Department, UCSD, December 1996.

[10] S. M. Figueira and F. Berman, "Predicting Slowdown for Networked Workstations," in *Proceedings of the Sixth International Symposium on High-Performance Distributed Computing*, August 1997.

[11] S. J. Fink, S. B. Baden, and S. R. Kohn, "Flexible Communication Mechanisms for Dynamic Structured Applications", in *Proceedings of the Third International Workshop on Parallel Algorithms for Irregularly Structured Problems*, Santa Barbara, CA, August 1996.

[12] G. Fox, "Hardware and Software Architectures for Irregular Problem Architectures," in *Unstructured Scientific Computation on Scalable Multiprocessors*, P. Mehrotra, J. Saltz, and R. Voigt, The MIT Press, Cambridge, MA, pp. 125–160, 1992.

[13] S. Leutenegger and X. Sun, "Distributed Computing Feasibility in a Non-Dedicated Homogeneous Distributed System", NASA - ICASE Technical Report 93-65, September 1993.

[14] Message-Passing Interface Forum, "MPI: A Message-Passing Interface Standard", University of Tennessee, Knoxville, TN, June 1995.

[15] NAS Parallel Benchmarks, http://www.nas.nasa.gov/NAS/NPB.

[16] NOW, http://now.cs.berkeley.edu.

[17] J. Schopf and F. Berman, "Performance Prediction in Production Environments," in *Proceedings of IPPS/SPDP '98*, to appear.

[18] V. Sunderam, "PVM: A Framework for Parallel Distributed Computing", *Concurrency: Practice and Experience*, vol. 2, n. 4, pp. 315–339, December 1990.

[19] J. Weissman, "The Interference Paradigm for Network Job Scheduling", in *Proceedings of the Heterogeneous Computing Workshop*, pp. 38-45, April 1996.

[20] D. Whitley, T. Starkweather, and DUAnn Fuquay, "Scheduling Problems and Traveling Salesman: The Genetic Edge Recombination Operator," in *Proceedings of International Conference on Genetic Algorithms*, 1989.

[21] R. Wolski, "Dynamically Forecasting Network Performance to Support Dynamic Scheduling Using the Network Weather Service," in *Proceedings of the 6th High-Performance Distributed Computing Conference*, August 1997.

[22] X. Zhang and Y. Yan, "A Framework of Performance Prediction of Parallel Computing on Non-dedicated Heterogeneous Networks of Workstations", in *Proceedings of 1995 International Conference of Parallel Processing*, vol. I, pp. 163-167, 1995.

Biographies

Silvia M. Figueira was born in Rio de Janeiro, Brazil. She received both her B.S. and M.Sc. degrees in Computer Science from the Federal University of Rio de Janeiro, Brazil, in 1988 and 1991, respectively, and her Ph.D. 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 of NCE/UFRJ (The Computing Center of the Federal University of Rio de Janeiro) from 1985 to 1991. Currently, she is a Postdoctoral Fellow at the Computer Science Department of the University of California, San Diego. Her current research interests are in high-performance distributed computing.

Francine Berman is Professor of Computer Science and Engineering at U. C. San Diego and a Senior Fellow at the San Diego Supercomputer Center. She received her Ph.D. in 1979 from the University of Washington. Her research focuses on application scheduling in metacomputing, and programming environments, models and tools which support high-performance computing on distributed networks. Dr. Berman has participated on numerous Program and Conference Committees and will serve as co-Chair of the 1999 High Performance Distributed Computing Conference. She currently serves on the editorial boards of IEEE Transactions on Parallel and Distributed Computing, the Journal of Parallel and Distributed Computing, SIAM Review, and as Area Editor for Metacomputing at The Journal of Supercomputing.