Modeling Contention Effects in Clustered Environments

Silvia M. Figueira[1] and Francine Berman[2]

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 clustered environments share resources with other applications. Since system load can vary dramatically, it is critical to provide an accurate model of the effects of contention on application performance in order to provide realistic assessments of application behavior. In this paper, we present 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 and is parameterized by the data allocation policy employed by the targeted application, the local slowdown present in each node of the cluster, and the relative weight associated with each node in the cluster.

1. Introduction

Clusters of workstations have been used effectively as parallel machines for large, data-parallel, scientific applications [1, 4, 6, 13]. Workstations in such clusters are typically time-shared, causing competing applications to split 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.

When slowdown can be predicted accurately [9], this information can be used to improve scheduling decisions. Good scheduling is fundamental to the performance of parallel applications in shared distributed systems [3], and an adequate contention model is critical to the development of performance-efficient schedules.

In this paper, we develop a model for predicting the slowdown imposed on data-parallel applications executing on time-shared clusters of workstations. Our model utilizes a contention measure – the ***aggregate slowdown factor*** **–** to adjust the execution time of an application to account for 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 of the node. Note that the dedicated capacity of nodes in a cluster may be non-uniform if 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 three common data allocation policies (uniform, constraint-based, and load-dependent) in Section 3, and verify their accuracy experimentally in Section 4. Section 5 concludes with a summary.

1. Supported in part by CAPES and UFRJ (Brazil), and by NSF contract number ASC-9301788.
2. Supported in part by NSF contract number ASC-9301788.

2. The Environment

We will 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 to execute a data-parallel application X on a cluster is given by

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

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 X on the cluster without interference from competing applications.

For this paper, we focus on loosely synchronous applications as classified by Fox [11]. These applications are coarse-grained data-parallel scientific applications in which computation and communication phases alternate. Such applications can profit from distribution in clustered environments as shown in [2].

The computational environment is a cluster of workstations, 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. For this model, we consider the effects due to system overhead as well as application communication costs 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.

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.

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 Factors

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 data allocation 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. In a round-robin environment, we can assume that $sd_a = p_a + 1$, where p_a is the number of additional applications executing on node a.

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]), such that the slowest node has the smallest weight. 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 application dependent [7, 8] and should be calculated with a benchmark that is representative of the class of the targeted application.

The following subsections develop aggregate slowdown factors for three data allocation policies commonly used to partition high-performance data-parallel applications:

- **uniform distribution**, in which each node receives a roughly equivalent amount of data,
- **constraint-based distribution**, in which data is distributed among the nodes according to a set of constraints (e.g., memory availability), and
- **load-dependent distribution**, in which the amount of data allocated to each node is calculated based on its capacity.

Note that the model developed does not cover applications that have a dynamic work distribution independent of the load, i.e., applications for which work distribution 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 Uniform Distribution

When data is uniformly partitioned among the nodes, and all nodes begin execution concurrently, the slowest node determines the time to execute the application. The time to execute the application without contention (X_{ded}) is $\max_a \{1/w_a\}$ time units, and the time to execute the application with contention (X) is given by $\max_a \{sd_a/w_a\}$. Hence from equation (1),

$$sd = \frac{X}{X_{ded}} = \max_a \{sd_a/w_a\} / \max_a \{1/w_a\} . \tag{2}$$

Since $\max_a \{1/w_a\} = 1$, for any node a, equation (2) can be reduced to

$$sd = \max_a \{sd_a/w_a\} . \tag{3}$$

Figure 1 illustrates equation (3). Given that data for application A is distributed uniformly among the nodes, and the nodes are homogeneous, node 1 will determine the time to execute application A since it is the most heavily loaded. In this case, $sd = sd_1 = 3$. However, if the nodes are heterogeneous and node 2 is three times slower than the others, the time to execute application A is determined by node 2, and $sd = (sd_2 / w_2) / (1 / w_2) = 2 / 1 = 2$.

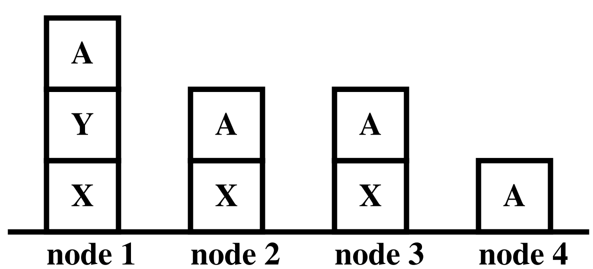

Figure 1: Cluster formed by 4 nodes and shared by three applications: A, X, and Y. Application A's data is divided among the nodes uniformly. Each box represents the amount of data associated with the corresponding node.

3.2 Constraint-Based Distribution

When data is distributed based on a set of constraints such as memory availability, the aggregate slowdown can be calculated from the extra amount of data (relative to the data assigned in a uniform distribution) 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 data distribution were uniform, and the extra work that must be executed by a node because the data distribution 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

$$\begin{aligned} time_a &= (time_{a,ded} \times sd_a) + (time_{a,ded} \times ew_a \times sd_a) \\ &= time_{a,ded} \times sd_a \times (1 + ew_a)\,, \end{aligned} \tag{4}$$

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 data than it would be if the distribution 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 (4), we can calculate the aggregate slowdown as

$$sd = \frac{X}{X_{ded}} = \max_a \left\{ \frac{(1 + ew_a) \times sd_a}{w_a} \right\} / \max_a \left\{ \frac{(1 + ew_a)}{w_a} \right\}, \tag{5}$$

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.

Equation (5) assumes that the targeted application will use the same constraint-based data distribution (given by ew_a) in determining both X and X_{ded}. This may not be always the case, since the data distribution used for X may also be affected by load-dependent constraints, such as memory availability. Therefore, we must allow for different constraint-based distributions to be used for X and X_{ded}. Call these constraint-based distributions d_1 and d_2. To accommodate for d_1 (for X) and d_2 (for X_{ded}), equation (5) is modified to

$$sd = \max_a \left\{ \frac{(1 + ew_{a,1}) \times sd_a}{w_a} \right\} / \max_a \left\{ \frac{(1 + ew_{a,2})}{w_a} \right\}, \tag{6}$$

where

- $ew_{a,y}$ = extra work executed by node *a* with distribution 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 distribution d_y (y = 1, 2).

In the setting where the dedicated time X_{ded} is given for a uniform data distribution, equation (5) can be reduced to

$$sd = \max_a \left\{ \frac{(1 + ew_a) \times sd_a}{w_a} \right\} / \max_a \{ 1/w_a \} \ , \tag{7}$$

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

Since $\max_a \{ 1/w_a \} = 1$, for any node a, equation (7) can be reduced to

$$sd = \max_a \left\{ \frac{(1 + ew_a) \times sd_a}{w_a} \right\} . \tag{8}$$

Figure 2 illustrates the constraint-based distribution. It shows application A's data distribution in a four-node cluster. The application has 12 data units, which are distributed among the nodes according to memory availability. The dotted lines in the figure determine application A's data 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 equation (8), $sd = (1 + 0) \times 2 / 1 = 2$. 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 (8).

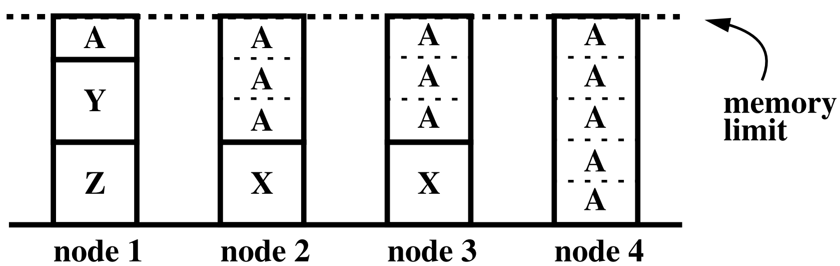

Figure 2: Cluster formed by 4 nodes and shared by three applications: A, X, and Y. Application A's 12 units of data (which are determined by the dotted lines) are divided among the nodes according to memory availability. Each box represents the amount of data associated with the corresponding node.

3.3 Load-Dependent Distribution

In this distribution, data is allocated according to the available computational capacity in the cluster. In this case, data is distributed so that all nodes will finish together (ideally), assuming they all started at the same time. Consequently, we need a different approach (than with the uniform and constraint-based distributions) to calculate the aggregate slowdown imposed on each node.

For this distribution, 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

$$sd = \frac{aggregate\ capacity_{ded}}{aggregate\ capacity} = \left(\sum_{a=1}^{n} w_a\right) \Big/ \left(\sum_{a=1}^{n} \frac{w_a}{sd_a}\right), \tag{9}$$

Figure 3 illustrates a load-dependent distribution. The data for application A is distributed 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, then sd = (2 + 2 + 2 + 1) / (2 / 2 + 2 / 2 + 2 / 3 + 1 / 1) = 7 / 3.67 = 1.9 according to equation (9).

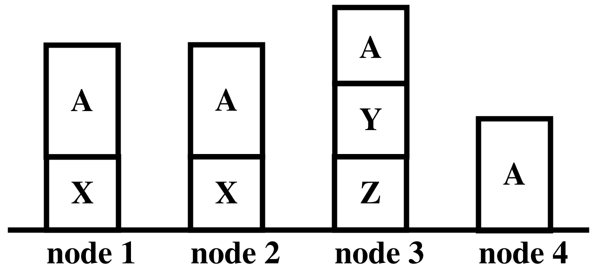

Figure 3: Cluster formed by 4 heterogeneous nodes and shared by three applications: A, X, and Y. Application A's data is divided among the nodes according to their computational capacity. Each box represents the amount of data associated with the corresponding node.

4. Experiments

The formulas presented in the previous section provide a model for the aggregate slowdown. To assess the accuracy of these formulas, we performed a collection of experiments on a range of CPU-bound benchmarks, commonly found in high-performance scientific applications. In these experiments, we compared actual execution times for an application on a cluster with synthetic load with modeled execution times calculated from dedicated execution times and the aggregate slowdown factor. The synthetic load was formed by a combination of serial and/or CPU-bound data-parallel applications distributed over the cluster. Our experiments showed the modeled execution times to correlate with actual execution times within 15% for the three data distributions. This error reflects the fact that although local scheduling policies are reduced to round-robin for coarse-grained applications, they do not reduce to a "perfect" round-robin. In fact, the error obtained correlates inversely with the granularity of both the targeted application and the competing applications.

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.

4.1 Experiments on the Homogeneous Cluster

Figure 4 illustrates a representative experiment using a *uniform* data distribution policy. The experiment shown is a data-parallel Jacobi3D application [5] (developed using KeLP [10] and MPI [12]) which is executed for different problem sizes (given by $N \times N \times N$). The figure shows modeled and measured times for execution on 4 nodes of the DEC Alpha-Farm, where other applications are also executing. The data is distributed uniformly among the nodes. In experiment 1 (*exp 1*), one additional CPU-bound data-parallel application executes on two of the nodes, imposing a slowdown of 2 on the execution of the Jacobi3D application. In experiment 2 (*exp 2*), one additional CPU-bound data-parallel application executes on two of the nodes, one of which is also shared by a serial CPU-bound application. This situation is illustrated in Figure 1. In this case, the Jacobi3D algorithm was slowed by a factor of 3 on the cluster.

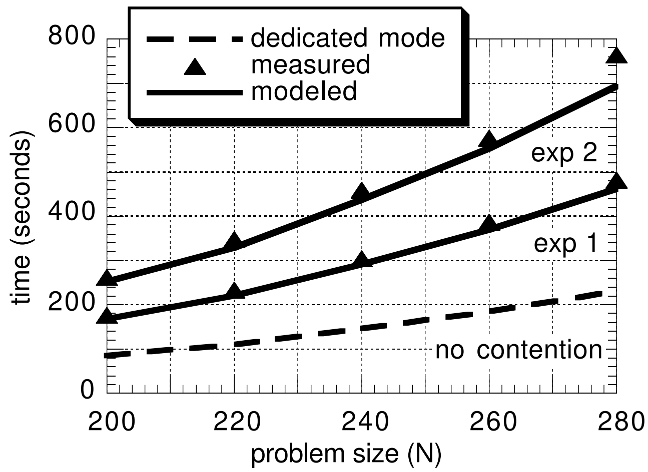

Figure 4: Time to execute the Jacobi3D algorithm on 4 nodes of the DEC Alpha-Farm in dedicated mode and with 2 different loads.

Figure 5 represents experiments using a *constraint-based* data distribution policy. Shown is a Multigrid application [5] (developed using KeLP [10] and MPI [12]) 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 distribution shown in Table 1 (column *f*). Table 1 also shows the other parameters (*sd* and *ew*) used to calculate the aggregate slowdown (3.0).

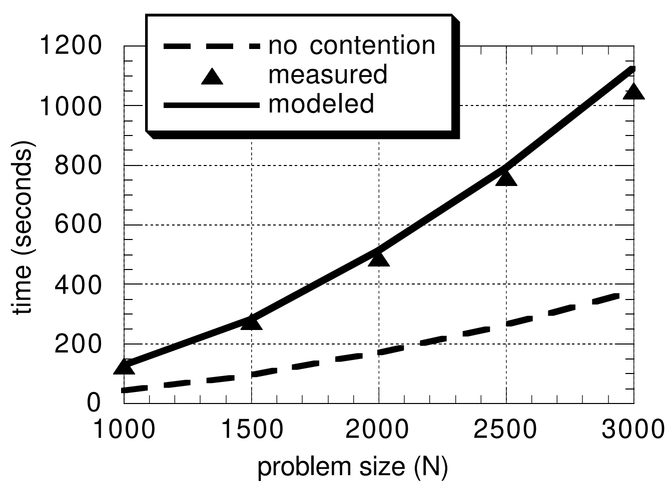

Figure 5: Time to execute the Multigrid application on 4 nodes of the DEC Alpha-Farm, with constraint-based distribution, in dedicated mode and with contention on the nodes.

Table 1: Local slowdown in each node of a dedicated 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

Figure 6 illustrates experiments with the *load-dependent* data allocation policy. Shown is a representative SOR application [5] developed using PVM [14] and executed on 4 nodes of the DEC Alpha-Farm. In both experiments data for the SOR application was distributed among the nodes according to the load. In experiment 1 (*exp 1*), there is also 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 3. 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.

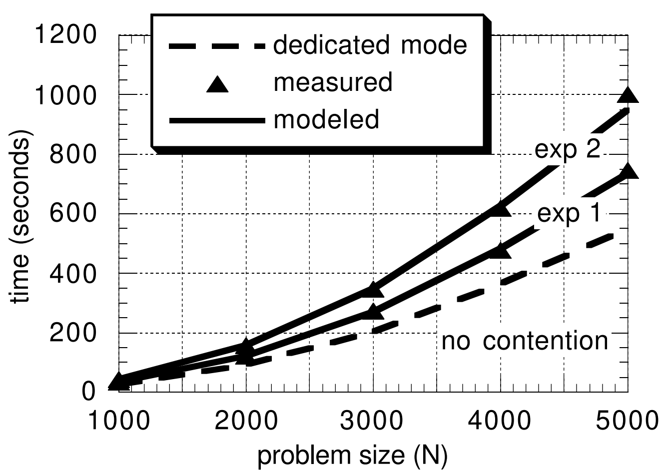

Figure 6: Time to execute the SOR algorithm on 4 nodes of the DEC Alpha-Farm in dedicated mode and with 2 different loads.

4.2 Experiments on the Heterogeneous Cluster

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

Figure 7 represents experiments using a *uniform* data distribution policy on a heterogeneous cluster. The experiment shown is a data-parallel Jacobi2D application [5] (developed using KeLP [10] and MPI [12]) executing for different problem sizes (given by $N \times N$) on the four nodes of a cluster consisting of two DEC Alphas and two IBM RS-6000s. One of the DEC Alphas and one of the IBM RS-6000s are also used to execute a CPU-bound data-parallel application (whose data is distributed according to the computational capacity of the nodes), causing the aggregate slowdown to be 2.

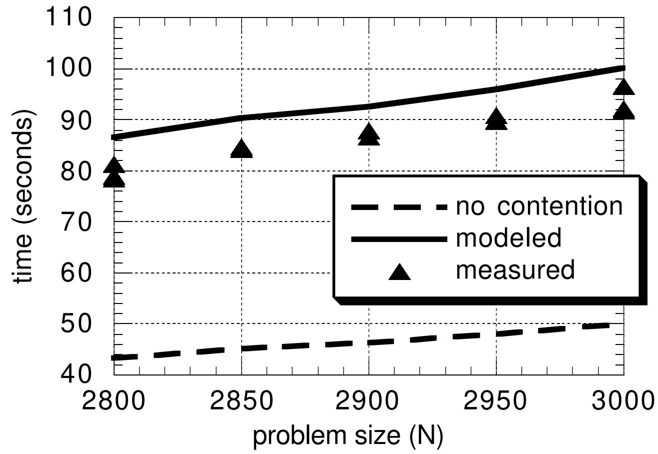

Figure 7: Times for the Jacobi2D application executing with uniform work distribution on a heterogeneous cluster in dedicated mode and with contention.

Figure 8 represents experiments using a *constraint-based* data distribution policy. Shown is a representative SOR application [5] developed using PVM [14] 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 distribution shown in Table 2 (column *f*).

Table 2: Work distribution for the application in dedicated mode

Node	w	f	$(1 + ew_i) / w_i$
$alpha_1$	3.07	1 / 6 (17%)	0.22
$alpha_2$	3.07	1 / 3 (33%)	0.43
rs_1	1	1 / 6 (17%)	0.66
rs_2	1	1 / 3 (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 3 shows the data distribution (*f*) and slowdown (*sd*) parameters used in this experiment. According to Table 2 and Table 3, the aggregate slowdown for this case is 2.67 / 1.33 = 2.01.

Table 3: Data for experiment 1 in Figure 8

Node	w	sd	f	$(1 + ew_i) \times sd_i / w_i$
$alpha_1$	3.07	1	1 / 6 (17%)	0.22
$alpha_2$	3.07	1	1 / 3 (33%)	0.43
rs_1	1	2	1 / 3 (33%)	**2.67**
rs_2	1	2	1 / 6 (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. Table 4 shows the data distribution (*f*) and slowdown (*sd*) used in this experiment.

According to Table 2 and Table 4, 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 4: Data for experiment 2 in Figure 8

Node	w	sd	f	$(1 + ew_i) \times sd_i / w_i$
$alpha_1$	3.07	4	15 / 55 (27%)	1.42
$alpha_2$	3.07	1.33	15 / 55 (27%)	0.47
rs_1	1	3	15 / 55 (27%)	**3.27**
rs_2	1	2	10 / 55 (19%)	1.45

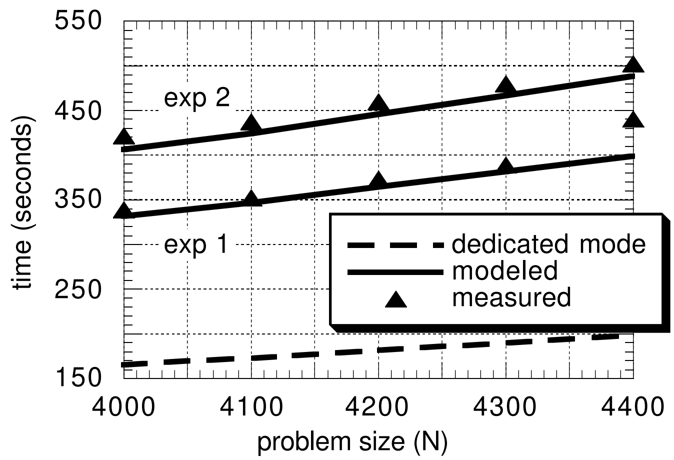

Figure 8: Times for two experiments with the SOR algorithm executing with constraint-based data distribution on a heterogeneous cluster in dedicated mode and with contention.

Figure 9 represents experiments using a *load-dependent* data distribution policy with a Genetic Algorithm application [15] developed using PVM [14]. Shown are modeled and measured times for execution with different problem sizes (given by population size) on four nodes of a heterogeneous cluster consisting of two DEC Alphas and two IBM RS-6000s. In this experiment, the IBM RS-6000s are also executing a CPU-bound data-parallel application. The aggregate slowdown is 1.18.

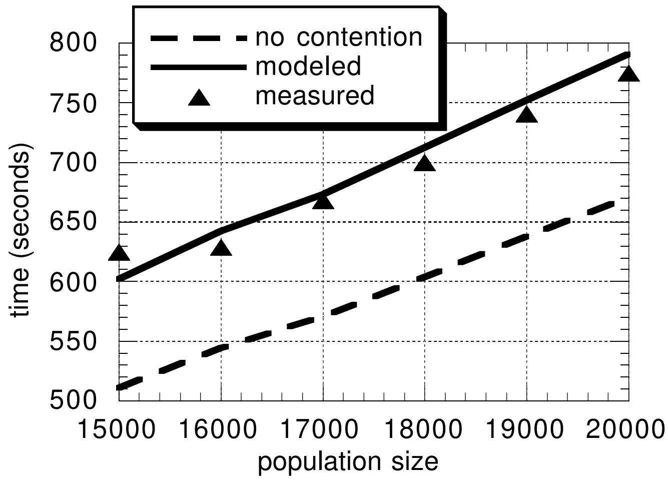

Figure 9: Times for the Genetic Algorithm executing with load-dependent work distribution on a heterogeneous cluster in dedicated mode and under contention.

5. 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 requirement for performance-efficient scheduling. The model is parameterized by the policy used to distribute application data, 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 model 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 data allocation policy and was developed here for three common data allocation policies (uniform, constraint-based, and load-dependent). 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 in the performance of an individual application due to contention on the cluster.

Contention can perturb application performance dramatically and must be accounted for in the prediction of performance. The contention model developed here provides a fundamental building block for extensions in which the job mix can be varied further and the effects of out-of-core memory accesses may be included. Such extensions would increase the accuracy of the determination of application slowdown, and improve the prediction of overall application performance in shared heterogeneous clusters.

References

[1] 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.

[2] F. Berman, R. Wolski, S. Figueira, J. Schopf, and G. Shao, “Application-Level Scheduling on Distributed Heterogeneous Networks”, in *Proceedings of Supercomputing’96*, November 1996.

[3] F. Berman and R. Wolski, “Scheduling from the Perspective of the Application”, in *Proceedings of the Fifth International Symposium on High-Performance Distributed Computing*, pp. 100–111, August 1996.

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

[9] S. M. Figueira and F. Berman, "Modeling the Effects of Contention on the Performance of Heterogeneous Applications," in *Proceedings of the Fifth International Symposium on High-Performance Distributed Computing*, pp. 392–401, August 1996.

[10] 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.

[11] 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.

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

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

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

[15] 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.