Using Stochastic Intervals to Predict Application Behavior on Contended Resources

Jennifer M. Schopf*
Computer Science Department
Northwestern University
jms@cs.nwu.edu

Francine Berman†
Dept. of Computer Science and Engineering
University of California, San Diego
berman@cs.ucsd.edu

Abstract

Current distributed parallel platforms can provide the resources required to execute a scientific application efficiently. However, when these platforms are shared by multiple users, performance prediction becomes increasingly difficult due to the dynamic behavior of the system.

This paper addresses the use of stochastic values, represented by intervals, to parameterize performance models. We describe a method for using upper and lower bound information to parameterize application prediction models in order to make better predictions about the application's behavior in a contentious environment. We demonstrate this technique for a set of 3 applications under different workloads on a production network of workstations.

1 Motivation and Outline

In order to achieve performance in multi-user distributed environments, it is critical to provide performance models which accurately represent the execution behavior of programs in contended systems. In [15], we explored the accuracy of structural performance models which allowed for stochastic input parameters. Stochastic values allow performance models to be parameterized by a range of possible values., Such values more accurately represent parameters of system and application behavior. For example, bandwidth cold be reported as 6-8 Mbits/second rather than a single (point) value (for example, and average).

In previous work, we investigated the accuracy of structural models when parameterized by stochastic values that could be adequately represented by normal distributions. In this paper, we loosen the requirement and consider structural performance models which are parameterized by simple intervals. We demonstrate for three applications that interval predictions resulting from structural models parameterized by interval values provide a good prediction of observable behavior in multi-user distributed environments.

In this paper we define an extension to a modeling technique to allow for model parameters that are stochastic values represented by an upper bound and a lower bound. Section 2 gives a brief overview of the structural modeling approach. Section 3 describes how intervals are defined, and details interval arithmetic. This results in application performance predictions that are stochastic predictions, and more accurately describe the application behavior, as shown experimentally in Section 4. Section 4.5 compares this approach to previous work that used normal distributions to represent stochastic values, and other related work. We conclude and give future work in Section 6.

2 Overview of Structural Modeling

In previous work [13, 14] we developed a technique called **structural modeling** as an approach to accurate performance modeling. Structural modeling uses the functional structure of the application to define a set of equations that reflect the time-dependent, dynamic mix of application tasks occurring during execution in a distributed parallel environment. A structural model consists of a top-level model, component models, and input parameters. A top-level model is a **performance equation** consisting of component models and composition operators. Each component model is also defined to be a performance equation, parameterized by platform and application characteristics, such as benchmarks, dynamic system measurements, problem size, or the number of iterations to be executed. The output of a structural model is a predicted execution time. Structural models can also be developed for alternative performance metrics, such as speedup or response time.

A good performance model, like a good scientific theory, is able to explain available observations and predict future events, while abstracting away unimportant details [5]. Structural models are represented by equations that allow us to abstract away details into the top level models and the

*Partially supported by NASA GSRP Grant #NGT-1-52133, Darpa grant #N66001-97-C-8531.

†Supported in part by NSF grant #ASC-9701333, Darpa grant #N66001-97-C-8531.

parameters, but still accurately model the performance by showing the inter-relation of application and system factors through the component models and their parameterizations. If the predictions are for use in a timely manner – for example, as part of a runtime scheduling approach – they must be able to be evaluated quickly and with minimal computational overhead as well, a factor that we address throughout.

In order to use stochastic values, not only do we need to describe how to define them, but we must define the needed arithmetic. In the following we assume that accurate structural models are available to the user. Section 4.5 describes some possible side effects if this is not the case.

3 Defining Stochastic Values using Intervals

Once an accurate structural model has been defined, several practical issues must be addressed in order to parameterize it with stochastic values. In this section we discuss how dynamic information can be used to define a stochastic value using an upper bound and a lower bound, and the needed arithmetic to combine the values.

3.1 Defining an Interval

Given data in the form of a time-series, the simplest way to represent the variability of a stochastic value is as an interval. A similar approach was used for queuing network models in [8]. We define the **interval** of a stochastic value X to be the tuple

$$X = [\underline{x}, \overline{x}] \quad where \quad \{x \in X | \underline{x} \leq x \leq \overline{x}\} \qquad (1)$$

The values $\underline{x}$ and $\overline{x}$ are called the **endpoints** of the interval. The value $\underline{x}$ is the minimum value over all $x \in X$ and is called the **lower bound**, and the value $\overline{x}$ is the maximum value, called the **upper bound**. The **size** of an interval $X = [\underline{x}, \overline{x}]$ is defined as $|X| = \overline{x} - \underline{x}$.

The main advantage of using intervals is the simplicity and intuitiveness of the approach. Determining the maximum and minimum value of the interval for a stochastic value is almost always possible, and correlates to the intuition behind defining a lower and upper bound for a value.

The primary disadvantage of using intervals is that a given interval may need to be a very large to account for outlying values. Figure 1 shows four possible stochastic values, all with very different distributions, all represented by the same interval. For example, a stochastic value with two or more "modes", such as shown in lines two and three of Figure 1, would be represented by the same interval as a value with a single mode.

Another important concern is the **sharpness** of the defined intervals [12]. A set of bounds $[l, u]$ are sharper than another set $[l', u']$ if $l' \leq l$ and $u' \geq u$, where both $[l, u]$ and $[l', u']$ contain all values of some stochastic value X. Sharp, or **tight**, bounds on the resulting intervals of a computation are especially important for predictions used for scheduling, where tighter bounds on a predicted execution time can lead to a more efficient execution time. This is a different issue than the large interval resulting from outliers. In that case, and for all the intervals depicted in Figure 1, the bounds around the data are tight around the data, where an unsharp interval is one that is "loose" around the data.

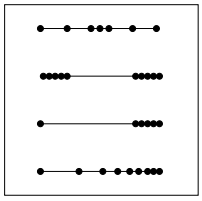

Figure 1. Four sample stochastic values, each with the same interval representation.

3.2 Arithmetic over Intervals

Combining stochastic values represented as intervals involves obeying well defined rules for computing interval arithmetic [12]. Formulas are given in Tables 1, 2, and 3. The result of arithmetic on intervals in structural models is an interval which provides an upper bound and lower bound on the prediction.

The standard interval arithmetic formulas are also defined assuming any two values from the same stochastic value may be correlated. That is, a mutual or reciprocal relation exists between them, so each should be treated as a separate variable in the range of possible values. This may lead to the overestimation of apparently simple expressions. For example, if A=[$\underline{a}$, $\overline{a}$], then A-A = [$\underline{a}$ - $\overline{a}$, $\overline{a}$ - $\underline{a}$], instead of [0,0]. More concretely, if A = [5,7], A-A = [-2,2]. This is also called the *dependence* or *simultaneity* problem [12].

In some cases, this problem can be mitigated by rewriting the expression to avoid multiple occurrences of arguments [8]. For example, the expression $f(X,Y) = \frac{X-Y}{X+Y}$ for interval values X and Y may be rewritten as $1 - \frac{2}{1+X/Y}$ if $0 \notin Y$. If the expression is evaluated in the latter form, it results in the exact range of $f(x,y)$ for $x \in X$ and $y \in Y$. Evaluating this expression with the intervals X = [4, 5], Y = [1,2] results in [2/7, 4/5] using the first formula, and [1/3, 2/3] using the second.

Addition	x + y = [$\underline{x}$ + $\underline{y}$, $\overline{x}$ + $\overline{y}$]
Subtraction	x - y = [$\underline{x}$ - $\overline{y}$, $\overline{x}$ - $\underline{y}$]

Table 1. Addition and Subtraction over interval values, x = [$\underline{x}$, $\overline{x}$], y=[$\underline{y}$, $\overline{y}$].

Despite these possible disadvantages due to the assumptions needed to use standard interval arithmetic, this approach is overwhelmingly appealing due to the low overhead in acquiring the needed information and it's intuitive appeal.

	$y \geq 0$	$y \ni 0$	$y \leq 0$
$x \geq 0$	$[\underline{xy}, \overline{xy}]$	$[\overline{x}\underline{y}, \overline{xy}]$	$[\overline{x}\underline{y}, \underline{x}\overline{y}]$
$x \ni 0$	$[\underline{x}\overline{y}, \overline{xy}]$	$[\min(\underline{x}\overline{y}, \overline{x}\underline{y}), \max(\underline{xy}, \overline{xy})]$	$[\overline{x}\underline{y}, \underline{x}\underline{y}]$
$x \leq 0$	$[\underline{x}\overline{y}, \overline{x}\underline{y}]$	$[\underline{x}\overline{y}, \underline{xy}]$	$[\overline{xy}, \underline{x}\underline{y}]$

Table 2. Multiplication of interval values x = [$\underline{x}$,$\overline{x}$] and y = [$\underline{y}$,$\overline{y}$].

	$y > 0$	$y < 0$
$x \geq 0$	$[\underline{x}/\overline{y}, \overline{x}/\underline{y}]$	$[\overline{x}/\overline{y}, \underline{x}/\underline{y}]$
$x \ni 0$	$[\underline{x}/\underline{y}, \overline{x}/\underline{y}]$	$[\overline{x}/\overline{y}, \underline{x}/\overline{y}]$
$x \leq 0$	$[\underline{x}/\underline{y}, \overline{x}/\overline{y}]$	$[\overline{x}/\underline{y}, \underline{x}/\overline{y}]$

Table 3. Division of interval values x = [$\underline{x}$,$\overline{x}$] and y = [$\underline{y}$,$\overline{y}$]. Note that interval division of $\frac{x}{y}$ is only defined if 0 is not in y.

4 Experimental Verification

The following experiments demonstrate that interval predictions can accurately capture program behavior in multi-user distributed environments with reasonable sharpness.

In the experiments, the stochastic CPU information used in the models was supplied by the Network Weather Service [17, 18, 19]. In categorizing the multiple background workloads seen for each experiment set, we define a machine with a **low** variation in CPU availability as having a variation of 0.25 or less, on a scale from 0 to 1, a **medium** variation when the CPU availability varied more than 0.25, but less than 0.5, and a **high** variation when the CPU values varied over half the range.

We examined three applications on a distributed network of Sparc workstations located in the UCSD Parallel Computation Lab. The machines are connected with a mix of 10 Mbit (slow) and 100 Mbit (fast) ethernet over an Intel switch, and all run Solaris. Both the CPU and the network were shared with other users. Additional experiments are given in [14].

4.1 Performance Metrics

To evaluate the stochastic predictions, we use two different performance metrics. The first we call **capture**. This measures the percentage of actual values falling within the predicted range for a set of application runs. This statistic is important because the primary goal is to define predictions that are accurate in estimating the execution behavior of the application under contention. However, this statistic can be misleading since one reason a prediction may capture the majority of the values is due to the fact that the bounds were large, or the interval was not sharp.

Our second metric is called **sharpness**. It is important to not only predict a range of behaviors that reflects the actual execution behavior range, but this range should be as tight as possible in order to have more accurate information. We define the sharpness metric by dividing the predicted range into five equal parts, as shown in Figure 2. If the majority of the values for a set of runs fall in the center portion, the values could have been much tighter. If they fall into the mid ranges, the prediction could have been somewhat sharper, and if they fall into the outer ranges, they could not have been sharper without sacrificing the capture proportion. In addition, we can use this metric to evaluate if there was a bias in the prediction, that is, if the majority of the predictions were too high or too low. Bias can be caused by outliers that don't reoccur, or by a bias in the original model. These values for the entire set of experiments are given in Table 4.

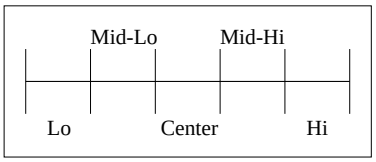

Figure 2. Sharpness intervals.

Figure	Under	Lo	Mid Lo	Center	Mid Lo	Hi	Over
SOR1 Figure 3	0	0	8	5	5	6	1
SOR2 Figure 4	0	2	6	8	4	3	3
GA 1 Figure 5	0	0	2	8	5	3	6
GA2 Figure 6	0	3	9	9	1	3	9
LU 1 Figure 7	0	0	8	9	2	1	0
LU2 Figure 8	0	2	10	9	0	1	0

Table 4. Sharpness table for experiments, labeled for area of interval where actual values fell. Under and Over indicate actual values that were not captured by the predictions.

4.2 Application 1 - Successive Over-Relaxation Code

Successive Over-Relaxation (SOR) is a Regular SPMD code that solves Laplace's equation. Our implementation uses a red-black stencil approach where the calculation of each point in a grid at time t is dependent on the values in a stencil around it at time $t-1$. The application is divided into "red" and "black" phases, with communication and computation alternating for each. In our implementation, these two phases repeat for a predefined number of iterations.

Figure 3 shows the interval stochastic value predictions and the actual time of the SOR application, run when the PCL cluster had two processors with a highly variable CPU availability, one with a medium variable availability, and one with a low variable availability. Using interval representations, we capture 26 of 27 values. From the sharpness information given in Table 4, we see a slight bias to the higher values, but as this is not seen for the other SOR data sets, it is likely due to an outlier value for the CPU availability. From this information we also see that these values have a good sharpness to them, since decreasing the interval would likely lower the capture percentage significantly.

A second set of experiments for the SOR code on the PCL cluster are presented in Figure 4, run when the PCL cluster had one high variable availability, two medium variable availability, and one low variable availability. These experiments show the benefits of stochastic predictions when the execution times exhibit an extreme variance, in this case variance production over 300%. For these runs, the interval representation captured 24 of the 27 values. From the sharpness table, the actual values are spread across the entire range indicating that tighter bounds would adversely affect the capture percentage, indicating a good fit in terms of sharpness.

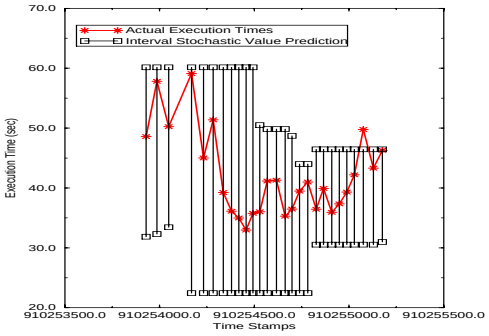

Figure 3. SOR1- Interval stochastic value prediction for the SOR benchmark.

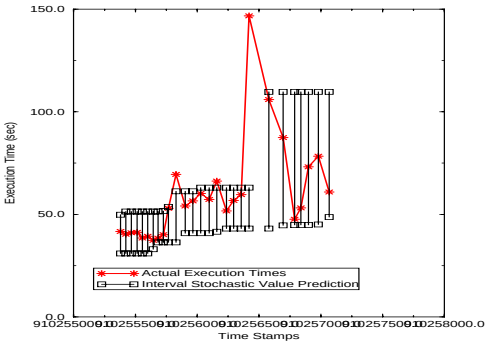

Figure 4. SOR2- Interval stochastic value prediction for the SOR benchmark.

4.3 Application 2 - Genetic Algorithm Code

We implemented a genetic algorithm (GA) heuristic for the Traveling Salesman Problem (TSP) [6, 16]. Our distributed implementation of this genetic algorithm uses a global population and synchronizes between generations [3]. All of the Slaves operate on a global population (each member of the population is a solution to the TSP for a given set of cities) that is broadcast to them. Each Slave works in isolation to create a specified number of children (representing new tours), and to evaluate them. Once all the sets of children are received by the Master, they are sorted (by efficiency of the tour), some percentage are chosen to be the next generation, and the cycle begins again.

GA1, Figure 5, shows the interval stochastic value predictions and the actual time, run when the two machines on the cluster had a highly variable availability, and two had low variability. The predictions captured 19 of 25 values. GA2, Figure 6, shows the interval stochastic value predictions and the actual time run when the PCL cluster had two highly variable machines and two low variability machines. The interval representation captured 16 of 25 values. In terms of sharpness, it appears that the intervals may have a bias associated with them towards underestimation, or may be too tight, and do not take into account the high variability in the dedicated performance of this code in GA2.

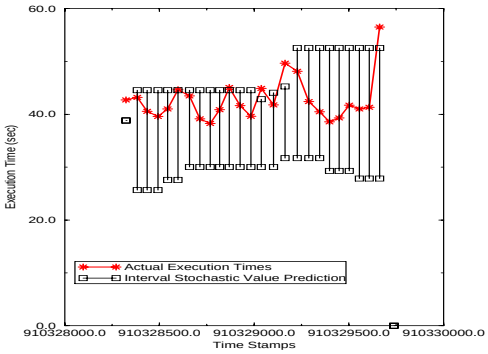

Figure 5. GA1- Interval stochastic value prediction for the GA code.

4.4 Application 3 - LU Benchmark

The LU benchmark is a simulated CFD application that solves a block-lower-triangular/block-upper-triangular system of equations, and is one of the NAS Parallel Benchmarks (NPB) [1]. This system of equations is the result of an unfactored implicit finite-difference discretization of the Navier-Stokes equations in three dimensions. The LU benchmark finds lower triangular and upper triangular matrixes such that $L \cdot U = A$ for an original matrix A. It has the feature that it tests the communication aspects of the system well by sending a relatively large number of 5 word messages.

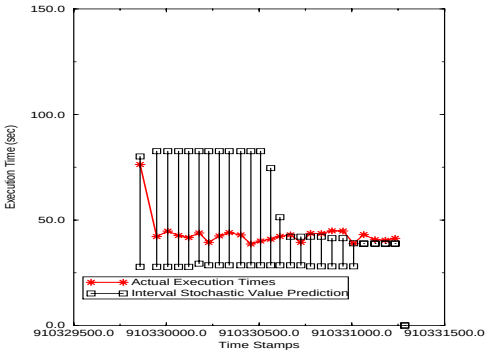

Figure 6. GA2- Interval stochastic value prediction for the GA code.

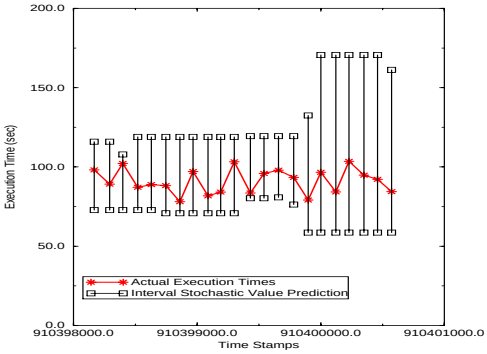

Figure 8. LU2- Interval stochastic value prediction for the LU benchmark.

LU1, Figure 7, shows the interval stochastic value predictions and the actual time, run when two of the machines showed a high variability in availability, and two had a low variability. The interval representation captured 21 of the 25 values. LU2, Figure 8, shows the interval stochastic value predictions and the actual time, run when the PCL cluster one high variability machine, one medium variability machine and two low variability machines. The interval representation captured 22 of the 25 values. The intervals could be sharper, although no bias is evident. Again, we feel that this may be due to the highly predictable dedicated performance.

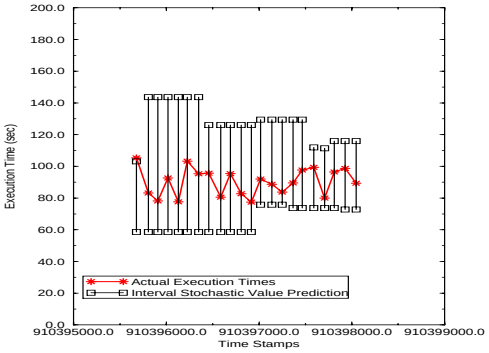

Figure 7. LU1- Interval stochastic value prediction for the LU benchmark.

4.5 Discussion

In summary, for the majority of the experiments, we achieved predictions that captured the majority of the execution behavior using interval representations for the stochastic information. We saw that the sharpness of the interval could be affected by both the predictability of the application in a dedicated setting, the variation in available CPU, and any bias in the original model. We postulate that the time frame used for the range of values used to estimate the available CPU should be adjusted by these values, instead of fixed in an application dependent manner. This is a topic for future work.

5 Related Work

In previous work [15, 14] we investigated using a normal distribution instead of an upper bound and lower bound to make predictions in contentions environments. This had the advantage of being able to ignore outlying values, since we used a prediction ranged based on the mean plus or minus two standard deviations, with the intention of capturing only 95% of the values in the best case. However, this methodology was complicated by the fact that many system characteristics, available CPU for one, do not have a normal distribution, thus leading to errors in the predictions, as well as the more complicated arithmetic involved when handling special cases like modal distributions.

There are several related approaches. Note that stochastic values as defined here are not related to Petri net models [9, 11], also called “stochastic models”, in any way except through the application of conventional statistical techniques.

Some researchers are beginning to use probabilistic techniques to represent data for predictions of application performance in distributed environments. Brasileiro et al. [4] use probability distribution functions to calculate wait times on a token ring network. This work borrows heavily from queuing theory in a much more theoretical setting than our production setting.

The most closely related work to our approach is by Mendes as part of the Rice Fortran D95 compiler [10]. They generate symbolic performance predictions using a data parallel compiler. The compiler generates upper and lower bounds for predicted execution time by considering extrema of system-dependent constants (eg. memory references times). This work assumes some baseline system

benchmarks, but at a much lower level than we expect, and cannot take into account dynamic runtime information except as user-provided estimates.

Our work is somewhat related to "quality of information" such as timeliness and accuracy is used in metacomputing [2, 7]. These approaches have begun to address the need for additional information in order to make educated decisions about performance in a cluster environment with dynamic load. Using a stochastic value to represent the range of values possible from some system or application characteristic is one step in this direction.

6 Summary and Future Work

In this paper, we describe a new approach to application performance prediction in multi-user (production) environments. We have defined stochastic values to reflect a likely range of behavior for model parameters, and extended the definition of structural models to allow for stochastic parameters as well as stochastic performance predictions. We describe the representation of stochastic values using an upper bound and a lower bound. Our experiments demonstrate that in production settings, stochastic values can accurately identify the range of application execution behavior, providing more comprehensive and more accurate information about application execution than point values.

Continuing work in this area includes analysis to help further determine a proper time frame over which to determine the intervals, as well as how to use stochastic models in stochastic scheduling techniques. We are also analyzing the advantages and disadvantages of using histograms to represent stochastic values as a way to combine the arithmetic efficiency of intervals with the shape detail of distributions.

References

[1] D. Bailey, T. Harris, W. Saphir, R. van der Wijngaart, A. Woo, and M. Yarrow. The nas parallel benchmarks 2.0. Technical Report Report NAS-95-020, Numerical Aerospace Simulation Facility at NASA Ames Research Center, December 1995. Also available at `http://science.nas.nasa.gov/Software/NPB/Specs/npb2_0/npb2_0.html`.

[2] F. Berman, R. Wolski, and J. Schopf. Performance prediction engineering for metacomputing. `www.cs.ucsd.edu/groups/hpcl/apples/PPE/index.html`, 1997.

[3] K. Bhatia. Personal communication, 1996.

[4] M. A. G. Brasileiro, J. A. Field, and A. B. Moura. Modeling and analysis of time critical applications on local area networks. In *Proc. of Chilean Computer Science Society*, pages 459–71, 1991.

[5] I. Foster. *Designing and Building Parallel Programs*. Addison-Wesley, 1995.

[6] Lawler, Lenstra, Kan, and Shmoys. *The Traveling Salesman Problem*. John Wiley & Sons, 1985.

[7] C. Lee, C. Kesselman, J. Stepanek, R. Lindell, S. Hwang, B. S. Michel, J. Bannister, I. Foster, and A. Roy. The quality of service component for the globus metacomputing system. In *Proceedings of IWQoS '98*, pages 140–142, 1998.

[8] J. Lüthi and G. Haring. Mean value analysis for queuing network models with intervals as input parameters. Technical Report TR-950101, Institute of applied science and information systems, university of Vienna, July 1995.

[9] M. A. Marson, G. Balbo, and G. Conte. A class of generalized stochastic petri nets for the performance analysis of multiprocessor systems. *ACM TOCS*, pages 93–122, May 1984.

[10] C. L. Mendes and D. A. Reed. Integrated compilation and scalability analysis for parallel systems. In *Proceedings of PACT '98*, 1998.

[11] M. Molloy. Performance analysis using stochastic petri nets. *IEEE Transactions on Parallel and Distributed Systems*, C-31:913–7, September 1990.

[12] A. Neumaier. *Interval methods for systems of equations*, chapter Basic Properties of Interval Arithmetic. Cambridge University Press, 1990.

[13] J. M. Schopf. Structural prediction models for high-performance distributed applications. In *CCC '97*, 1997. Also available as `www.cs.ucsd.edu/users/jenny/CCC97/index.html`.

[14] J. M. Schopf. *Performance Prediction and Scheduling for Parallel Applications on Multi-User Clusters*. PhD thesis, University of California, San Diego, 1998. Also available as UCSD CS Dept. Technical Report, Number CS98-607, `www.cs.nwu.edu/~jms/Thesis/thesis.html`.

[15] J. M. Schopf and F. Berman. Performance prediction in production environments. In *Proceedings of IPPS/SPDP '98*, 1997.

[16] D. Whitley, T. Starkweather, and D. Fuquay. Scheduling problems and traveling salesman: The genetic edge recombination operator. In *Proceedings of International Conference on Genetic Algorithms*, 1989.

[17] R. Wolski. Dynamically forecasting network performance using the network weather service(to appear in the journal of cluster computing). Technical Report TR-CS96-494, UCSD, CSE Dept., 1996.

[18] R. Wolski. Dynamically forecasting network performance to support dynamic scheduling using the network weather service. In *Proc. 6th IEEE Symp. on High Performance Distributed Computing*, August 1997.

[19] R. Wolski, N. Spring, and J. Hayes. Predicting the cpu availability of time-shared unix systems. In *submitted to SIGMETRICS '99 (also available as UCSD Technical Report Number CS98-602)*, 1998.