International Journal of Foundations of Computer Science

USING STOCHASTIC INFORMATION TO PREDICT APPLICATION BEHAVIOR ON CONTENDED RESOURCES

JENNIFER M. SCHOPF
Computer Science Department, Northwestern University
Evanston, IL 60626 USA

and

FRANCINE BERMAN
Department of Computer Science and Engineering, University of California, San Diego
La Jolla, CA 92093 USA

Received (received date)
Revised (revised date)
Communicated by Editor's name

ABSTRACT

Prediction is a critical component in the achievement of application execution performance. The development of adequate and accurate prediction models is especially difficult in local-area clustered environments where resources are distributed and performance varies due to the presence of other users in the system. This paper discusses the use of stochastic values to parameterize cluster application performance models. Stochastic values represent a range of likely behavior and can be used effectively as model parameters. We describe two representations for stochastic model parameters and demonstrate their effectiveness in predicting the behavior of several applications under different workloads on a contended network of workstations.

Keywords: Parallel distributed computing, cluster computing, performance modeling, performance prediction, scheduling, stochastic values.

1. Motivation

Application performance prediction models are a fundamental building block to achieving good application execution times. The purpose of such models is to provide a prediction of application performance using environment and application characteristics as parameters, such as bandwidth, available CPU, message size, operation counts, etc. Model parameters are generally assumed to have a single likely value, which we refer to as a **point value**. For example, a point value for bandwidth might be 7 Mbits/second.

However, in practice, point-valued model parameters are often a best guess, an estimate under ideal circumstances, or a value that is accurate only for specific time and load conditions. In environments in which model parameters are likely to vary, it may be more useful to represent system and application characteristics as a **distribution** of possible values over a range; for example, bandwidth might be reported as having a mean of 7 Mbits/second with a standard deviation of 1 Mbit/second. We refer to values that can be represented by distributional data as **stochastic values**. Whereas a point value gives a single value for a quantity, a stochastic value gives a set of possible values weighted by probabilities to represent a range of likely behavior [35]. More precisely, we define a stochastic value X to be the the set of values $\{x_1, \ldots, x_n\}$ with the associated function $F(x_i) = p_i$, where p_i is the probability associated with x_i, for all values $x_i \in X$. Note that $\sum_i p_i = 1$. We call a prediction that can be represented as a distribution of execution times a **stochastic prediction**. Such predictions can be calculated by using stochastic values to parameterize performance prediction models.

In [31], we demonstrated that stochastic predictions could be used effectively by schedulers to improve the performance of applications in clustered, contended environments. In this paper, we focus on two possible representations to be used as inputs for a structural modeling framework that can generate stochastic predictions: using *normal distributions* and using an *interval* representation. Each of these representations has advantages and disadvantages that will be discussed.

To make it feasible to use stochastic values for performance predictions, the performance prediction model must be able to accept parameters that are not simple point values. This means that the model must also be able to combine the parameters to determine the model prediction. In particular, we may need to define arithmetic and collective operation functions to combine stochastic and point values given a particular representation.

This paper is organized as follows: Section 2 introduces a performance modeling framework, called structural modeling, that allows for stochastic-valued model parameters and can generate stochastic-valued predictions. In Section 3 we address representing stochastic values as normal distributions, and describe how to combine such values arithmetically. In Section 4 we describe an alternative representation of stochastic values that uses intervals. In Section 5, we compare these two representations with respect to the accuracy of the resulting performance predictions using several applications in various multi-user environments. Related work is presented in 6. We conclude in Section 7.

2. Overview of Structural Modeling

In previous work [29, 30] we developed a technique called **structural modeling** that uses the functional structure of the application to define a set of equations to reflect the 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, for example, in Equation 1 below. Each component model is also defined, possibly recursively, 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, as shown in Equations 2- 7 below. 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.

As an example, a structural model for a distributed red-black Successive Over-Relaxation (SOR) calculation is given below. SOR is a Regular SPMD code that solves Laplace's equation. Our distributed implementation is based on 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, and has stochastic values for the CPU and bandwidth information.

$$\begin{aligned} \textbf{SORExTime} \; = & \\ \textit{NumIts}\,(\quad \textit{Max}\,[& \quad \textbf{RedComp}(i) + \textbf{RedComm}(i) + \\ & \quad \textbf{BlackComp}(i) + \textbf{BlackComm}(i)\,]\,) \end{aligned} \tag{1}$$

where

- *NumIts* : The number of iterations.
- *Max* : A maximum function.
- **RedComp**(i), **BlackComp**(i): Execution time for the Red/Black Computation phase on processor i, for example,

  $$\textbf{RedCompBM}(i) = NumElts(i) * BM(\texttt{Red}, i) \tag{2}$$

  with

  - $NumElt(i)$: Number of elements calculated by processor i.
  - $BM(\texttt{TASK}, i)$: Time for processor i to compute task `TASK` for one element.

- **RedComm**(i), **BlackComm**(i): Time to send and receive receive data between processor i and its neighbors during the Red/Black phase, for example,

  $$\textbf{RedComm}(i) = \textbf{SendLR}(i) + \textbf{ReceLR}(i) \tag{3}$$

  where

  $$\textbf{SendLR}(i) = \textbf{PtToPt}(i, i+1) + \textbf{PtToPt}(i, i-1) \tag{4}$$

  $$\textbf{ReceLR}(i) = \textbf{Process}(i+1) + \textbf{Process}(i-1) \tag{5}$$

  and

  $$\textbf{PtToPt}(x, y) = NumElt(y) * \frac{Size(Elt)}{BW(x, y)} \tag{6}$$

  $$\textbf{Process}(x) = NumElt(x) * Size(Elt) * Unpack(x) \tag{7}$$

  where

- $NumElt(y)$: Number of elements in a message for processor y, usually available from static code analysis.

- $Size(Elt)$: Size of a single element in bytes, available from static code analysis.

- $BW(x, y)$: Bandwidth in bytes per second between processor x and processor y, available from a priori benchmarks or a dynamic measurement system.

- $Unpack(x)$: Time to unpack a byte for processor x.

Structural models provide an effective framework in which to model the performance of distributed parallel applications [29, 30]. In the following we assume that accurate structural models are available to the user. We discuss the possible repercussions on performance should the structural model itself not provide accurate predictions as part of Section 3.3 and with the experimental results in Section 5.

In the next sections, we discuss two possible representations for model parameters for structural models.

3. Representing Stochastic Values Using Normal Distributions

Normal distributions, also called Gaussian distributions after Carl Friedrich Gauss who proposed them as a model for measurement errors, are representative of a large collection of random variables. As such, many real phenomena in computer systems generate distributions that are close[a] to normal distributions [2, 1, 9].

A normal distribution can be defined by the formula:

$$f(x) = \frac{1}{\sigma\sqrt{2\pi}} e^{-(x-\mu)^2/2\sigma^2}, \qquad -\infty < x < \infty \tag{8}$$

for parameters μ, the **mean**, which gives the "center" of the range of the distribution, and σ, the **standard deviation**, which describes the variability in the distribution and gives a range around the mean. Normal distributions are symmetric and bell-shaped, and have the property that the range defined by the mean plus and minus two standard deviations captures approximately 95% of the values of the distribution.

Figure 1 shows a histogram of runtimes for an SOR benchmark on a single workstation with no other users present, and the normal distribution based on the data mean, m, and standard deviation, *sd*. Distributions can be represented graphically in two common ways: by the **probability density function** (pdf), as shown on the left in Figure 1, which graphs values against their probabilities, similar to a histogram, and by the **cumulative distribution function** (cdf), as shown on the right in Figure 1, which illustrates the probability that a point in the range is less than or equal to a particular value.

In the following subsection, we describe the necessary compositional arithmetic to use normal distributions in structural models; in Sections 3.2 and 3.3 we discuss

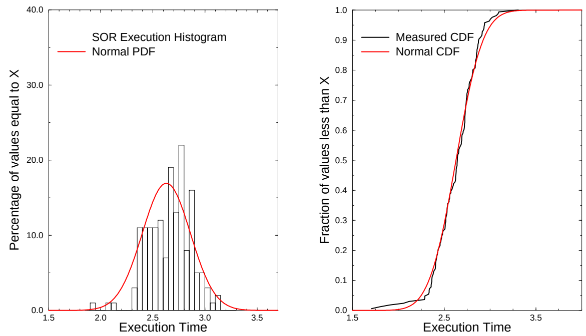

Fig. 1. Graphs showing the pdf and cdf of SOR benchmark with normal distribution based on data mean and standard deviation.

alternatives to consider when the assumption of a normal distribution is too far from the actual distribution of the stochastic value.

3.1. Arithmetic Operations over Normal Distributions

In order to extend structural prediction models for use with stochastic values, we need to provide a way to combine them arithmetically. In this subsection we define common arithmetic interaction operators for stochastic values represented by normal distributions by taking advantage of the fact that normal distributions are closed under linear combinations [18].

For each arithmetic operation, we define a rule for combining stochastic values based on standard statistical error propagation methods [5]. In the following, we assume that point values are represented by P and all stochastic values are of the form (m_i, sd_i) and represent normal distributions, where m_i is the mean and sd_i is the standard deviation.

When combining two stochastic values, two cases must be considered: when the distributions are correlated and when they are uncorrelated. Two distributions are **correlated** when there is an association between them, that is, they jointly vary in a similar manner [10]. More formally, correlation is the degree to which two or more attributes or measurements on the same group of elements show a tendency to vary together. For example, when network traffic is heavy, available bandwidth tends to be low and latency tends to be high. When network traffic is light, available bandwidth tends to be high and latency tends to be low. We say that the distributions of latency and bandwidth are correlated in this case.

When two stochastic values are **uncorrelated** they do not jointly vary in a similar manner. This may be the case when the time between measurements of a single quantity is large, or when the two stochastic values represent distinct characteristics. For example, available CPU on two machines not running any applications in common may be uncorrelated.

Table 1 summarizes the arithmetic operations between a stochastic value and a point value, two stochastic values from correlated distributions, and two stochastic

[a] See Section 3.3 for a discussion on ways to determine "close".

values from uncorrelated distributions.

An example of a structural model in which two stochastic values must be added is

$$\mathbf{Comm} = Latency + \frac{MsgSize}{Bandwidth} \tag{9}$$

where both *Latency* and *Bandwidth* are stochastic values. Subtraction of two stochastic values would have the same form as addition, only with a negative value for one of the m_i. Since normal distributions are closed under addition and subtraction, the resulting stochastic value will also have a normal distribution.

While less common in structural modeling than addition, there are times when it is necessary to multiply two stochastic values. For example, we might calculate the time for a computational portion of a code to run as the product of the available CPU load and the number of data points, divided by the time per data point using a dedicated benchmark. If the application is data dependent or nondeterministic, then it is likely that the dedicated benchmark may be represented by a stochastic value. If both the available CPU and the benchmark are stochastic values, we would need to calculate their product. We treat division of (m_i, sd_i) by (m_j, sd_j) as multiplication of (m_i, a_i) by (m_j^{-1}, sd_j^{-1}) for $m_j \neq 0$. Note that the product of stochastic values with normal distributions does not itself have a normal distribution. Rather, it is long-tailed. In many circumstances, we can approximate the long-tailed distribution with a normal distribution and ignore the tail as discussed below in Section 3.2.1.

	Addition	Multiplication
Point Value and Stochastic Value	$(m_i, sd_i) + P =$ $((m_i + P), sd_i)$	$P(m_i, sd_i) = (Pm_i, Psd_i)$
Stochastic Values with Correlated Distributions	$\sum_{i=1}^{n} (m_i, sd_i) =$ $\left(\sum_{i=1}^{n} m_i, \quad \sum_{i=1}^{n} \lvert sd_i \rvert\right)$	$(m_i, sd_i)(m_j, sd_j) =$ $(\ m_i m_j, (sd_i m_j + sd_j m_i + sd_i sd_j))$
Stochastic Values with Uncorrelated Distributions	$\sum_{i=1}^{n} (m_i, sd_i) \approx$ $\left(\sum_{i=1}^{n} m_i, \sqrt{\sum_{i=1}^{n} sd_i^2}\right)$	$(m_i, sd_i)(m_j, sd_j) \approx$ $\left(m_i m_j, \left(m_i m_j \sqrt{\left(\frac{sd_i}{m_i}\right)^2 + \left(\frac{sd_j}{m_j}\right)^2}\right)\right)$

Table 1. Arithmetic Combinations of a Stochastic Value with a Point Value and with other Stochastic Values [5].

3.1.1. Collective operations over stochastic values

Often structural models will combine components using collective operations such as *Max*, *Min*, or other operators. The combination of stochastic values for operations over a group must often be addressed in a situation-dependent manner.

For example, consider the calculation of the *Max* operator. Depending on the penalty for an incorrect prediction, different approaches may be taken. *Max* could be calculated by selecting the largest mean of the stochastic value inputs, or by selecting the stochastic value with the largest magnitude value in its entire range. For example, to compute the *Max* of three possible stochastic values $\{A = (4, 0.5)$, $B = (3, 2)$, $C = (3, 1)\}$, A has the largest mean, and B has the largest value within its range. However on average, the values of A are likely to be higher than the values of B. The usage of the resulting *Max* value, and the quality of the result required, will determine how *Max* should be calculated. This information would need to be supplied by the model builder, scheduler or user.

3.2. Using Normal Distributions to Represent Non-normal Stochastic Model Parameters

In this section, we provide examples of stochastic parameters that are not normal, but can often be adequately represented by normal distributions.

3.2.1. Long-tailed data

Not all system characteristics can be accurately represented by normal distributions. Figure 2 shows the pdf and cdf for bandwidth data between two workstations over 10 Mbit ethernet. This is a typical graph of a **long-tailed distribution**, that is, the data has a threshold value, and varies monotonically from that point, generally with the median value several points below (or above) the threshold. A similarly shaped distribution, shown in Figure 3, may be found in data resulting from dedicated runs of a distributed Genetic Algorithm code, described in Section 5.4

Neither of these distributions are normal, however, it may be adequate to approximate them using normal distributions. Normal distributions are a good substitution for long-tailed model parameters only when inaccuracy in the predictions generated by the structural model can be tolerated by the scheduler, performance model, or other mechanism that uses the data.

3.2.2. Modal distributions

Some model parameters are best represented by multi-modal distributions. One characterization of the general shape of a distribution is the number of peaks, or *modes*. A distribution is said to be *unimodal* if it has a single peak, *bimodal* if it has two peaks, and *multi-modal* if it has more than two peaks. Figure 4 shows a histogram of available CPU data for an Ultra Sparc workstation running Solaris taken over 12 hours using `vmstat`. The Unix tool `vmstat` reports the exact CPU activity at a given time, in terms of the processes in the run queue, the blocked processes, and the swapped processes as a snapshot of the system every n seconds (where for our trace, $n = 5$). For this distribution, the majority of the data lies in three modes: a mode centered at 0.94, a mode centered at 0.49, and another mode centered at 0.33. For this stochastic parameter, the modes are most likely

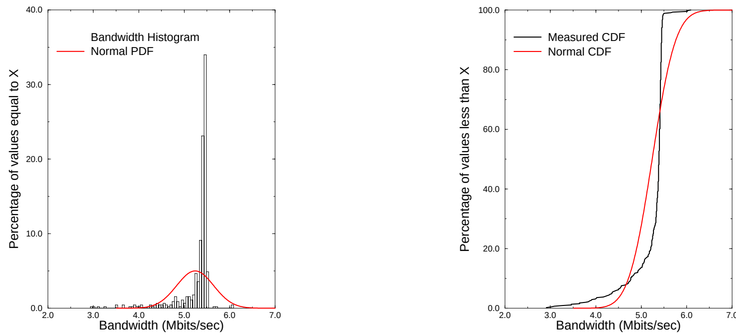

Fig. 2. Graphs showing the pdf and cdf for bandwidth between two workstations over 10Mbits ethernet with long-tailed distribution and corresponding normal distribution.

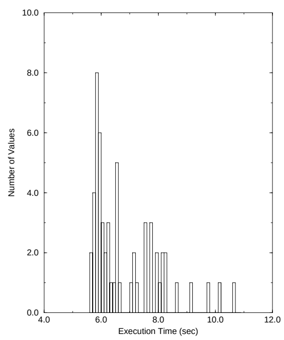

Fig. 3. Graph showing histogram of dedicated behavior of GA code for problem size 800 on a Sparc 10.

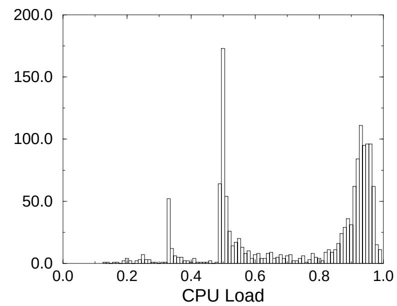

Fig. 4. Available CPU on a production workstation.

an artifact of the scheduling algorithm of the operating system. Most Unix-based operating systems use a round-robin algorithm to schedule CPU bound processes: When a single process is running, it receives all of the CPU. When two processes are running, each uses approximately half of the CPU, when there are three, each gets a third, etc. This is the phenomenon exhibited in Figure 4.

To represent a modal parameter using a normal distribution, it is important to know whether or not the the values represented by the parameter remain within a single mode during the timeframe of interest. If the values of the parameter remain within a single mode (i.e. if they exhibit **temporal locality**), we can approximate the available CPU as a normal distribution based on the data mean and standard deviation of the appropriate mode without excessive loss of information. An example of this (from a 24 CPU hour trace) is shown as a time series in Figure 5.

If the values of the parameter change modes frequently or unpredictably, we say that that the data exhibits **temporal non-locality**. An example of this, taken from the same 24 hour CPU trace as before, is shown as a time series in Figure 6. In this case, some way of deriving a prediction must be devised that takes into account the fluctuation of the parameter data between multiple modes.

A brute-force approach to representing multi-modal data would be to simply ignore the multi-modality of the data and represent the stochastic value as a normal

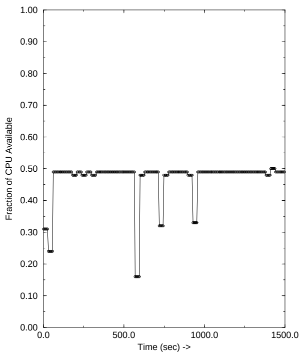

Fig. 5. Time series showing temporal locality of CPU data.

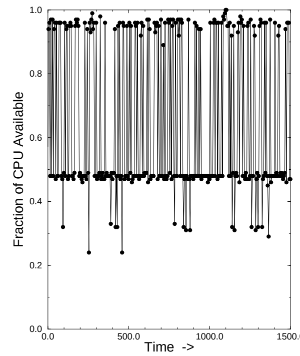

Fig. 6. Time series showing non-temporal locality of CPU data.

distribution based on the mean and standard deviation of the data as a whole. However, it is unlikely that this approximation will capture the relevant behavior characteristics of the data with any accuracy. Because of the multi-modal behavior, a mode with a small variance in actuality may end up being represented by a normal distribution with a large variance (and a large standard deviation). Using this brute-force method for the data in Figure 6, the mean would be 0.66 and the standard deviation would be 0.24.

An alternative approach would be to calculate an *aggregate mean* and *aggregate standard deviation* for the value based on the mean and standard deviation for each mode. Let (m_i, sd_i) represent the mean and the standard deviation for the data in mode i. We define the aggregate mean (**AM**) and the aggregate standard deviation (**ASD**) of a multi-modal distribution by:

$$AM = \sum p_i(m_i) \tag{10}$$

$$ASD = \sum p_i(sd_i) \tag{11}$$

where p_i is the percentage of data in mode i. Since we represent each individual mode in term of a normal distribution, (AM, ASD) will also have a normal distribution. For the data in Figure 6, $AM = 0.68$ and $ASD = 0.031$.

Note that using the aggregate mean and the aggregate standard deviation is an attempt to define a normal distribution that is somehow "close" to the multi-modal distribution. Determining whether two distributions are "close" is itself an interesting problem that we discuss briefly in the subsection below.

3.3. When is a distribution "close" to normal?

In the previous subsections, we made a key assumption that the values in the distribution were "close" to (could be adequately represented by) normal distributions. In order to define "close" we can consider several methods for determining the similarity between a given data set and the normal distribution represented by its data mean and its data standard deviation.

One common measurement of goodness-of-fit is the chi-squared (χ^2) technique [11]. This is a quantitative measure of the extent to which observed counts differ from the expected counts over a given range, called a cell. The value for χ^2 is the sum of a goodness-of-fit for all quantities:

$$\chi^2 = \sum_{all\ cells} \frac{\text{(observed cell count - expected cell count)}^2}{\text{expected cell count}} \tag{12}$$

for the observed data and the expected data resulting from the normal distribution. The value of the χ^2 statistic reflects the magnitude of the discrepancies between observed and expected cell counts, a larger value indicates larger discrepancies.

Another metric of closeness in the literature is called 1-distance between pdf's [22] where:

$$\|f_n - f_d\|_1 = \int_{-\infty}^{\infty} |f_n(x) - f_p(x)| dx \tag{13}$$

for function f_d for the data and f_n for the normal distribution based on the data mean and standard deviation. This corresponds to a maximal error between the functions.

For both of these metrics, a user or scheduler would need to determine a threshold for closeness acceptable for their purposes.

If we approximate non-normal data using a normal distribution there may be several effects. When the distribution of a stochastic value is represented by a normal distribution but is not actually normal, arithmetic operations might exclude values that they should not. By performing arithmetic on the mean and standard deviation we are able to use optimistic formulas for uncorrelated values in order to narrow the range that is considered in the final prediction. If the distributions of stochastic values were actually long-tailed, for example, this might cut off values from the tail in an unacceptable way.

Normal distributions are closed under linear combinations [18], but general distributions are not. If we use arithmetic rules defined for normal distributions on non-normal data, we have no information about the distribution of the result. Finally, it may not be possible to ascertain the distribution of a stochastic value, or the distribution may not be sufficiently "close" to normal. In such cases, other representations must be used. In the next section, we explore alternative an for representing stochastic values, namely using intervals.

4. Representing Stochastic Values using Intervals

This section offers an alternative to representing stochastic values as normal distributions by representing them as intervals. The definition of an interval is given, and the rules for interval arithmetic are explained, followed by a discussion of the necessary assumptions as well as the advantages and disadvantages of this approach.

4.1. Defining Intervals

The simplest way to represent the range of data represented by a stochastic value is as an interval of values. The **interval** of a stochastic value X is the tuple

$$X = [\underline{x}, \overline{x}] \qquad where \qquad \{x \in X | \underline{x} \leq x \leq \overline{x}\} \tag{14}$$

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 **length** 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 always possible, and correlates to the intuition behind defining lower and upper bounds for a value.

A disadvantage of representing stochastic values by intervals is that interval arithmetic assumes that the distribution of values over an interval is uniform. This disadvantage can sometimes be mitigated for non-uniformly distributed intervals by using a technique called interval splitting [23, 34]. Interval splitting is a a procedure that divides the interval into sub-intervals, each of which more closely approximates a uniform distribution, and results in a histogram. Other methods to obtain good results include recursive differentiation [27], the use of Taylor expansions [13], and other optimization techniques. However, these methods may greatly increase the computational complexity of calculating an interval.

4.2. Arithmetic over Intervals

Combining stochastic values represented as intervals involves obeying well defined rules for computing interval arithmetic [27]. Formulas are given in Tables 2, 3, and 4. The result of arithmetic on intervals in structural models is an interval that provides an upper bound and lower bound on the prediction.

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

Table 2. Addition and Subtraction over interval values, $x = [\underline{x}, \bar{x}]$, $y=[\underline{y}, \bar{y}]$.

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

Table 3. Multiplication of interval values $x = [\underline{x},\bar{x}]$ and $y = [\underline{y},\bar{y}]$.

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

Table 4. Division of interval values $x = [\underline{x},\bar{x}]$ and $y = [\underline{y},\bar{y}]$. Note that interval division of $\frac{x}{y}$ is only defined if 0 is not in y.

As with normal distributions, the correlation of multiple stochastic values represented by intervals must be determined. The interval arithmetic described in these tables does not assume that multiple values are uncorrelated. This may lead to the overestimation of apparently simple expressions, since each occurrence of a single variable in an arithmetic expression is treated as a distinct variable. For example, if $A=[\underline{a}, \bar{a}]$, then $A-A = [\underline{a} - \bar{a}, \bar{a} - \underline{a}]$, instead of [0,0]. This is also called the *simultaneity* problem [27].

In some cases, the simultaneity problem can be mitigated by rewriting the expression to avoid multiple occurrences of arguments [20]. 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. To further reduce the effect of the simultaneity problem, Hansen [12] has developed a generalized interval arithmetic, but this requires additional assumptions about the characteristics of the intervals not available in more practical settings.

It is possible to avoid the problems associated with standard interval arithmetic if it can be assumed that the probabilities associated with the values in an interval are monotonic (increasing or decreasing throughout the entire range). When this assumption holds, a sharp interval evaluation can be computed simply by evaluating the function at the endpoints of the argument interval [20]. That is, instead of using the interval arithmetic described above to solve the structural model parameterized by stochastic values $X_i = [\underline{x_i}, \overline{x_i}]$ and resulting in the prediction $P = [\underline{p}, \overline{p}]$, we solve the model twice: first using the $\underline{x_i}$ values to evaluate $\underline{p}$, then solving the model using the $\overline{x_i}$ values to evaluate $\overline{p}$.

5. Experimental Verification

The experiments in this section address two questions:

1. Can stochastic value predictions accurately capture program behavior in multi-user clustered environments?
2. How do stochastic values represented by normal distributions and by intervals compare in terms of accuracy of generated performance prediction?

5.1. Experimental Model

Our experiments were conducted on a cluster of workstations under a variety of ambient workload conditions, rather than introducing synthetic workloads. As such, the ambient load on each machine varied during the course of a set of experiments, and even during a single run. In order to compare experimental sets, we defined a machine with a **low** variation in percentage of CPU availability as having a variation of 0.25 or less, on a scale from 0 to 1, a **medium** variation when the percentage of CPU availability varied more than 0.25, but less than 0.5, and a **high** variation when the percentage of CPU availability varied over half the range. The stochastic CPU information used in the models was supplied by the Network Weather Service (*NWS*) [37, 38, 39], a facility that can provide both monitored data and predictions of dynamic values for bandwidth and available CPU on a given system. The NWS supplied us with accurate run-time information about the percentage of CPU availability on our machines at 10 second intervals.

In the course of our experiments, we examined three applications (a distributed Successive Over-relaxation [SOR] code, a distributed Genetic Algorithm [GA] code, and a distributed LU Decomposition [LU] code) on a distributed network of Sparc workstations located in the UCSD Parallel Computation Lab (*PCL*). The machines were connected with a mix of 10 Mbit (slow) and 100 Mbit (fast) ethernet over an Intel switch, and all ran Solaris. Both the CPU and the network were shared with other users. Additional experiments are given in [30].

5.2. Performance Metrics

To address the question of the accuracy of stochastic predictions in capturing program behavior, we use a performance metric that evaluates how often actual application execution times fall within the stochastic prediction range. We call this metric **capture**. The capture metric measures the percentage of execution times falling within the predicted range for a set of application runs, and can be used to measure the goodness of predictions made with either an interval representation or a normal distribution representation. In the case of a normal distribution representation, we define the upper bound and lower bound of the range to be the mean plus or minus two standard deviations.

For interval representations, we would also like to know if the range of the interval prediction is "too big" with respect to the actual values of the execution. To determine this, we used a metric called the **sharpness** of an interval. The idea is that the interval range should not only predict a range of behaviors that includes the actual execution behavior range, but this range should be as small as possible in order to provide more useful information about the execution behavior of the application.

We say that 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 [27]. Sharp, or **tight**, bounds on the prediction intervals are especially important when the predictions are used for scheduling, where compact bounds on a predicted execution time can lead to a better prediction of execution time. Note that a tight interval may be large due to the inclusion of outliers, as shown in Figure 7, and does not provide any information with respect to the distribution of the data within the interval. A tight interval only shows that the interval need not be larger to capture all the relevant data.

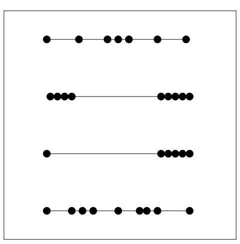

Fig. 7. Four sample stochastic values, each with the same interval representation.

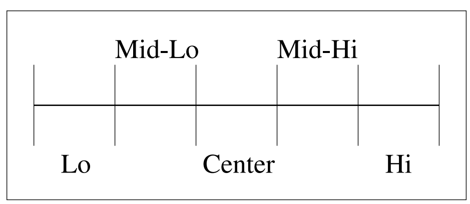

Fig. 8. Interval buckets.

We can try to ascertain something about not only sharpness but the uniformity and bias of the values within an interval by dividing the predicted range into five equal "buckets", as shown in Figure 8. The proportion of data points within each bucket provides can provide useful information about the interval. For example, if the majority of the values for a set of runs fall in the center bucket, the values cannot be uniform throughout the interval. If no values are in the outer buckets, then the interval could have been sharper and still captured the values. The uniformity and bias execution time data over the 5 buckets for the entire set of experiments is given in Table 5.

Figure	Under	Lo	Mid Lo	Center	Mid Hi	Hi	Over
SOR1 Figure 10	0	0	8	5	5	6	1
SOR2 Figure 13	0	2	6	8	4	3	3
GA 1 Figure 15	0	0	2	8	5	3	6
GA2 Figure 17	0	3	1	9	1	3	8
LU 1 Figure 19	0	0	8	9	3	1	0
LU2 Figure 21	0	2	10	9	0	1	0

Table 5. Uniformity and bias 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.

5.3. SOR Experiments

Successive Over-Relaxation (SOR) is a Regular SPMD code that solves Laplace's equation, as described in Section 2. We illustrate the SOR experiments with two sets of experiments (SOR1 and SOR2) conducted under different workloads, representative of a larger set given in [30]. Each set of experiments consisted of about 25 runs. In the first set (SOR1), the PCL cluster had two processors with highly variable CPU availability and two with low variable CPU availability machines. In the second set (SOR2), the PCL cluster had one high variable availability, two medium variable availability, and one low variable availability.

Figure 9 shows the normal distribution stochastic value predictions versus the actual time, and Figure 10 shows the interval stochastic value predictions and the actual time for the SOR1 runs. For illustration, we show the recorded CPU values in Figure 11 for this experiment, in the others we will only give summary statistics.

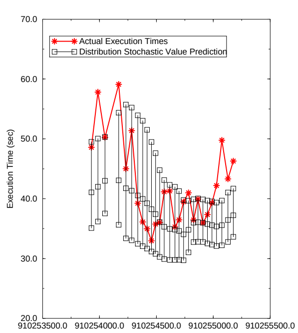

Fig. 9. SOR1 - Normal distributions stochastic value prediction for SOR benchmark.

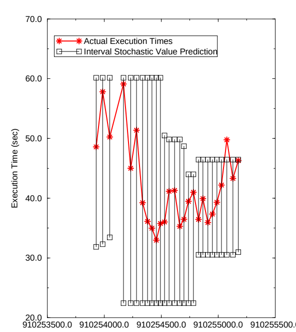

Fig. 10. SOR1 - Interval stochastic value prediction for the SOR benchmark.

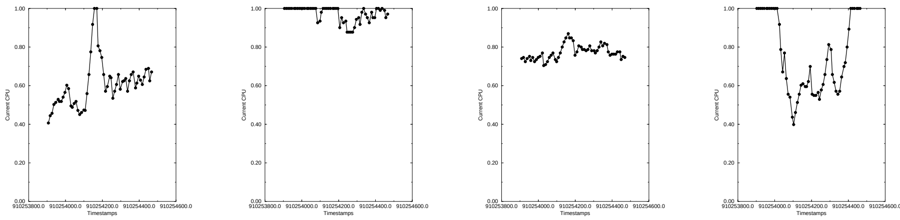

Fig. 11. CPU values during runtime for prediction experiments depicted in Figure 9 and Figure 10.

For the SOR1 runs, both stochastic value representations do well: Using normal distribution representations, we capture 20 of 27 runs; using interval representations, we capture 26 of 27. From the information given in Table 5, we see mild asymmetry in the distribution of the interval for SOR1, with a slight bias to the higher values. Note that decreasing the interval for these runs would likely lower the capture percentage significantly.

Figures 12 and 13 show the normal distribution and interval predictions versus actual time for the SOR2 runs. For these runs, using a normal distribution representation captured 13 of 27 runs, while the interval representation captured 24 of the 27. From the distribution table, we see that the actual values are spread across the entire range indicating that tighter bounds would adversely affect the capture percentage. In this case, the interval does a good job of predicting the range of execution times. For both runs, we also graph the mean of the normal distribution stochastic predictions, a reasonable point value representation, and there is considerable discrepancy between the mean and the actual execution time.

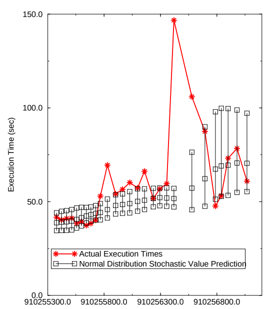

Fig. 12. SOR2- Normal distribution stochastic value prediction for the SOR benchmark.

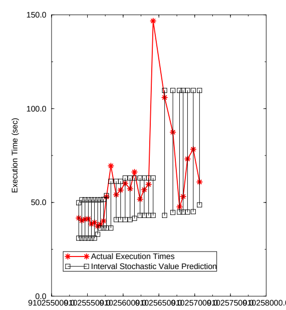

Fig. 13. SOR2- Interval stochastic value prediction for the SOR benchmark.

5.4. GA Experiments

We implemented a genetic algorithm (GA) heuristic for the Traveling Salesman Problem (TSP) [19, 36]. Our distributed implementation of this genetic algorithm uses a global population and synchronizes between generations [7]. 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. We use stochastic values for the CPU information. We illustrate the GA experiments with 2 sets of experiments that we describe below, representative of a larger set [30].

During the GA1 experiments, two machines on the cluster had a highly variable availability, and two had low variability. Figure 14 shows the normal distribution stochastic value predictions versus the actual time, which captured 18 of 24 runs, and Figure 15 shows the interval stochastic value predictions, which captured 19 runs. There is a strong bias in the interval representation towards the values in the upper buckets, and the interval itself could be tighter and still reflect the same capture rate.

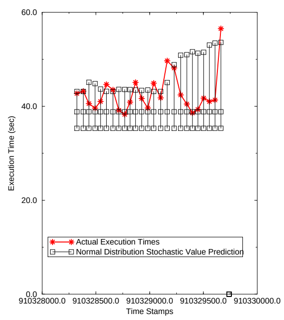

Fig. 14. GA1 - Normal distribution stochastic value prediction for the GA code.

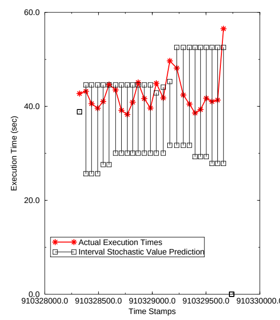

Fig. 15. GA1- Interval stochastic value prediction for the GA code.

During the GA2 experiments, the PCL cluster had two highly variable machines and two low variability machines. Figure 16 shows the normal distribution stochastic representation, which captured 21 of 25 runs, and Figure 17 interval stochastic representation, which captured 23. In this set of experiments, it appears that the interval has a bias towards the lower end of the range and is probably too tight. We note that in the GA performance model even the dedicated GA benchmark (which provides information about execution time for one iteration on a dedicated processor as shown previously in Figure 3) has a wide range of values due to non-determinism. A stochastic value parameterization of the behavior of this component may improve the overall accuracy of the performance model predictions.

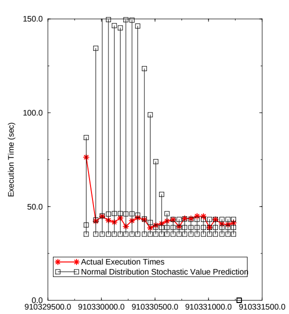

Fig. 16. GA2- Normal distribution stochastic value prediction for the GA code.

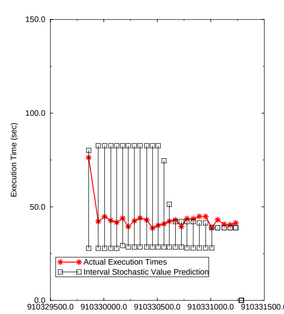

Fig. 17. GA2- Interval stochastic value prediction for the GA code.

5.5. LU Experiments

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) [3]. This system of equations is the result of an unfactored implicit finite-difference discretization of the Navier-Stokes equations in three dimensions. It has the feature that it tests the communication aspects of the system well by sending a relatively large number of 5 word messages. We use stochastic values for the CPU information.

During the LU1 experiments, two of the machines showed a high variability in availability, and two had a low variability. Figure 18 shows the normal distribution stochastic value prediction, which captured 17 of 21 runs. Figure 19 shows the interval stochastic value predictions and captured 20 runs. These predictions are biased towards the higher ranges, and could be tighter.

During the LU2 experiments, the PCL cluster had one high variability machine, one medium variability machine and two low variability machines. Figures 20 and 21 show actual execution times versus predictions with the normal distribution representation for the stochastic values and predictions with the interval representation for the stochastic values respectively. The normal distribution representation captured 17 of 22 runs. The interval representation captured all 22 and was biased towards the low end in these experiments.

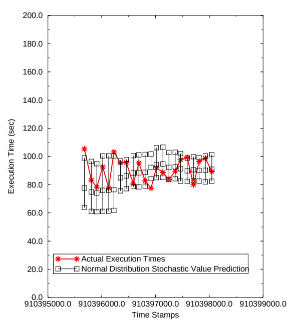

Fig. 18. LU1- Normal distribution stochastic value prediction for the LU benchmark.

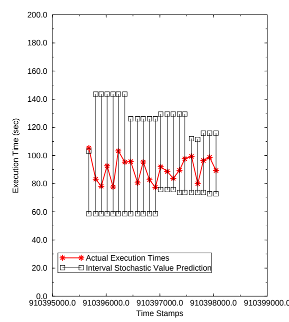

Fig. 19. LU1- Interval stochastic value prediction for the LU benchmark.

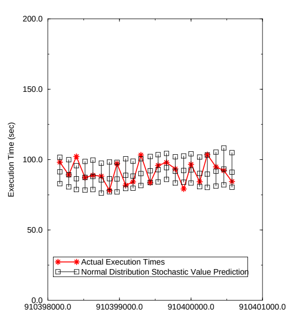

Fig. 20. LU2- Normal distribution stochastic value prediction for the LU benchmark.

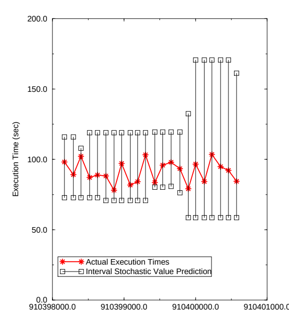

Fig. 21. LU2- Interval stochastic value prediction for the LU benchmark.

5.6. Discussion

Recall that we set out to address two questions:

1. Can stochastic values represented by normal distributions or intervals accurately capture program behavior in multi-user clustered environments?

2. How do stochastic values represented by normal distributions and stochastic values represented by intervals compare in terms of accuracy of generated performance prediction?

To address the first question, it is clear that both representations do a good job of capturing much of the behavior for the demonstration applications. As described before, if we use the mean of the normal distribution stochastic parameter representation as a representative point value for the range, the error between actual execution times and the mean of the predictions is between 2% and 60% for SOR1 and SOR2, 1% and 77% for GA1 and GA2, 3% and 45% for LU1 and LU2. In contrasting this error with proportion of data values captured, it is clear that the stochastic parameters provide important additional information about execution behavior.

To address the second question, the data is inconclusive. Sometimes the normal distribution representation does a better job of capturing the data than the interval representation, and sometimes not. We feel this has to do with the variability of the application behavior. For LU and SOR, the distribution predictions are better, possibly indicating that capturing 95% of the data is accurate for applications with constant dedicated benchmarks. For the GA, the interval predictions were better, possibly indicating that with variable application behavior we need to take into account 100% of the data. However, these results are inconclusive and one are in which future work will be conducted.

6. Related Work

There are several related approaches to note, however stochastic values as defined here are not related to Petri net models [24, 26], 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. [8] use probability distribution functions to calculate wait times on a token ring network. This work borrows heavily from queuing theory for a more theoretical setting than our production environment. Formulas are provided to determine the suitability of an application for a given network based on queuing models, but no practical experiments are shown. In addition, Das' group at Penn State has been studying the effect of assuming normal distributions in simulations [9]. In addition, the WARP project [32, 33] models resource contention using queuing models.

The most closely related work to our approach is by Mendes as part of the Rice Fortran D95 compiler [25]. They generate symbolic performance predictions using a data parallel compiler. fragment upper and by considering (eg. memory the Fortran to extract information in symbolic form, of all the activity in the translated number of processors (P) and predict the program's performance, to evaluate this cost model for specific and N. This work, as we do, assumes some benchmarks, for example the time to send a X, the time to receive a message of size X, an assignment and the time to do an However, some terms in the be able to be determined at may change as the problem size colleagues allow the use of an estimated lower and upper bound for performance model values and build two models - one for the lower bound and one for the upper. The range between the two can be very large, and no attempt is made to minimize it. This work also stresses the need for models to be tied to the compiled code, not the source code, for analysis, and underscore the important insight that implementations must be analyzed, not machine and application separately.

Another approach to using a range of values in models is to build upon fuzzy logic [16], where a fuzzy number is represented by a set of real numbers and an associated membership function. Autopilot [28] couples a fuzzy logic rule base for distributed decision making with wide area performance sensors and policy control actuators. The rule base embodies common sense rules for resource management that are more amenable to fuzzy logic rules due to the conflicting goals involves and poorly understood optimization spaces involved. Lüthi also examined approaches for using fuzzy logic for network queuing models [21] by replacing single value parameters of open as well as closed queuing models by fuzzy numbers, and discusses asymptotic results for these models.

perturbations to stability in being ported execution was represented by a even corresponded to a graph node, likely unavailable in distributed parallel an event on one platform will not correspond to on another platform due to compiler, software and differences. Using this graph and detailed platform the

Karlin's work in competitive analysis using partial information also bears relevance to our work [15, 14]. Competitive analysis is concerned with comparing the performance of on-line algorithms with that of optimal off-line algorithms. Often work in this area is faced with the concern of only partial information being available, and decisions being made appropriately in that environment.

as timeliness and accuracy is used to parameterize of Finally, research in the area of in the area of Grid computing [6, 17] has begun to address the need for additional information in order to make educated decisions about performance in a distributed environments 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.

7. Summary

In this paper, we described a new approach to improving the predictions of application performance models in contended clustered environments. We defined stochastic values to reflect a likely range of behavior for model parameters, and showed that stochastic values provide important additional information on the execution behavior of our demonstration applications in a contended clustered environment.

We presented two representations for stochastic values: normal distributions and intervals. Each representation has advantages and disadvantages. Intervals are the most easily defined since the minimum and maximum value in a data set for a stochastic value are easy to determine. However, outliers in the data can affect the size of the interval, and no details about the shape of the data are included in a simple range. Normal distributions can be combined using well understood operations on the data, and provides important additional information about the distribution of the data within the range. We did not address other distributions or more general distributions in this paper. However, note that in order to use such distributions to parameterize performance models, we must be able to combine them arithmetically, and it must be computationally efficient to do so.

Future work includes examining more applications and settings to try to determine which representation should be used in which situation. Extensions also include extending these methods for use when only partial information is available.

Acknowledgements

This work was supported in part by NASA GSRP grant #NGT-1-52133, Darpa grant #N66001-97-C-8531, and NSF grant #ASC-9701333.

This work would not have been possible without the generous help and support of the AppLeS Group, and especially Rich Wolski. The SOR application is compliments of Silvia Figueira, and the GA compliments of Karan Bhatia. Thanks also to Neil Spring, Graziano Obertelli and Jim Hayes for extensive help with the Network Weather Service.

References

the order cited in the text. Use the style following examples. For journal names, use the standard

1. Adve, V., and Vernon, M. The influence of random delays on parallel execution times. In *Proceedings of Sigmetrics '93* (1993).
2. Adve, V. S. *Analyzing the Behavior and Performance of Parallel Programs*. PhD thesis, University of Wisconsin-Madison, December 1993. Also available as University of Wisconsin Computer Sciences Technical Report #1201.
3. Bailey, D., Harris, T., Saphir, W., van der Wijngaart, R., Woo, A., and Yarrow, M. The nas parallel benchmarks 2.0. Tech. Rep. 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`.
4. Bailey, D. H., Barszcz, E., Barton, J. T., Browning, D. S., Carter, R. L., Dagum, L., Fatoohi, R. A., Frederickson, P. O., Lasinski, T., Schreiber, R. S., Simon, H. D., Venkatakrishnam, V., and Weeratunga, S. K. The nas parallel benchmarks. *International Journal of Supercomputer Applications 5*, 3 (Fall 1991), 63–73.
5. Barry, B. A. *Errors in Practical Measurement in Science, Engineering and Technology*. John Wilsey & Sons, 1978.
6. Berman, F., Wolski, R., and Schopf, J. Performance prediction engineering for metacomputing. `http://www.cs.ucsd.edu/groups/hpcl/apples/PPE/index.html`, 1997.
7. Bhatia, K. Personal communication, 1996.
8. Brasileiro, M. A. G., Field, J. A., and Moura, A. B. Modeling and analysis of time critical applications on local area networks. In *Proc. of Chilean Computer Science Society* (1991), pp. 459–71.
9. Das, C. Personal communication with rich wolski, 1997.
10. Devore, J., and Peck, R. *Statistics: The Exploration and Analysis of Data*. Duxbury Press, 1996, p. 88.
11. Devore, J., and Peck, R. *Statistics: The Exploration and Analysis of Data*. Duxbury Press, 1996, p. 567.
12. Hansen, E. R. A generalized interval arithmetic. *Interval Mathematicals, proceedings of the International Symposium* (1975).
13. Hansen, E. R. Global optimization using interval analysis. *Monographs and textbooks in pure and applied mathematics* (1992).
14. Irani, S., Karlin, A. R., and Phillips, S. Strongly competitive algorithms for paging with locality of reference. *SIAM Journal on Computing 25*, 3 (June 1996), 477–97.
15. Karlin, A. R., Manasse, M. S., McGeoch, L. A., and Owicki, S. Competitive randomized algorithms for nonuniform problems. *Algorithmica 11*, 6 (June 1994), 542–71.
16. Kasabov, N. K. *Foundations of neural networks, fuzzy systems and knowledge engineering*. The MIT press, 1996.
17. Kesselman, C., and Bannister, J. Qualis: Guaranteed quality of service for metacomputing, 1997. Private communication.
18. Larsen, R. J., and Marx, M. L. *An Introduction to Mathematical Statistics and*

Its Applications. Prentice-Hall, 1986.

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

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

21. Lüthi, J., and Haring, G. Fuzzy queuing network models of computer systems. In *Proceedings of the 13th UK Performance Engineering Workshop* (1997).

22. Majumdar, S., Lüthi, J., and Haring, G. Histogram-based performance analysis for computer systems with variabilities or uncertainties in workload. Tech. Rep. Research Report SCE-95-22, Department of systems and computer engineering, Carleton University, Ottawa, Canada, November 1995.

23. Majumdar, S., and Ramadoss, R. Interval-based performance analysis of computing systems. In *Proceedings of the Third International Workshop on Model Analysis and Simulation of Computer and Telecommunication Systems* (1995).

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

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

26. Molloy, M. Performance analysis using stochastic petri nets. *IEEE Transactions on Parallel and Distributed Systems C-31* (September 1990), 913–7.

27. Neumaier, A. *Interval methods for systems of equations*. Cambridge University Press, 1990, ch. Basic Properties of Interval Arithmetic.

28. Ribler, R. L., Simitci, H., and Reed, D. A. The autopilot performance-directed adaptive control system. *Future Generation Computer Systems, special issue (Performance Data Mining)* (1997).

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

30. Schopf, J. M. *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, `http://www.cs.nwu.edu/~jms/Thesis/thesis.html`.

31. Schopf, J. M., and Berman, F. Stochastic scheduling. In *Proceedings of SC'99* (1999).

32. Simon, J., and Wierum, J.-M. Accurate performance prediction for massively parallel systems and its applications. In *Proceedings of Euro-Par '96 (Second International European Conference on Parallel Processing)* (1996), vol. 2, pp. 675–88.

33. Simon, J., and Wierum, J.-M. Performance prediction of benchmark programs for massively parallel architectures. In *Proceedings of the 10th Annual Intl. Conf. on High-Performance Computers, (HPCS 96)* (1996).

34. Skelboe, S. Computation of rational integer functions. *BIT 14* (1974).

35. Taylor, H. M., and Karlin, S. *An Introduction to Stochastic Modeling*. Academic Press, 1993.

36. Whitley, D., Starkweather, T., and Fuquay, D. Scheduling problems and traveling salesman: The genetic edge recombination operator. In *Proceedings of*

International Conference on Genetic Algorithms (1989).

37. Wolski, R. Dynamically forecasting network performance using the network weather service(to appear in the journal of cluster computing). Tech. Rep. TR-CS96-494, UCSD, CSE Dept., 1996.

38. Wolski, R. 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).

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