Performance Prediction Using Intervals

Extended Abstract

Jennifer M. Schopf* and Francine Berman†
Computer Science and Engineering Department
University of California, San Diego
{jenny,berman}@cs.ucsd.edu
UCSD CSE Dept Technical Report #CS97-541, May 1997.

Abstract

Accurate performance predictions are difficult to achieve for applications on production distributed parallel systems. We address this problem by allowing characteristic application and system data to be represented as an **interval** *of possible values. We use intervals as parameters to a structural modeling technique, and predict the range of application behavior that will occur. We test this approach on a production system using a distributed SOR code, and demonstrate the resulting additional information and accuracy resulting from the interval values.*

1 Introduction

Clusters of distributed machines have become a common platform for high performance parallel applications but remain a challenging environment in which to achieve good performance. One reason for this is the difficulty in making predictions of application performance in a dynamic distributed computing environment. We address this problem by allowing characteristic application and system data to be represented in performance models as an *interval* of possible values, resulting in more useful predictions of application execution performance.

Most performance prediction approaches use model parameters to describe system and application characteristics such as bandwidth, CPU load information, message sizes, 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 8 Mbits/second.

However, in practice these point values are more often a best guess, an estimate under ideal circumstances, or an accurate value only for a given time frame. Often it is more useful to represent system and application characteristics as a range of possible values, for example, bandwidth may be reported as 8Mbits/second ± 2Mbits/second. We call values represented in this way *interval values*.

We have developed a methodology that takes advantage of representing model parameters as interval values to model application and system characteristics. By knowing the range of values a system characteristic may assume, we can determine the range of application behavior that will occur. We accomplish this by extending the domain of structural performance models to allow for interval values, resulting in predictions that are interval values as well.

Our experiments show that interval values can be used to enhance the information conveyed by a performance prediction. Whereas point value predictions provide a likely estimate of application execution behavior, interval values convey both a prediction and a range of likely behavior. In particular, interval value predictions provide valuable additional information that can be supplied to a scheduler or to the programmer to improve the overall execution time of distributed parallel applications.

1.1 Motivation

To illustrate the role of interval values in parameterizing a performance model, consider a simple two-machine system (consisting of machines A and B) executing an embarrassingly parallel application with a fixed number of units of work to be completed. In order to predict the amount of work each machine should be assigned, we need an estimate of the capacity (or time per unit of work) of each machine, as shown in Table 1.

In a dedicated environment, the time for machine A to perform one unit of work is 10 seconds and for machine B is 5 seconds. These are both represented as point values since often there is negligible variance in runtimes in a dedicated environment. To balance the workload for

*Supported in part by NASA GSRP Grant #NGT-1-52133.
†Supported in part by NSF Grant #ASC 93-01788

the application in this environment, machine B should receive twice as much work as machine A.

However, in a production environment, contention for the processor and memory will cause the capacity of the machines to vary in a machine-dependent fashion. If we determine capacity using a mean capacity measure over a 24-hour period, it is possible that A and B will have the same capacity on average, say 12 seconds per unit of work. In this case, it would make sense to evenly balance the work between the machines.

Mean values are only summary statistics for a range of possible values, and may neglect crucial information about the distribution of work over time. It is likely that because machine B is much faster than machine A it has more users, and therefore a more dynamic load. Because of this, at any given time the capacity for machine B is 12 seconds per unit of work ± 30%. That is, the time to complete a unit of work will vary over an interval from 8.4 seconds to 15.6 seconds. On the other hand, since machine A is a slower machine without as many users contending for it's use, the actual performance capacity might be 12 seconds per unit of work ± 5% (or 11.4 to 12.6 seconds per unit of work). On average both machines perform the same, however at any given time their performance may vary radically.

	Machine A	Machine B
Dedicated	10 sec	5 sec
Production (point)	12 sec	12 sec
Production (interval)	12 sec ± 5%	12 sec ± 30%

Table 1. Machine Capacities for 2-machine example.

With this additional information about the interval of problem behavior, we can develop a more sophisticated execution strategy, depending on the user's performance metric. If the accuracy of the prediction is a priority, then more work could be assigned to the low variance machine (machine A). If the penalty for poor predictions is low, we might optimistically assign a greater portion of the work to the often faster machine B. How and when this information is used is the job of a smart scheduler, but there are many uses for the additional information, for example with Quality of Service (QoS) guarantees.

This paper is organized as follows. In Section 2 we review the structural modeling approach and illustrate it with an SOR application example. In Section 3 we define interval values and operations over them for use in structural modeling. In Section 4, we demonstrate the determination of a range of prediction values using a structural model of SOR with interval values. We conclude with a summary and a discussion of future work in Section 5.

2 Structural Modeling

Structural models represent the top-most functionality of the application's constituent components and their interactions. The constituent component models in turn use both application and system characteristics (CPU load, message sizes, operation counts, bandwidth figures, etc) as point value model parameters. Details of this approach can be found in [Sch97], a part of the Application-Level Scheduler (AppLeS) project [BW96, BWF$^+$96].

We will extend the structural modeling approach to define predictions for applications when model parameters are represented as interval values. Section 3 defines interval values and describes the needed operations over them for use in structural modeling.

2.1 SOR Example

To illustrate the structural modeling approach, we will step through the development of a model for a distributed red-black Successive Over-Relaxation (SOR) code. This program is a typical distributed stencil application on an N by N grid. The application is divided into "red" and "black" phases, with communication and computation alternating for each. This repeats for a predefined number of iterations. A data decomposition for this code is depicted in Figure 1.

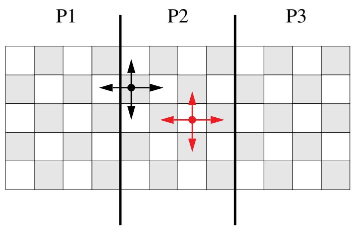

Figure 1. Strip decomposition for red-black SOR

The structural model for the SOR application should echo its functional structure. Therefore it has four constituent components: the Red Computation, the Red Communication, the Black Computation and the Black Communication. One possible top-level structural model for the overall execution time of the SOR application is

$$\mathbf{ExTime} = Max[\mathbf{RedComp_i} + \mathbf{RedComm_i} + \mathbf{BlackComp_i} + \mathbf{BlackComm_i}]$$

In this application, the time for a single iteration is equivalent to the time for the slowest processor to complete that iteration, assuming that synchronization occurs once per iteration. In practice, however, there is loose synchronization between neighbors at each communication, possibly resulting in "drift". In our implementation, the SOR code was statically load-balanced and executed on a quiescent system (no other users present), hence the drift was minimal.

2.2 Component Models

Each of RedComp$_i$, RedComm$_i$, BlackComp$_i$, and BlackComm$_i$ are **component models**. Component models are defined, possibly recursively, as combinations of model parameters (benchmarks, latency and bandwidth measurements, CPU performance, operations counts, etc.) and/or other component models. A given component model may have several possible instantiations.

The communication components of the SOR can be modeled using simple bandwidth models for each point-to-point send, i.e.

$$\mathbf{Comm_i = SendLR_i + ReceLR_i}$$

with

$$\mathbf{SendLR_i = PtToPt(i, i+1) + PtToPt(i, i-1)}$$

$$\mathbf{ReceLR_i = PtToPt(i+1, i) + PtToPt(i-1, i)}$$

and

$$\mathbf{PtToPt(x, y)} = NumElt_i * \frac{Size(Elt)}{BW(x,y)}$$

These component models employ the model parameters:

- $NumElt_i$: Number of elements in a message,
- $Size(Elt)$: Size of a single element in bytes, and
- $BW(x,y)$: Bandwidth, in bytes per second, between processors x and y.

We extend the structural model so that model parameters can be represented either as point values, for example in the case of $Size(Elt)$, or as interval values, which might be a more accurate representation for $BW(x,y)$. In the next section we determine the interactions needed for this extension.

Component models must also be defined for the computational components. For example, one possible model using an operation count would be

$$\mathbf{Comp_i} = NumElts_i * Op(i, Elt) * CPU_i * CPULoad_i$$

where the model parameters are:

- $NumElt_i$: Number of elements computed by processor i,
- $Op(i, Elt)$: Number of operations to compute a single element on processor i,
- CPU_i : Time to perform one operation on dedicated processor i, and
- $CPULoad_i$: The percentage of CPU i that this application will be able to use.

In our experiments (presented in Section 4), we use interval values for $CPULoad_i$, and point values for all other model parameters.

3 Using Interval Values for Predicting Performance

In [Sch97], structural models combined point value model parameters using arithmetical operators to produce a point value prediction of the application behavior. In the last section we indicated that this approach can be extended in a natural way to include interval valued model parameters, but we did not define how these intervals would be combined, nor did we focus on the distribution of values over the interval. We focus on the second issue first.

3.1 Defining Interval Values

We represent the model parameters needing interval values as a value and a range. In this paper we assume the distribution over the range is normal. For many application and system characteristics, this is a valid representation of the range of the possible data; examples of this are shown in the following figures. However, we have found that assuming a range is normal even where the behavior of the data is long-tailed or multi-modal is satisfactory in many cases. Our experiments in Section 4 present one case for this, others are detailed below.

One reason we make the assumption of a normal distribution is that normal distributions are closed under linear combinations, allowing us to manipulate the interval values in our models [LM86]. Any distribution with this property can be used.

In this section we represent characteristic behavior with two standard statistical functions. The Probability Distribution Function (PDF) shows the probability of some x being equal to a specific value. This corresponds to a histogram of actual data showing the percentage of values falling within a small range. A Cumulative Distribution Function (CDF) shows the probability of some x being less than or equal to a value.

One situation where the behavior of a characteristic is clearly normal is that of a benchmark on a dedicated system. Figure 2 shows a histogram of runtimes for a sample sorting code on a single workstation with no other users present, and the corresponding normal distribution PDF. Figure 3 shows the CDF for this data.

Although the workstation is dedicated, there is still some variation in the time for the application to run, possibly due to memory affects or system behavior. If these effects are significant, we may need to represent the data using an interval value. In this case, a normal distribution adequately represents the possible values for the runtimes.

A disadvantage to representing the data as a normal distribution is that in some cases, the distribution of actual values may be long-tailed. An example of this can be seen in Figure 4, a histogram of ethernet bandwidth

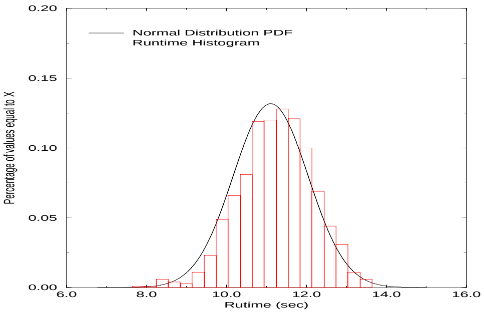

Figure 2. PDF of sample runtime with normal distribution.

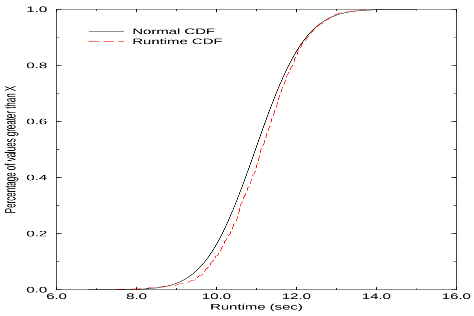

Figure 3. CDF of sample runtime with normal distribution

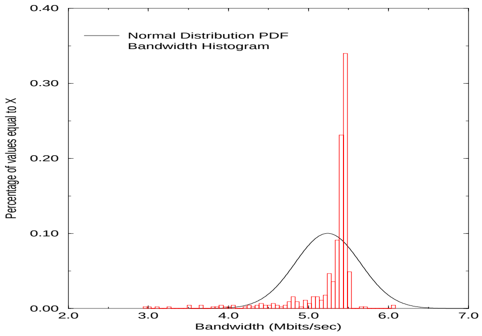

Figure 4. Long-tailed distribution of bandwidth values, with corresponding normal PDF.

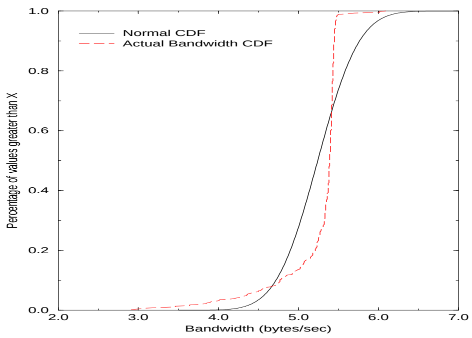

Figure 5. CDF for long-tailed distribution of bandwidth values.

between two workstations with the normal distribution given the data mean for the histogram data. Figure 5 shows the corresponding CDF. We are currently investigating the mathematical properties of other distributions, such as the Gamma distribution, to compensate in these cases.

In many cases we can adequately represent long-tailed behavior using a normal distribution as long as some precautions are taken. Using a normal distribution centered around the median value instead of the mean allows the most common value to be represented more accurately. Also, care must be taken so that predictions making this assumption are not used as absolute guarantees of system behavior. By limiting the range of values, we ignore those in the long-tail, so behavior associated with the tail will not be taken into account. On average, our predictions should be accurate, however since we are ignoring data not all behavior will be adequately represented, so absolute guarantees of behavior cannot be made.

For some application or system characteristics, such as CPU load, the data may not be representative of a normal distribution as a whole, but can be viewed as several sets of data, each having a normal distribution. An example of this is Figure 6, a histogram of load data for a workstation. This data consists of three distributions, a possibly normal distribution centered at 0.94, a long-tailed distribution centered at 0.49 and another normal distribution centered at 0.33, each of which we refer to as a mode.

As long as the the application executes while the load remains within a single mode, the data over the interval can be represented as a single distribution. For longer running applications, the data would need to be represented using a bi- or tri-modal distribution, appropriately. In our experiments we found some leeway in the restriction of having the load stay within one mode. As long as the load for the majority of the runtime was within a single mode, or if time spent in a higher load mode was compensated with time spent in a lower load mode, we could still predict application behavior.

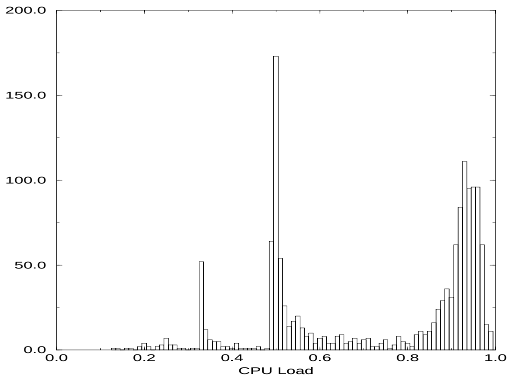

Figure 6. Load on a production workstation.

In practice, interval values are typically reported in two ways, either as an absolute range (for example, a bandwidth of 8 Mbits/second ± 2 Mbits/second) or as a percentage (for example, a load of 0.48 ± 10%). In this paper, we translate percentage ranges to absolute ranges algebraically.

3.2 Arithmetic Operations over Interval Values

Both top-level structural models and constituent component models are possibly recursive functions combining model parameters arithmetically. When we extend the domain of model parameters to include interval values we must address the issue of having them interact and produce interval value results.

For each arithmetic operation, we define a rule for combining interval values based on standard statistical error propagation methods [Bar78]. In the following, we assume that all interval values are of the form $X_i \pm v_i$ and are from normal distributions, where X_i is the mean and v_i is the standard deviation. As noted before, these techniques will hold true for distributions closed under linear combinations, not just normal distributions.

Addition/Subtraction by a point value

Addition is the most prevalent interaction in component models, so it is common to have a point value added to an interval value. For example, if the only interval value in a model were the CPU load, and we assumed that load did not affect communication, when we added together the results of a communication component and a computation component, we would be adding a point value to an interval value.

A general formula for this for interval value $X_i \pm v_i$ and point value P is

$$(X_i \pm v_i) + P = (X_i + P) \pm v_i$$

As the initial distribution over v_i was normal, so is the resulting distribution.

Subtraction can be viewed as addition of a negative value, resulting in the same interaction rule as for addition.

Addition/Subtraction by an interval value

As stated previously, addition is a prevalent operation in component models, so it is common occurrence for two interval values to be added as well. An example of this appears in the SOR communication component model, where several **PtToPt** values must be added, or in a communication component for latency and bandwidth, for example

$$Latency + \frac{MsgSize}{Bandwidth}$$

where both Latency and Bandwidth are interval values.

In general, since variance in a sum is not greater than the absolute sum of the variances, we have

$$\sum_{i=1}^{n}(X_i \pm v_i) = \sum_{i=1}^{n} X_i \pm \sum_{i=1}^{n} |v_i|$$

Alternatively, in many cases since it is unlikely that all actual values will be at the maximum of the interval, or all of them will be at the minimum, we can use a probability-based square root error computation for the error of a sum of terms

$$\sum_{i=1}^{n}(X_i \pm v_i) \approx \sum_{i=1}^{n} X_i \pm \sqrt{\sum_{i=1}^{n} v_i^2}$$

This is similar to the derivation of the standard error of means.

Subtraction of two interval values would have the same result as addition, only with one negative value for X_i.

Since normal distributions are closed under addition and subtraction, the resulting interval value would also have a normal distribution.

Multiplication/Division by a point value

Multiplying by a point value is common in component models. In many cases, it is sufficient to view only one quantity, such as load on the machines, as having a significant variance. One example of this would be the case of using a benchmark (for a dedicated system with a low variance) which is then multiplied by a CPU load on that system in production mode. For example, from the SOR computation component in the previous section

$$\mathbf{Comp_i} = NumElts_i * Op(i, Elt) * CPU_i * CPULoad_i$$

The quantity $NumElts_i * Op(i, Elt) * CPU_i$ is likely to be a point value in most situations, while $CPULoad_i$, the dynamic load on a CPU, would be represented as an interval value.

The form for calculating the resulting interval value $X_i \pm v_i$ when multiplying an interval value by a point value, P, is

$$P(X_i \pm v_i) = PX_i \pm Pv_i$$

Since normal distributions are closed under linear combinations [LM86], the resulting interval value will also have a normal distribution.

Similarly, division can be viewed as multiplication by a point value Q such that $Q = \frac{1}{P}$, again resulting in a normal distribution.

Multiplication/Division by an interval value

It is sometimes the case that multiple model parameters should be represented as interval values. For example, in the **PtToPt** component model for the SOR code, if both bandwidth and the number of elements per message are interval values, we would need a rule to address the result of their multiplication.

According to standard statistical error propagation, two interval values would multiply as follows:
$(X_i \pm v_i)(X_j \pm v_j) = X_i X_j \pm (v_i X_j) \pm (v_j X_i) \pm (v_i v_j)$

In cases where $v_i v_j$ is very small compared to the other terms (which should be common for high quality information), we can use the second rule for addition of intervals (above) for the resulting general form

$$\prod_{i-1}^{n}(X_i \pm v_i) \approx \prod_{i-1}^{n}(X_i) \pm \prod_{i-1}^{n} X_i \sqrt{\sum_{i=1}^{n}\left(\frac{v_i}{X_i}\right)^2}$$

Likewise, the rule for the division of two interval values is

$$\frac{X_i \pm v_i}{X_j \pm v_j} \approx \frac{X_i}{X_j} \pm \frac{X_i}{X_j}\sqrt{\left(\frac{v_i}{X_i}\right)^2 + \left(\frac{v_j}{X_j}\right)^2}$$

However, we run into difficulties with both multiplication and division of multiple interval values because the resulting interval value does not have a normal distribution: it is long-tailed. In many circumstances, we can approximate it it with a normal distribution and ignore the tail with little impact on the final prediction results. However, care must be taken in making this assumption, as described in the last subsection for Figure 4.

3.3 Group Operations over Interval Values

The combination of interval values for operations over a group needs to be addressed in a situation dependent fashion. An example of an operation in this class is the *Max* function. Depending on how conservative we require the prediction to be, different approaches may be used in combining interval values.

For example, we might have three interval values: A = 4 ± 0.5, B = 3 ± 2, and C = 3 ± 1. Depending on the situation, *Max* could be calculated by chosing the largest mean value, or by selecting the longest possible value. That is, if a conservative prediction were needed, we might use value B for the maximum, since it is at most 5, as opposed to value A which is at most 4.5. If a less conservative prediction were satisfactory it might be more accurate to use value A as the maximum since on average it should be the maximum value of the set. This function, and other group functions, would need to be defined by the scheduler or user that is being supplied the predictions.

We have defined how interval values for model parameters can be used to find a prediction interval for an application. We will illustrate how interval values for predictions reflect the probable range of application behavior in the next section.

4 Experimental Verification

We illustrate this methodology using a distributed successive over-relaxation (SOR) code on a network of workstations, and evaluating the accuracy of our predictions using both point values and interval values for load information.

The dynamic CPU load information we use is supplied by the Network Weather Service (NWS) [Wol96]. The NWS provides software for monitoring and predicting the "weather" on networked resources using sensors to dynamically probe and record information on CPU usage, latency, bandwidth, and memory information. Then, using a set of statistical methods it makes several predictions for the next state of the system, dynamically choosing the best predictor. This system allowed us to have very accurate information about the CPU load on our machines as well as the variance of that load.

The SOR code was implemented over a production network of Sparc workstations, running both SunOS and Solaris. Note that the actual execution times vary from run to run due to the variance of the load over the execution time for each problem size. Figures 7 and 8 both show our prediction interval and actual run times for the SOR code when the load on the slowest machine varied between 0.48 ± 0.04 over the run of the problem size for the majority of the execution time.

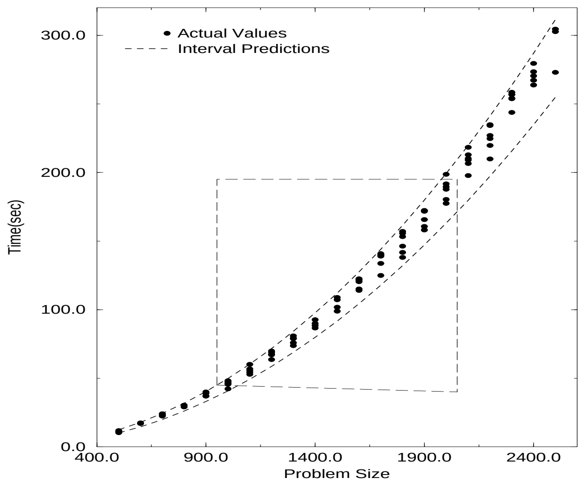

Figure 7. Actual times versus model times for SOR application.

In practice we found that the load on the slowest machine for a not insignificant percentage of the run time could be much higher or much lower, but our predictions were robust enough to include this behavior. Figure 8 shows a close up of the actual values, the point value prediction and the interval value prediction for problem sizes in the dotted rectangle of Figure 7.

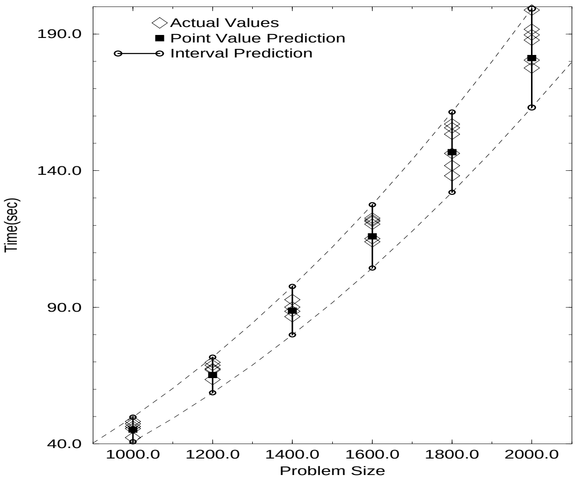

Figure 8. Actual times versus model times for the SOR application, for selected portion of Figure 7.

While load behavior is not strictly normal, this assumption did not affect the validity of our predictions. The point predictions are within 10% of the actual execution values for the problem sizes, however the interval predictions more accurately establish the range of execution behaviors for this application.

5 Conclusion

We have shown that on production distributed parallel systems, application and system characteristics are often more accurately represented as interval values. Using these interval values in models results in predictions of a range of application execution behavior. These intervals provide valuable additional information for schedulers and users to improve the overall execution time of distributed parallel applications.

Acknowledgements

Thanks to Allen Downey, UCB and SDSC, and Rich Wolski, UCSD, for many useful discussions and commenting on a previous draft of this paper. Silvia Figueira, UCSD, supplied the original implementation of the SOR code. The Network Weather Service implementation used in our experiments is due to Neil Spring, UCSD and SDSC, and originally developed by Rich Wolski. Further information on this project is available on-line at `http://www.cs.ucsd.edu/users/jenny/research.html`, and for the AppLeS project at `http://www.cs.ucsd.edu/groups/hpcl/apples/apples.html`.

References

[Bar78] B. Austin Barry. *Errors in Practical Measurement in Science, Engineering and Technology.* John Wilsey & Sons, 1978.

[BW96] Fran Berman and Richard Wolski. Scheduling from the perspective of the application. In *Proceedings of the Fifth IEEE International Symposium on High Performance Distributed Computing*, 1996.

[BWF+96] Fran Berman, Richard Wolski, Silvia Figueira, Jennifer Schopf, and Gary Shao. Application-level scheduling on distributed heterogeneous networks. In *Proceedings of SuperComputing '96*, 1996.

[LM86] Richard J. Larsen and Morris L. Marx. *An Introduction to Mathematical Statistics and Its Applications*, chapter Chapter 7.3 Linear Combinations of Normal Random Variables. Prentice-Hall, second edition, 1986.

[Sch97] Jennifer M. Schopf. Structural prediction models for high-performance distributed applications. In *Proceedings of the Cluster Computing Conference (CCC '97)*, 1997. Also available as `http://www.cs.ucsd.edu/users/jenny/CCC97/index.html`.

[Wol96] Rich Wolski. Dynamically forecasting network performance using the network weather service. Technical Report TR-CS96-494, University of California, San Diego, Computer Science and Engineering Deptartment, October 1996. Also available at `http://www-cse.ucsd.edu/users/rich/publications.html`.