Resource Signal Prediction and Its Application to Real-time Scheduling Advisors (Thesis Summary)

Peter August Dinda
pdinda@cs.cmu.edu

February 6, 2000

Abstract

A distributed interactive application spawns resilient real-time tasks with known resource requirements in response to aperiodic user actions. When running in a shared computing environment that supports neither reservations nor globally-respected priorities, such an application must carefully choose which host runs a task in order to increase the chances that the task's deadline will be met.

A real-time scheduling advisor is middleware that the application can use to find the most appropriate host for the task. In addition to recommending a host, the advisor also predicts the running time of the task on that host. The application uses this feedback to modify the task's resource requirements or deadline until a host is found where the task will meet its deadline with high probability.

This dissertation recommends basing real-time scheduling advisors on the explicit prediction of resource signals, which are easily measured, time-varying, scalar quantities that are strongly correlated with resource availability. This resource-oriented approach has numerous advantages. It scales well, makes decisions based on up-to-date information, can support other forms of adaptation advisors, and can easily leverage advances in statistical signal prediction techniques. However, resource signal predictions exist at considerable remove from predictions of application performance.

To show that this gap can be spanned, this dissertation describes the design, implementation, and performance evaluation of a prototype real-time scheduling advisor that is based on the prediction of host load signals. I have found that, despite its complex properties, which include self-similarity and epochal behavior, host load can be usefully predicted using linear time series models. These models have sufficiently low overhead to be used in practice, and I have developed a toolkit to make it easy to do so. Furthermore, I have devised an algorithm that uses host load predictions to compute a confidence interval for the running time of a task on a particular host. My real-time scheduling advisor uses these confidence intervals to provide useful recommendations to applications. Each layer of this online system has been evaluated using real host load signals and workloads.

1 Introduction

Users demand responsiveness from interactive applications such as scientific visualization tools, image editors, modeling tools based on physical simulations, and games. Such applications react to aperiodically arriving messages that arise from user actions. In response to each message, the application program executes a task whose computation creates visual and aural feedback for the user. This feedback helps determine the subsequent actions of the user, and thus subsequent tasks. The aperiodicity in message arrival is a result of having a "human in the control loop." To be responsive, the application must execute the task induced by each message as quickly as the user reasonably expects. This timeliness requirement can be easily expressed as a deadline for the task. The application is resilient in the face of missed deadlines.

At one point, creating applications that were responsive in this sense was relatively easy since the user's machine was very predictable from the point of view of the programmer. Today, however, the user's machine is becoming increasingly less predictable due to operating system jobs, daemons, other users' jobs, and the like. These external factors make the actual computation rate the programmer can expect vary in complex ways. Further, the user's machine may simply not be fast enough or have enough memory to perform some computations responsively.

On the other hand, the user's machine is no longer alone — there are many other hosts on the local area network which it can talk to. As the overheads of remote execution facilities such as remote procedure call (RPC) systems, object request brokers (ORBs), and distributed shared memory (DSM) systems decline, it becomes more tempting to execute tasks remotely to achieve the responsiveness that interactive applications require. Interactive applications are thus becoming distributed interactive applications. Unfortunately, the overwhelming majority of networks and hosts do not provide any sort of resource reservations, or even priority-based scheduling that a programmer could build upon to make a group of hosts behave in a more predictable fashion than an individual host. However, these shared, unreserved environments can be measured with increasingly sophisticated tools.

The ability to run a task on any host in the environment greatly increases the *opportunity* for the task to meet its deadline. However, to exploit this opportunity, the application must *choose* an appropriate host, which can be difficult. A real-time scheduling advisor is a middleware tool that advises the application as to the host where the task's deadline is most likely to be met. It may also provide additional information, such as the predicted running time of the task on the proffered host, which the application can use to adapt in other ways.

A real-time scheduling advisor bases its advice on resource measurements, the application's characterization of the task's resource demands, and the required deadline. Because reservations are unavailable in the computing environment, this advice comes with no guarantees — a real-time scheduling advisor operates on a best-effort basis. The usefulness of this service then depends on its measured performance. This dissertation shows that the measured performance of a real-time scheduling advisor running in real computing environments can be formidable. The real-time scheduling advisors we developed can greatly increase the probability that a task's deadline is met. Furthermore, they can accurately predict the performance of tasks *before* they are run, thus giving the application a chance to adapt in different ways when insufficient resources are available to meet the original deadline. Finally, they can introduce appropriate randomness into their scheduling decisions, thus allowing advisors to operate obliviously of each other with a low probability of disastrous interaction.

The design space for real-time scheduling advisors is vast, but most designs involve predicting the performance, either explicitly or implicitly, of the task on each of the prospective hosts. The performance is determined predominantly by resource availability. Designs that use explicit prediction are sub-divided into resource-oriented prediction approaches and application-oriented prediction approaches. Resource-oriented approaches predict future resource availability using information available about the resource. These pre-

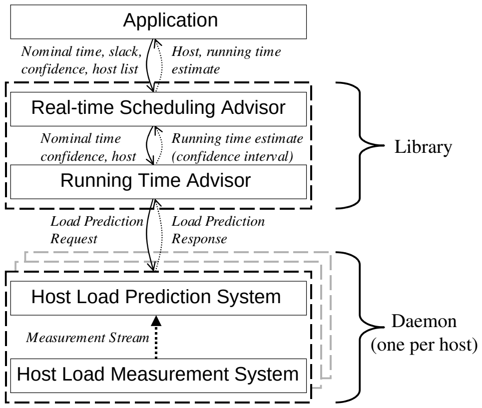


Figure 1: The structure of the resource-prediction-based real-time scheduling advisor

dictions of resource availability, and the task's resource demands are then supplied to a model that estimates the task's performance. Application-oriented approaches predict task performance directly using application information such as the performance of previous tasks.

This dissertation argues for basing real-time scheduling advisors on explicit resource-oriented prediction, specifically on the prediction of resource signals. This approach has much to recommend it over the application-oriented prediction approach: It scales better; Multiple applications or nodes of a single application can easily share the same predictions; It operates independently of application execution and thus can always provide the latest information about any resource; It can provide the basis for other kinds of scheduling advisors and quality of service predictors; It can more easily leverage advances in the statistical signal processing and time series analysis communities.

In contrast to the application-oriented approach, the resource-oriented approach predicts quantities that are at some remove from those that concern the application. This dissertation demonstrates that it is possible to span this gap between resource predictions and task performance. This enables effective real-time scheduling advisors based on explicit resource-oriented prediction.

The core of the dissertation describes the design, implementation, and evaluation of a real-time scheduling advisor for compute-bound tasks. The advisor is based on explicit resource-oriented prediction using the techniques of linear time series analysis to predict available CPU time. The resource signal is host load (specifically, the Digital Unix five second load average), which we found to correlate very well with available CPU time.

Our explicit resource-oriented prediction approach is based on statistical signal processing of resource signals. Resource signals are sequences of periodic measurements that are strongly correlated with the availability of some underlying resource. We exploit linear time series analysis (for the most part) to characterize resource signals and find appropriate predictors for them. We developed a methodology for the process and a toolkit that facilitates carrying out the methodology and implementing on-line resource prediction systems for new resource signals.

2 Prototype real-time scheduling advisor

The core of this dissertation describes the design, implementation, and evaluation of a prototype real-time scheduling advisor. The prototype advisor schedules compute-bound tasks using explicit prediction of *host load signals*, specifically, the Digital Unix five second load average, sampled at 1 Hz. The low overhead and high performance of this system demonstrate the power of the explicit resource-oriented prediction approach.

Figure 1 shows the architecture of the system. At the highest level, the system is divided into two parts: a library, which is bound to a single application, and a daemon, one of which runs indendently on each host and can serve multiple applications. The library and the daemon can be further decomposed into independent components that can communicate in a number of ways. In the figure, a dashed arrow represents stream-oriented communication between components, while a symmetric pair of arrows represents request-response communication.

The daemon consists of a host load measurement system, which periodically measures host load, and a host load prediction system, which transforms each new measurement into a qualified prediction of future measurements. Each of the values in the prediction is qualified by an estimate of its error and how it correlates with the error of the other values. The prediction system uses linear time series models to predict host load. It continuously monitors the prediction accuracy, refitting the model when the accuracy drops below a threshold. The daemon stores a short history of the predictions and makes them available via a request-response protocol. This matches the periodic nature of the measurement and prediction systems with the aperiodic nature of application requests.

The library is easiest to describe from the perspective of an application request. A request consists of a nominal running time (t_{nom}) , a maximum slack ($slack$), a confidence level, and a list of hosts. The confidence level, which ranges from zero to one, tells the real-time scheduling advisor the minimum probability of meeting the deadline that the application requires. The real-time scheduling advisor transforms this single application request into requests for predictions of the task's running time on each of the hosts. Each of these requests consists of the nominal time and the confidence level. To answer a request, the running time advisor acquires the latest host load prediction from the host's daemon. It then uses a statistical model of the host's scheduler to transform the host load prediction, the nominal time of the task, and the confidence level into a prediction of the running time of the task on the host. The prediction contains both an expected running time and a confidence interval for the running time. After acquiring running time predictions for each of the hosts, the real-time scheduling advisor chooses a host at random from among those hosts whose running time predictions are less than the deadline. If no such host exists, it chooses the host with the minimum expected running time. The chosen host and the prediction of the running time of the task on that host are returned to the application. The application can then choose whether to accept the solution or to pose a different request.

The application or other middleware tools can also use the system at lower levels. For example, other kinds of schedulers can be based on the running time advisor, or can use host load predictions directly. The system is also extensible in a number of ways. It is easy to construct prediction systems for other kinds of resources, and it is easy to add support for new predictive models. The components of the system can also be arranged differently, using various transports to communicate over the network.

3 Summary and contributions

The following list summarizes the work described in this dissertation and the insights and contributions that it has produced.

- We identified the class of distributed interactive applications. In these applications, computation takes the form of tasks that are initiated by user action, which occurs aperiodically. Tasks are executed sequentially because they provide feedback that determines the user's next action. Because of this interactivity, consistent responsiveness is paramount. This requirement can be expressed in the form of a real-time deadline on each task's running time. However, the applications are resilient in the face of missed deadlines. Furthermore, the applications have been built with distributed operation in mind.

When a task arrives, a distributed interactive application knows its resource demands and its deadline, and can choose which host to run it. In addition, it may also be able to adapt the resource demands of the task by changing the computation it performs. (Chapter 1)

- We provided four examples of distributed interactive applications: QuakeViz, OpenMap, Acoustic CAD, and an image editor. We measured the CPU demands of representative QuakeViz applications, which we use as the basis of our randomized evaluations in this dissertation. (Chapter 1)

- We focused on running distributed interactive applications on typical shared, unreserved distributed computing environments. These environments do not provide resource reservations, admissions control, or a global notion of priority. Because of this lack of infrastructure, which seems likely to persist into the foreseeable future, traditional distributed soft real-time systems can not satisfy the needs of these applications. (Chapter 1)

- We introduced the concept of real-time scheduling advisors. A real-time scheduling advisor is a middleware tool that advises the application as to the host where a task's deadline is most likely to be met. The advice may be supplemented with additional information, such as the expected running time of the task. However, it is strictly best-effort. In essence, the real-time scheduling advisor guides the application in using its adaptation mechanisms (the choice of host, and perhaps also the ability to change resource demands) to help the task meet its deadline. The combination of these semantics, the characteristics of distributed interactive applications, and the shared, unreserved distributed computing environments we target defines the scheduling problem that real-time scheduling advisors solve. (Chapter 1)

- We explored the design space for real-time scheduling advisors and concluded that the prediction of resource availability, either implicit or explicit, is the basis of most designs. Explicit prediction approaches can furthermore be divided into application-oriented approaches, in which the performance of the application's tasks are probes on resource availability, and resource-oriented approaches, in which resources are monitored separately from the application. (Chapter 1)

- We identified the tradeoffs between the application-oriented approach and the resource-oriented approach. The resource-oriented approach is more scalable, makes decisions based on more up-to-date information, can support other forms of adaptation advisors, and can easily leverage advances in statistical prediction techniques. However, unlike the application-oriented approach, the resource-oriented approach produces and uses resource availability information that exists at considerable remove from the performance of the application. If this gap between the resources and the application can be spanned, then the resource-oriented approach is preferable. The core of the thesis shows that it can indeed be spanned. (Chapter 1)

- We explored the performance of the application-oriented approach, developing a predictive algorithm that provides near-optimal performance in simulation for a simplified version of the scheduling problem. The disadvantage of the application-oriented approach is not in its performance, but in its scalability problems. (Appendix A)

- We defined the notion of a resource signal, which is a scalar-valued, discrete-time signal that correlates with the availability of a resource. The resource we focused on in the thesis is CPU time. The resource signal we used is host load, specifically, the Digital Unix five second load average. (Chapter 1)

- We designed the framework for a real-time scheduling advisor (Figure 1) that is based on the prediction of host load signals. The core of the thesis fills out the structure shown in the figure. The framework can be extended to include other resources and other forms of advisors, as we describe later in this chapter. (Chapter 1)

- We developed a methodology for attacking the problem of predicting resource availability. The main idea behind the methodology is to transform a specific resource prediction problem into a general time series prediction problem as early as possible, and then to apply the substantial statistical machinery that already exists to address such problems. We applied this methodology to develop the host load measurement and prediction systems shown in Figure 1. Later in this chapter, we show how we are beginning to apply it to predicting the bandwidth of network connections. (Chapter 2)

- We designed, implemented, and evaluated the RPS Toolkit, which helps to carry out the latter steps of the methodology. The early steps of the methodology, which are done off-line and which are human-intensive, are well served with existing tools. The latter steps, which are concerned with large scale, machine-intensive evaluation, and building efficient on-line prediction systems, were, however, not well served. RPS improves this situation greatly. RPS consists of sensor libraries, a time series prediction library, a communication library, prediction components, and a parallelized evaluation system. The prediction components are composed at run-time to form on-line resource prediction systems. The overheads of these systems, when run at the measurement rates we typically expect to use, are miniscule. Further, they are capable of sustaining measurement rates 2-3 orders of magnitude higher than we typically need. RPS-based prediction systems have been incorporated into the CMU Remos [1] and BBN QuO [2] systems. (Chapter 2)

- We identified host load, specifically, the Digital Unix five second load average, as being an appropriate resource signal for CPU availability. We determined that 1 Hz was the appropriate rate at which to sample this signal. (Chapter 3)

- At two different times of the year, we collected week-long, 1 Hz traces of host load signals on a large number of different machines that we classify as production and research cluster machines, compute servers, or desktop workstations. These traces form the basis of our studies and evaluations in this disseration. (Chapter 3)

- We developed a technique called load trace playback which recreates a facsimile of the workload that a load trace measured. We used this technique to evaluate the running time and real-time scheduling advisors in Figure 1. We have also provided the playback tool and our collection of load traces to the research community as a way of producing realistic workloads. (see http://www.cs.cmu.edu/∼pdinda/LoadTraces/) Figure 2 illustrates load trace playback on a Digital Unix machine such as was used in our studies. (Chapter 5).

- We performed a detailed statistical study of the traces in order to determine the prospects for predicting host load signals. In particular, we focused on those aspects of the signal that are pertinent to prediction. This is the first study of this kind of which we are aware. We found that host load signals exhibit substantial autocorrelation structure, far beyond that which one would expect from merely the exponential smoothing the operating system applies to the signal. This suggested that linear models, which attempt to model such autocorrelation structures parsimoniously, might be appropriate for host load prediction. Figure 3 presents a time series analysis of one of our traces. (Chapter 3)

- We found that the host load signal is self-similar. This new result suggested that more complex and expensive linear models, such as ARFIMA models, might be required to predict these signals. Figure 4 shows Hurst parameter (point) estimates, computed using four different estimation techniques. (Chapter 3).

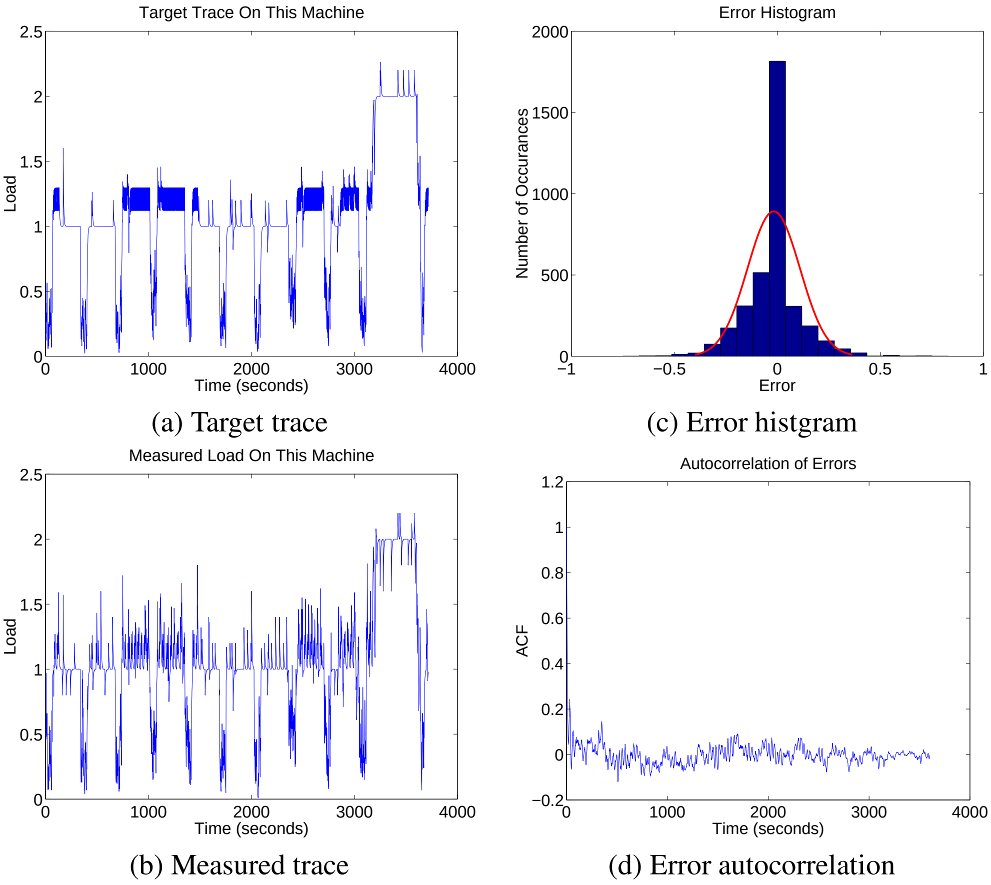


Figure 2: Trace playback on a Digital Unix Machine

- We found that the host load signal exhibits epochal behavior—the signal remains stationary for an extended period of time, and then abruptly transitions to another stationary regime. This new result argued against linear time series models because even those that explicitly model nonstationarity cannot model such abrupt transitions. Furthermore, if linear models could be used within an epoch, it might be necessary to refit the model at epoch transitions. Figure 5 shows a spectrogram of one of the load traces to illustrate this point. (Chapter 3)

- Having decided, with reservations, that linear models might be appropriate for host load prediction, we performed a large scale, randomized evaluation based on the traces to determine which kind of linear model, if any, was most appropriate. The study was unique because of its large scale, randomized approach, fine grain prediction, and focus on more sophisticated models. The surprising conclusion is that not only are linear models appropriate, but that relatively simple linear models are sufficient to provide useful predictions as far as 30 seconds into the future. We concluded that autoregressive models of order 16 or better (AR(16)) are appropriate for host load prediction. More sophisticated models have similar predictive power while requiring considerable more CPU time to use. Figure 6 shows the distribution of absolute performance (mean squared prediction error) of the different predictors for 30-second-ahead predictions on one of the more heavily loaded machines in our study. Figure 7 compares the average percentage reduction in mean squared error of the AR(16) and BM(32) models. (Chapter 4)

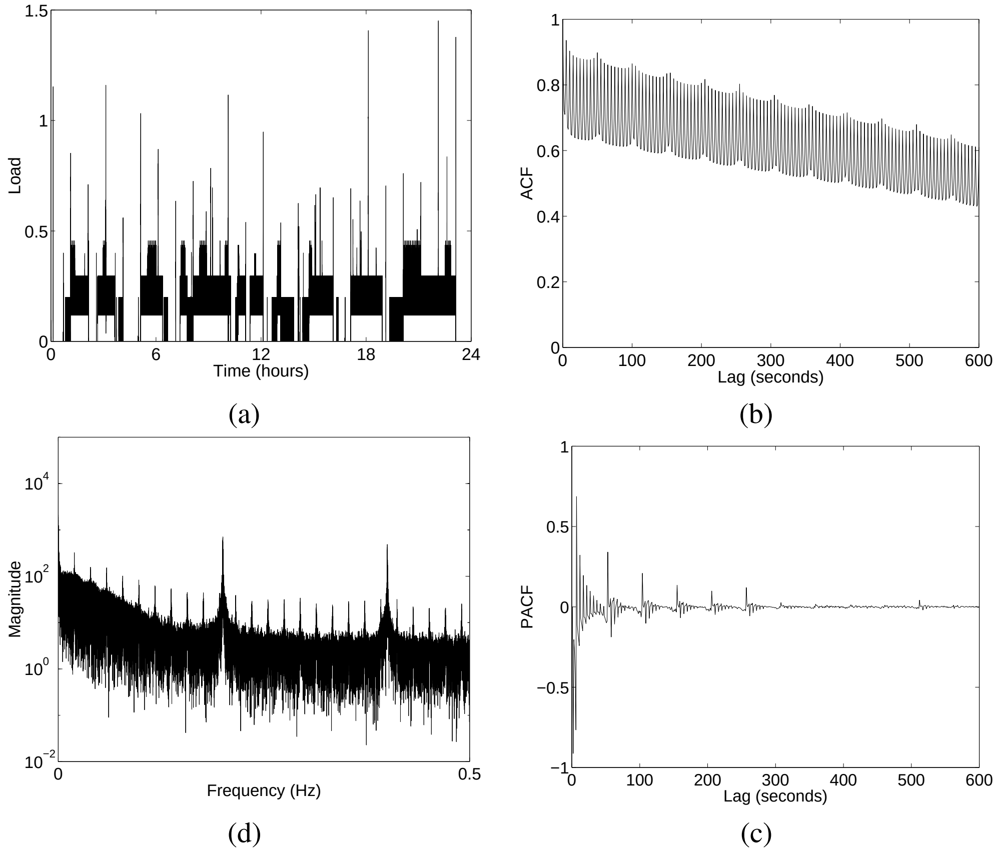
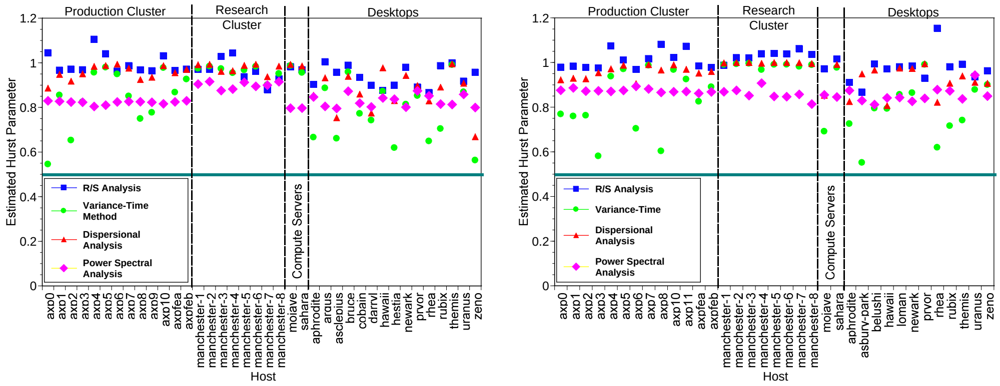


Figure 3: Time series analysis of axp7 load trace collected on August 19, 1997: (a) Load trace, (b) Autocorrelation function to lag 600 (10 minutes), (c) Partial autocorrelation function to lag 600 (10 minutes), (d) Periodogram.

- We implemented and evaluated the performance of an RPS-based host load prediction system that can be configured to use any predictive model. We use this system to evaluate the running time and real-time scheduling advisors of Figure 1. In addition to AR(16) models, we also used the LAST model (last sample as the prediction of all future samples), and MEAN model (long-term average of the signal as the prediction of all future samples) in our evaluations. For each of these models, the host load prediction system has negligible overhead. Figure 8 illustrates (a) how this system is composed out of RPS components and (b) the overhead of the system (percentage of 500 MHz Alpha 21164 CPU time) at different measurement rates. (Chapters 4, 2)

- We developed an algorithm to transform from host load predictions and a task's CPU demands to a confidence interval for the task's running time on the host. This forms the basis of the running time advisor shown in Figure 1. The transformation effectively models the Unix scheduler from the perspective of a new task. The host load predictions include estimates of prediction quality, namely the covariance matrix of the prediction errors, which form the basis of computing the confidence interval. In addition, for small tasks, it is vital to model the priority boost that the scheduler provides processes that have just finished an I/O operation such as returning from a read on a socket. We introduced a pre-processing step called load discounting to account for this effect. (Chapter 5)



(a) (b)

Figure 4: Hurst parameter point estimates: (a) August, 1997 traces, (b) March, 1998 traces.

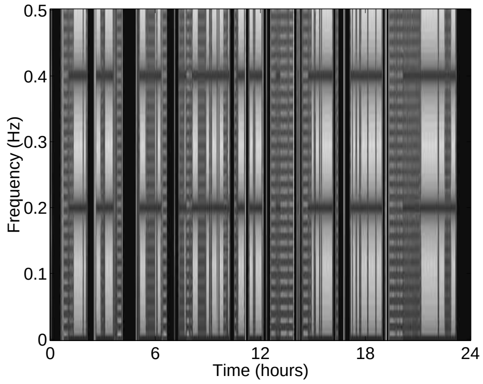


Figure 5: Spectrogram representations of load for host axp7 on August 19, 1997.

- We evaluated the running time advisor by running a large number of randomized testcases on hosts whose workloads were played back from the load traces we collected. The main result is that our system, using a strong host load predictor such as the AR(16) predictor, does indeed compute reasonable and useful confidence intervals for the running time of tasks. In comparison to LAST and MEAN, the benefits of AR(16) depend on the how heavily loaded the host is and on the nominal time of the task. On heavily loaded hosts, AR(16) models produce appropriately wider confidence intervals that correctly capture the increased variability in running time. On lightly loaded hosts, AR(16) models produce narrower confidence intervals that still cover the desired fraction of tasks. Figures 9 and 10 illustrate the quality of confidence intervals computed on typical low load and high load hosts, respectively. (Chapter 5)

- We developed an algorithm to select an appropriate host for meeting a task's deadline given confidence intervals for the running time of the task on different hosts. The algorithm attempts to introduce as much randomness as possible into the host selection process while still choosing a host on which the deadline will be met with the user's specified probability. This algorithm is the basis for the real-time scheduling advisor component of Figure 1. (Chapter 6)

- We evaluated the performance of the real-time scheduling advisor by running a large number of randomized testcases on hosts whose workloads were played back from the load traces we collected. We compared the advisor to two other approaches: selecting a host at random, and selecting a host whose

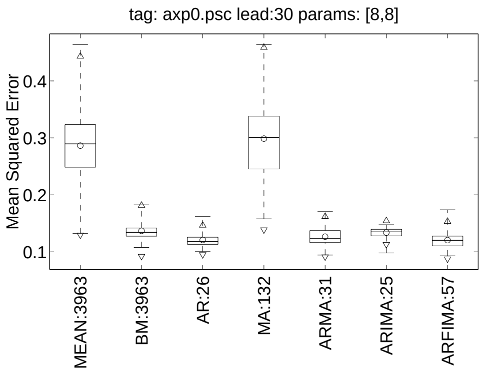


Figure 6: Axp0, 30 second lead, 8 parameters

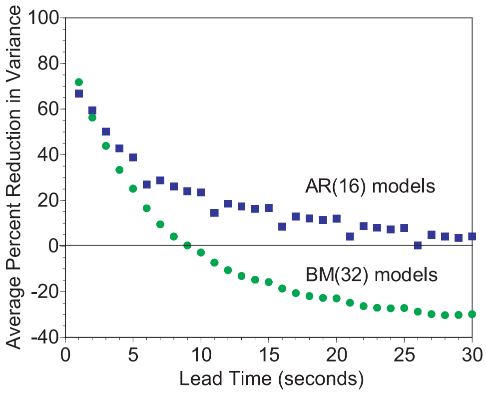
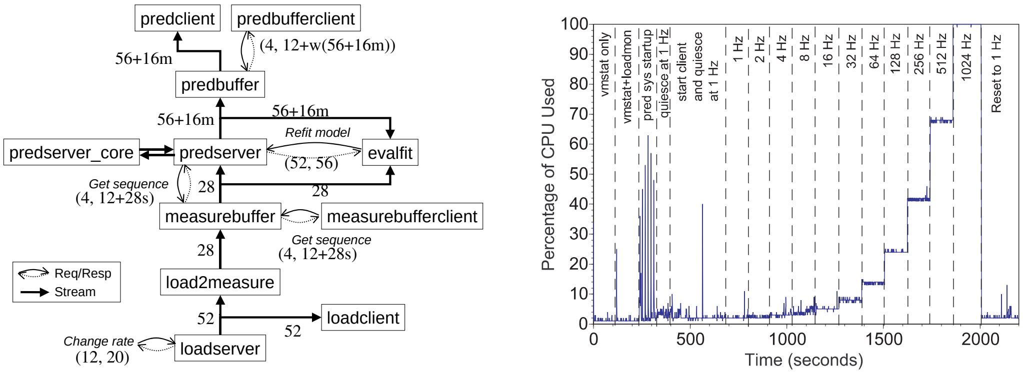


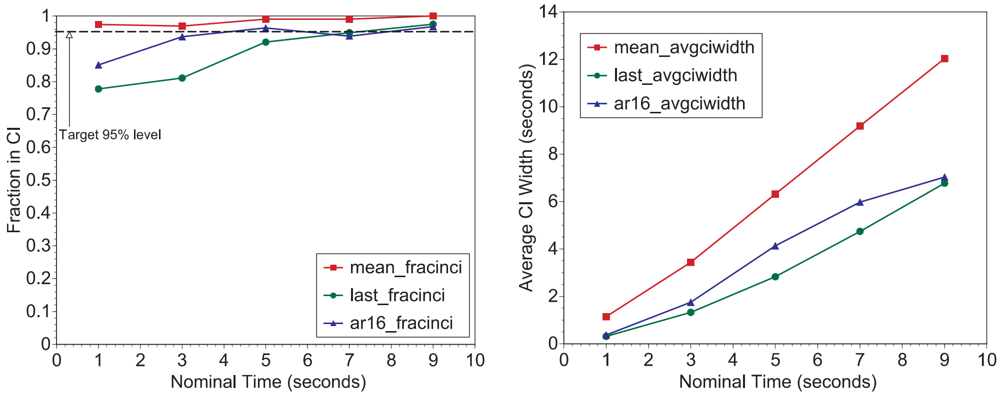
Figure 7: Paired, normalized comparisons of AR(16) and BM(32) models by lead time.

host load was measured to be the lowest. The measurement-based approach performed considerably better than the random approach in terms of the probability that a deadline would be met. In comparison to measurement, the prediction-based real-time scheduling advisor always performed at least as well by this metric, and considerably better in resource constrained situations. When it is just possible to meet a deadline, the prediction-based advisor significantly increases the probability that it will be met. In addition, the prediction-based advisor informs the application of the confidence interval for the task's running time on the host it has chosen. This means that the application knows, with considerable confidence, whether the deadline will be met *before* it runs the task, enabling it to adapt the task's CPU demands or deadline, if necessary. This is a capability that does not exist with the purely measurement-based or random scheduling approaches. Finally, the prediction-based approach is able to introduce considerable randomness into its scheduling decisions while providing the application with good performance. This means that it is much less likely to interact badly with other advisors running in the system. The additional benefits over the purely measurement-based approach come at a miniscule additional cost given the simple and cheap AR(16) model and the low overheads of RPS. Figure 11 shows the performance, as a function of slack, of the real-time scheduling advisor when scheduling to a group of four heavily loaded machines: (a) shows the probability that the deadline is met, (b) shows the probability that the deadline is met given that the advisor thinks it can be met, and (c) shows the expected number of hosts from which the scheduler can choose (ie, how much randomness it can introduce). (Chapter 6)



(a) System (b) Overhead for different measurement rates

Figure 8: Online host load prediction system composed out of the RPS prediction components described in Chapter 2.



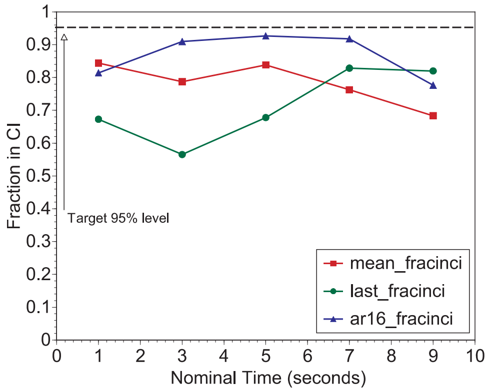
(a) Coverage (b) Span

Figure 9: Confidence intervals on a typical low load host.

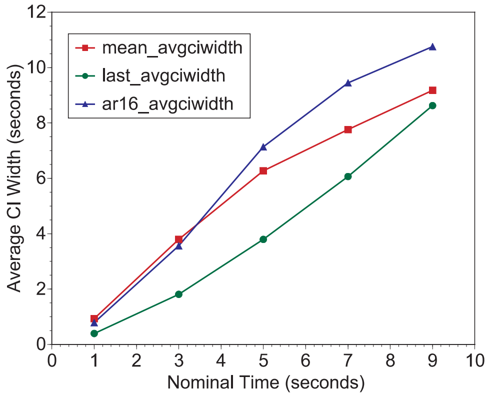
- The design, implementation, performance, and efficiency of the prototype host load prediction-based real-time scheduling advisor show that it is feasible to base real-time scheduling advisors on the explicit prediction of resource signals. Furthermore, we have shown that these advisors can provide a useful service to distributed interactive applications.

4 Conclusions and future work

This dissertation has introduced the concept of real-time scheduling advisors and argued for basing them on explicit resource-oriented prediction, specifically on the prediction of resource signals. To support the utility of such advisors, we identified the characteristics of a class of applications, distributed interactive applications, which can benefit from them, and provided examples of applications in this class. To support basing real-time scheduling advisors on the explicit prediction of resource signals, we showed that, in contrast to designs which are based on application-oriented prediction, a resource-oriented design is more scalable, makes decisions based on more up-to-date information, can support other forms of adaptation advisors, and can easily leverage advances in statistical prediction techniques. However, unlike the application-oriented approach, the resource-oriented approach produces and uses resource availability information that exists at considerable remove from the performance of the application. The core of the dissertation shows by example



(a) Coverage



(b) Span

Figure 10: Confidence intervals on a typical high load host.

that it is nonetheless possible to span this gap.

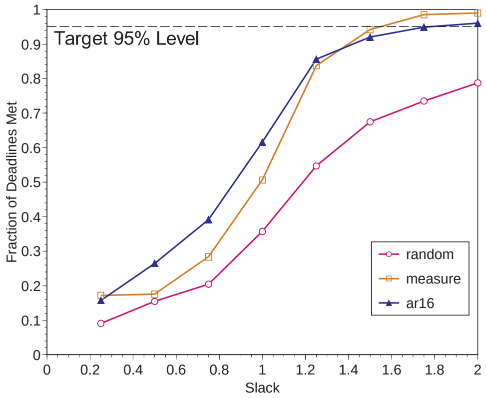
To show that an effective real-time scheduling advisor can be based on the prediction of resource signals, we designed, implemented, and evaluated a prototype system that uses host load prediction based on high-order autoregressive time series models to schedule compute-bound tasks. We evaluated each layer of the design. We found that the system provides effective scheduling advice to applications. Furthermore, because the system is resource-oriented and because of the careful compartmentalization of its functionality, it is not merely a real-time scheduling advisor, but can also provide several other kinds of useful information to applications, other forms of adaptation advisors, and other middleware. These include statistical estimates (confidence intervals) for the running of tasks or for available time on the processor, predictions of the host load signal along with estimates of prediction error, and raw measurements of the host load signal, including histories.

We have also introduced a resource signal methodology for attacking the problem of predicting the availability of a new resource, as well as a toolkit to help carry out the methodology. The ultimate result of using the methodology and the toolkit is a high performance on-line prediction system for the resource. We applied the methodology and the toolkit to determine how to predict the host load signal. The resulting host load prediction system is used as the basis of the real-time scheduling advisor of Figure 1 and has also been incorporated in the CMU Remos resource measurement system [1], and into BBN's QuO distributed object quality of service system [2].

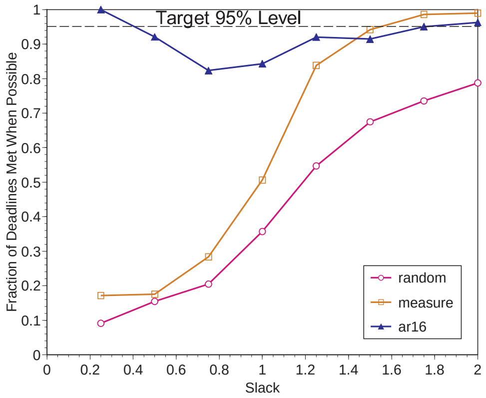
The structure of the real-time scheduling advisor, as shown in the Figure 1, provides a framework for our future work described below. This dissertation has developed effective real-time scheduling advisors for compute-bound tasks. We intend to generalize the system so that it can also support tasks with substantial communication requirements. The approach is to apply the resource signal methodology to signals that correspond to the availability of network bandwidth and latency, developing prediction systems for these signals, and then to incorporate their predictions into the real-time scheduling advisor. Over the longer term, we want to consider the prediction of other resource signals and the development of other kinds of adaptation advisors. We are also very interested in methods to capture or generate the workloads of interactive applications.

References

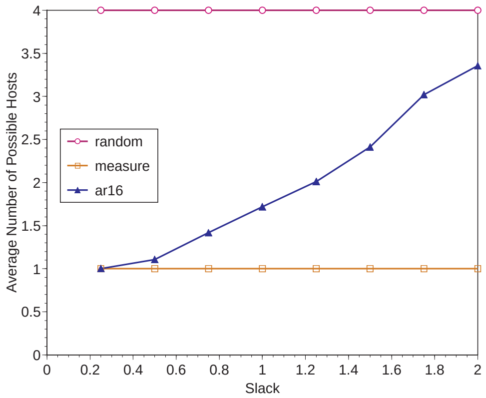
[1] Bruce Lowekamp, Nancy Miller, Dean Sutherland, Thomas Gross, Peter Steenkiste, and Jaspal Subhlok. A resource monitoring system for network-aware applications. In *Proceedings of the 7th IEEE Interna-*



(a) fraction of deadlines met



(b) fraction of deadlines met when possible



(c) number of possible hosts

Figure 11: Scheduling performance versus slack, 4LS scenario, 0 to 10 second tasks.

tional Symposium on High Performance Distributed Computing (HPDC), pages 189–196. IEEE, July 1998.

[2] John A. Zinky, David E. Bakken, and Richard E. Schantz. Architectural support for quality of service for CORBA objects. *Theory and Practice of Object Systems*, 3(1):55–73, April 1997.