Host Load Prediction Using Linear Models *

Peter A. Dinda
School of Computer Science
Carnegie Mellon University
5000 Forbes Avenue
Pittsburgh, PA 15213
E-mail: pdinda@cs.cmu.edu

David R. O'Hallaron
School of Computer Science and
Department of Electrical and Computer Engineering
Carnegie Mellon University
5000 Forbes Avenue
Pittsburgh, PA 15213
E-mail: droh@cs.cmu.edu

This paper evaluates linear models for predicting the Digital Unix five-second host load average from 1 to 30 seconds into the future. A detailed statistical study of a large number of long, fine grain load traces from a variety of real machines leads to consideration of the Box-Jenkins models (AR, MA, ARMA, ARIMA), and the ARFIMA models (due to self-similarity.) We also consider a simple windowed-mean model. The computational requirements of these models span a wide range, making some more practical than others for incorporation into an online prediction system. We rigorously evaluate the predictive power of the models by running a large number of randomized testcases on the load traces and then data-mining their results. The main conclusions are that load is consistently predictable to a very useful degree, and that the simple, practical models such as AR are sufficient for host load prediction. We recommend AR(16) models or better for host load prediction. We implement an online host load prediction system around the AR(16) model and evaluate its overhead, finding that it uses miniscule amounts of CPU time and network bandwidth.

Keywords: host load prediction, performance prediction, linear time series models

AMS Subject classification: Primary 68M14, Secondary 68M20

1. Introduction

Consider an application program that wants to schedule a compute-bound soft real-time task in a typical distributed computing environment [8]. By using mechanisms such as CORBA [25] or Java RMI [29], the task can be executed on any of the host machines in the network. Given this freedom, the application can choose the host on which the task's deadline is most likely to be met.

If the application could predict the exact running time of the task on each of the hosts, choosing a host would be easy. However, such predictions are unlikely to be exact due to the dynamic nature of the distributed system. Each of the hosts is acting independently, its vendor-supplied operating system scheduling tasks initiated by other users, paying no special attention to the task that the application is trying to schedule. The computational load on each of the hosts can vary drastically over time.

Because of this dynamic nature, the application's predictions of the task's running time on each of the hosts have confidence intervals associated with them. Better predictions lead to smaller confidence intervals, which makes it easier for the application to choose between the hosts, or to decide how likely a particular

* Effort sponsored in part by the Advanced Research Projects Agency and Rome Laboratory, Air Force Materiel Command, USAF, under agreement number F30602-96-1-0287, in part by the National Science Foundation under Grant CMS-9318163, and in part by a grant from the Intel Corporation.

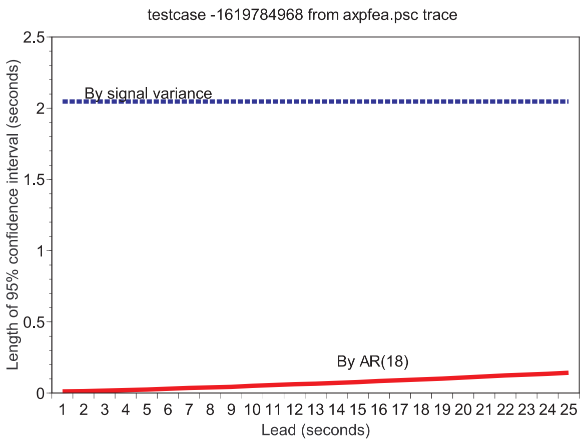


Figure 1. Benefits of prediction: length of 95% confidence interval for running time of a one second task on an interactive cluster machine with long term load average of 0.17.

host is to meet the deadline.

The running time of a compute-bound task on a host is strongly related to the computational load on the host. If we could predict the load that a task would encounter on some host, we could predict the running time on that host. Better load predictions lead to better predictions of running time and thus to smaller confidence intervals. For this reason, we concentrate on host load prediction here.

Better load predictions can indeed lead to drastically smaller confidence intervals on real hosts, even lightly loaded ones. For example, Figure 1 plots, for one such host, the length of the confidence interval for the running time of a one second task as a function of how far ahead the load predictions are made. The confidence intervals for a predictive AR(18) model (described in Section 4) and for the raw variance of the load signal itself are represented. Notice that for up to 25 seconds into the future, the AR(18) model provides a confidence interval of less than 200 ms while the raw load signal provides a confidence interval of over two seconds.

The *existence* of clear benefits of this kind motivates the following questions: Is host load *consistently* predictable or are examples like Figure 1 merely flukes? If host load is indeed consistently predictable, what classes of predictive models are appropriate for predicting it? What are the differences between these different classes in terms of their predictive power and their computational overheads? What class can be recommended for use in a real system? Finally, what is the overhead of an online host load prediction system that uses an appropriate model? This paper describes the results of a large scale, real world study to provide statistically rigorous answers to the questions of predictability. In addition, we present measurements of an online host load prediction system in order to answer the questions of overhead.

We found that host load is, in fact, consistently predictable to a very useful degree from past behavior, and that simple, practical linear time series models are sufficiently powerful load predictors. These results are somewhat surprising because load has complex behavior and exhibits properties such as self-similarity and epochal behavior that suggest that more complex models would be more appropriate. As far as we are aware, this is the first study to identify these properties of host load and then to rigorously evaluate the practical predictive power of linear time series models that both can and cannot capture them. Furthermore, our evaluation approach, while much more computationally intensive than those of traditional time series analysis, is unbiased and therefore lets us realistically gauge how the models actually behave when confronted with messy real world host load measurements. Our study is also unique in focusing on predictability on the scale of seconds as opposed to minutes or longer time scales, and thus is perhaps most useful in the context of interactive applications such as scientific visualization tools [1] running on distributed systems. Finally, our evaluation uses a toolkit for constructing distributed resource prediction services, RPS [12]. RPS has been used to build prediction systems for the Remos resource measurement system [23] and for BBN's QuO distributed object quality of service system [32].

We began by choosing to measure host load by the Digital Unix five-second load average. We found that this measure, which can be easily acquired by user-level programs, is closely related to the running time of short compute-bound tasks (Section 2.) We collected a large number of 1 Hz benchmark load traces, which capture all the dynamics of the load signal, and subjected them to a detailed statistical analysis, which we summarize in Section 3.

On the one hand, this analysis suggested that linear time series models such as those in the Box-Jenkins [6] AR, MA, ARMA, and ARIMA classes might be appropriate for predicting load. On the other hand, the existence of self-similarity-induced long-range dependence suggested that such models might require an impractical number of parameters or that the much more complex ARFIMA model class [20,16,4], which explicitly captures long-range dependence, might be more appropriate. Since it is not obvious which model is best, we empirically evaluated the predictive power of the AR, MA, ARMA, ARIMA, and ARFIMA model classes, as well as that of a simple ad hoc windowed-mean predictor called BM and a long-term mean predictor called MEAN. BM includes LAST, which predicts that the last measurement of load will continue in perpetuity, as a degenerate case. We describe these model classes, RPS's implementations of them, and the computational requirements these implementations in Section 4.

Our evaluation methodology, which we describe in detail in Section 5, was to run randomized testcases on the benchmark load traces. The testcases (190,000 in all, or about 5000 per trace) were randomized with respect to model class, the number of model parameters and their distribution, the trace subsequence used to fit the model, and the length of the subsequent trace subsequence used to test the model. We collected a large number of metrics for each testcase, but we concentrate on the mean squared (prediction) error metric in this paper. This metric is directly comparable to the raw variance in the load signal, and, with a normality assumption, can be translated into a confidence interval for the running time of a task. Moreover, by the central limit theorem, these estimates of a model class's expected mean squared error are themselves normally distributed, and thus we can determine, to a given significance level, whether one model provides lower expected mean squared error than another. Further, we determined how much the error varies from testcase to testcase. Finally, we performed paired comparisons between the expected normalized performance of the AR, BM, MEAN, and LAST models.

The results of our evaluation are presented in Section 6. We found that host load is consistently predictable to a useful degree from past behavior. Except for the MA models, which performed quite badly, the expected mean squared error of the predictive models, computed using the unpaired comparisons, were all roughly similar, although statistically significant differences were indeed found. The differences were greater on more heavily loaded hosts. These marginal differences do not seem to warrant the use of the more sophisticated model classes (ARMA, ARIMA, ARFIMA) because their run-time costs are much higher. However, the paired comparisons, which are statistically stronger, illustrate the benefits of AR models over simple ad hoc models such as BM and LAST. Our recommendation is to use AR models of relatively high order (16 or better) for load prediction within the regime we studied.

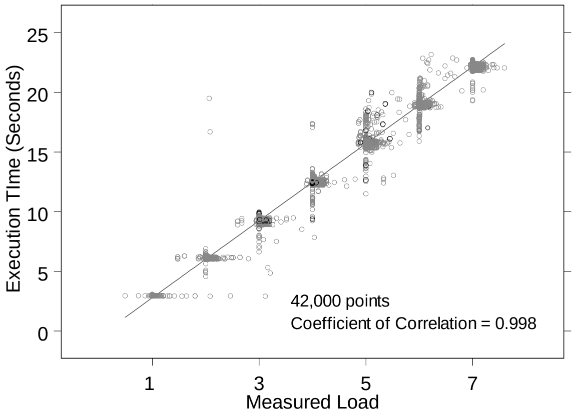


Figure 2. Relationship between average load during execution and running time.

We constructed a prototype RPS-based online host load prediction system using the AR(16) model and measured its overheads (Section 7.) The latency, from measurement to prediction, of this system is quite low. At the 1 Hz rate at which we measure host load, the system consumes nearly unmeasurable amounts of CPU time and network bandwidth, while its maximum achievable rate is three orders of magnitude higher than required. The conclusion is that such systems can not only provide useful predictions of host load, but can be made to be practical.

Section 8 puts our study in the context of other work in the area, and Section 9 summarizes and concludes with a discussion of future directions of this research.

2. Host load and running time

For CPU-bound tasks, there is an intuitive relationship between host load and running time. Consider Figure 2, which plots running time versus average load experienced during execution for tasks consisting of simple wait loops. The data was generated by running variable numbers of these tasks together simultaneously, at identical priority levels, on an otherwise unloaded Digital Unix machine. Each of these tasks sampled the Unix five-second load average at roughly one second intervals during their execution and at termination printed the average of these samples as well as their running time. It is these pairs that make up the 42,000 points in the figure. Notice that the relationship between the measured load during execution and the running time is almost perfectly linear ($R^2 > 0.99$.)

If load were presented as a continuous signal, we would summarize this relationship between running time and load as

$$\frac{t_{exec}}{1+\frac{1}{t_{exec}}\int_{t_{now}}^{t_{now}+t_{exec}} z(t)dt} = t_{nom}$$

where t_{now} is when the task begins executing, t_{nom} is the running time of the task on a completely unloaded machine, $z(t)$ is the continuous "background" load, and t_{exec} is the task's running time. In practice, we can only sample the load with some non-infinitesimal sample period Δ so we can only approximate the integral by summing over the values in the sample sequence. In this paper, $\Delta = 1$ second, so we will write such a sequence of load samples as $\langle z_t \rangle = \ldots, z_{t-1}, z_t, z_{t+1}, \ldots$ We will also refer to such a sequence as a *load signal.*

If we knew z_{t+k} for $k > 0$, we could directly compute the running time. Unfortunately, the best we can do is predict these values in some way. The quality of these predictions will then determine how tightly we can bound the running time of our task. We measure prediction quality by the *mean squared error*, which is the average of the square of the difference between predicted values and actual values. Note that there is a mean squared error associated with every *lead time* k. Two-step-ahead predictions ($k = 2$) have a different (and probably higher) mean squared error than one-step-ahead predictions ($k = 1$), and so on.

During some intervals of time, the load signal may be less predictable than others. Estimates of the mean squared error for those intervals would be larger than for other intervals. Because of this variability, care must be taken in comparing different prediction techniques. However, if we aggregate the estimates for many different intervals, the central limit theorem tells us that the resulting *expected mean squared error* is normally distributed, and thus we use this quantity in our comparisons. Intuitively, this is the mean squared prediction error one would expect from a given prediction technique confronted with a randomly chosen interval.

For the simplest prediction, the long-term mean of the signal, the mean squared error is simply the variance of the load signal. As we shall see in Section 3, load is highly variable and exhibits other complex properties, which leads to very loose estimates of running time. The hope is that more sophisticated prediction schemes will have much lower mean squared errors.

It is important to note that a prediction made with a linear time series model includes an estimate of its mean squared errors, as well as an estimate of the covariance of the errors. In other words, each prediction is qualified with an estimate of how erroneous it can be. These measures of prediction quality provide the basis of statistical reasoning using the predictions, such as choosing between two hosts based on their confidence intervals. Our evaluation is focused on the *measured* mean squared error, not on these estimates.

In this paper, we often translate (measured) mean squared errors into a confidence interval for the running time of a task. This operation is valid for short tasks which do not benefit from priority boosts when they start running. For longer tasks, computing confidence intervals requires using the covariance estimates as well as the mean squared errors [10, Chapter 5].

3. Statistical properties of load traces

The load on a Unix system at any given instant is the number of processes that are running or are ready to run, which in turn is the length of the ready queue maintained by the scheduler. The kernel samples the length of the ready queue at some rate and then exponentially averages the samples to produce a load average which can be accessed from a user program. It is important to note that while this filtering does tend to correlate the load average over the short term (shorter than the time constant of the filter), the exponential-averaging filter does expose the full spectral content of the underlying signal.

The Unix we used was Digital Unix (DUX.) DUX is interesting because the time constant is a mere five seconds, which minimizes the effect of phantom correlation due to the filter. This is especially important when gauging the efficacy of prediction techniques. While the analysis of this section and the prediction results of Section 6 use the filtered load signal as it is directly available to applications (and which correlates strongly with running time as shown in Section 2), we have also applied our analysis and prediction techniques to the "unfiltered" load signal with similar results.

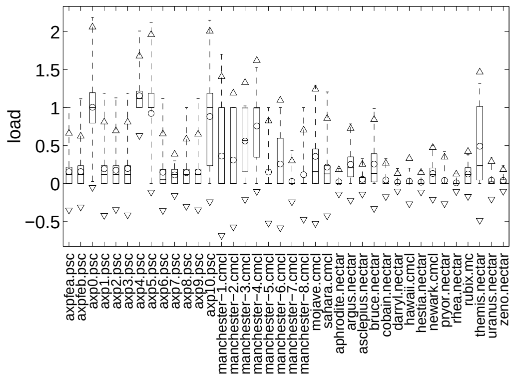


Figure 3. Statistical summary of load traces.

By subjecting various DUX systems to varying loads and sampling at progressively higher rates, we determined that DUX updates the load average value at a rate of 1/2 Hz. We chose to sample at 1 Hz in order to capture all of the load signal's dynamics made available by the kernel. We collected load traces on 38 hosts belonging to the Computing, Media, and Communication Laboratory (CMCL) at CMU, and to the Pittsburgh Supercomputing Center (PSC) for slightly more than one week in late August, 1997. A second set of week-long traces was acquired on almost exactly the same set of machines in late February through early March, 1998. The results of the statistical analysis were similar for the two sets of traces. In this paper, we describe and use the August 1997 set. A more detailed description of our analysis and the individual results for each trace is available elsewhere [9]. The traces themselves, as well as a tool for generating realistic workloads using them, are available from the web at http://www.cs.cmu.edu/∼pdinda/LoadTraces.

All of the hosts in the August 1997 set were DEC Alpha machines running DUX. Thirteen of the hosts are in the PSC's production cluster. Of these, there are two front-end machines, four interactive machines, and seven batch machines. In addition, traces were taken on eight hosts that comprise our lab's experimental cluster, two large memory machines used by our group as compute servers, and fifteen desktop workstations owned by members of our research group.

Figure 3 summarizes these traces in a Box plot variant. The central line in each box marks the median load, while the lower and upper lines mark the 25th and 75th percentiles. The lower and upper "whiskers" extending from the box mark the actual 2.5th and 97.5th percentiles. The circle marks the mean and the triangles mark the 2.5th and 97.5th percentiles assuming a normal distribution with the trace's mean and variance.

The following points summarize the results of our statistical analysis [9] that are relevant to this study:

(1) The traces exhibit low means but very high variability, measured by the variance, interquartile range, and maximum. Only four traces have mean loads near 1.0. The standard deviation (square root of the variance) is typically at least as large as the mean, while the maximums can be as much as two orders of magnitude larger. This high variability indicates that there exists ample opportunity for prediction algorithms to improve matters.

(2) Measures of variability, such as the variance and maximum, are positively correlated with the mean, so a machine with a high mean load will also tend to have a large variance and maximum. This correlation suggests that there is more opportunity for prediction algorithms on more heavily loaded machines.

(3) The traces have relatively complex, sometimes multimodal distributions that are not well fitted by common analytic distributions. However, we note here that assuming normality (but disallowing negative values) for the purpose of computing a 95% confidence interval is a reasonable operation.

(4) Time series analysis of the traces shows that load is strongly correlated over time. The autocorrelation function typically decays very slowly while the periodogram shows a broad, almost noise-like combination of all frequency components. An important implication is that linear models may be appropriate for predicting load signals. However, the complex frequency domain behavior suggests such models may have to be of unreasonably high order.

(5) The traces exhibit self-similarity with Hurst Parameters [2,4] ranging from 0.63 to 0.97, with a strong bias toward the top of that range. Hurst parameters in the range of 0.5 to 1.0 indicate self-similarity with positive near-neighbor correlations. This result tells us that load varies in complex ways on all time scales and has long-range dependence. Long-range dependence suggests that using the fractional ARIMA (ARFIMA) modeling approach [20,16,4] may be appropriate.

(6) The traces display what we term "epochal behavior." The local frequency content (measured by using a spectrogram) of the load signal remains quite stable

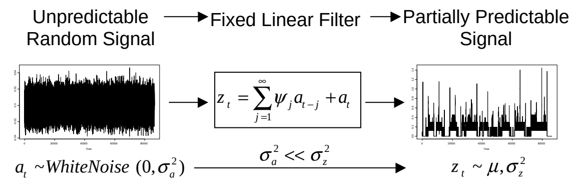


Figure 4. Linear time series models.

for long periods of time (150–450 seconds mean, with high standard deviations), but changes abruptly at the boundaries of such epochs. Such abrupt transitions are likely to be due to processes being created, destroyed, or entering new execution phases. This result implies that linear models may have to be refit at these boundaries. Our evaluation (Section 6) ignores these boundaries, so we do see the effect of inadvertently crossing one during prediction.

Strictly speaking, (6) means that load is not stationary. However, it is also not free to wander at will—clearly load cannot rise to infinite levels or fall below zero. This is not incompatible with the "borderline stationarity" implied by (5).

4. Linear time series models

The main idea behind using a linear time series model in load prediction is to treat the sequence of periodic samples of host load, $\langle z_t \rangle$, as a realization of a stochastic process that can be modeled as a white noise source driving a linear filter. The filter coefficients can be estimated from past observations of the sequence. If most of the variability of the sequence results from the action of the filter, we can use its coefficients to estimate future sequence values with low mean squared error.

Figure 4 illustrates this decomposition. In keeping with the relatively standard Box-Jenkins notation [6], we represent the input white noise sequence as $\langle a_t \rangle$ and the output load sequence as $\langle z_t \rangle$. On the right of Figure 4 we see our partially predictable sequence $\langle z_t \rangle$, which exhibits some mean μ and variance σ_z^2. On the left, we see our utterly unpredictable white noise sequence $\langle a_t \rangle$, which exhibits a zero mean and a variance σ_a^2. In the middle, we have our fixed linear filter with coefficients $\langle \psi_j \rangle$. Each output value z_t is the sum of the current noise input a_t and all previous noise inputs, weighted by the $\langle \psi_j \rangle$ coefficients.

Given an observed load sequence $\langle z_t \rangle$, the optimum values for the coefficients ψ_j are those that minimize σ_a^2, the variance of the driving white noise sequence $\langle a_t \rangle$. Notice that the one-step-ahead prediction given all the data up to and including time $t-1$ is $\hat{z}_{t-1}^1 = \sum_{j=1}^{\infty} \psi_j a_{t-j}$, since the expected value of $a_t = 0$.) The noise sequence consists simply of the one-step-ahead prediction errors and the optimal coefficient values minimize the sum of squares of these prediction errors.

The general form of the linear time series model is, of course, impractical, since it involves an infinite summation using an infinite number of completely independent weights. Practical linear time series models use a small number of coefficients to represent infinite summations with restrictions on the weights, as well as special casing the mean value of the sequence, μ. The statistical goal of these models is to capture the autocorrelation structure of the signal parsimoniously—in as few parameters as possible. To understand these models, it is easiest to represent the weighted summation as a ratio of polynomials in B, the backshift operator, where $B^d z_t = z_{t-d}$. For example, we can write $z_t = \sum_{j=1}^{\infty} \psi_j a_{t-j} + a_t$ as $z_t = \psi(B) a_t$ where $\psi(B) = 1 + \psi_1 B + \psi_2 B^2 + \ldots$ Using this scheme, the models we examine in this paper can be represented as

$$z_t = \frac{\theta(B)}{\phi(B)(1-B)^d} a_t + \mu \qquad (1)$$

where the different model classes we examine in this paper (AR, MA, ARMA, ARIMA, ARFIMA, BM (including LAST), and MEAN) constrain $\theta(B)$, $\phi(B)$ and d in different ways. In the signal processing domain, this kind of filter is known as a pole-zero filter. The roots of $\theta(B)$ are the zeros and the roots of $\phi(B)(1-B)^d$ are the poles. It is also a state-space filter, as can be seen if written as $z_t + \eta_1 z_{t-1} + \eta_2 z_{t-2} + \ldots + \eta_{p+d} z_{t-p-d} = a_t + \theta_1 a_{t-1} + \theta_2 a_{t-2} + \ldots + \theta_q z_{t-q}$ where $\eta(B) = \phi(B)(1-B)^d$ for integer d. In general, such a filter can be unstable in that its outputs can rapidly diverge from the input signal. This instability is extremely important from the point of view of the implementor of a load prediction system. Such a system will generally fit a model (choose the $\theta(B)$ and $\phi(B)$ coefficients, and d) using some m previous observations. The model will then be "fed" the next n observations and asked to make predictions in the process. If the coefficients are such that the filter is unstable, then it may explain the initial m observations very well, yet fail miserably and even diverge (and crash!) when used on the n observations after the fitting.

AR(p) models: The class of AR(p) models (purely autoregressive models) has $z_t = \frac{1}{\phi(B)}a_t + \mu$ where $\phi(B)$ has p coefficients. From the point of view of a system designer, AR(p) models are highly desirable since they can be fit to data in a deterministic amount of time. In the Yule-Walker technique that we used, the autocorrelation function is computed to a maximum lag of p and then a p-wide Toeplitz system of linear equations is solved. Even for relatively large values of p, this can be done almost instantaneously. Another advantage of AR(p) models is that they are remarkably stable.

MA(q) models: The class of MA(q) models (purely moving average models) has $z_t = \theta(B)a_t$ where $\theta(B)$ has q coefficients. MA(q) models are a much more difficult proposition for a system designer since fitting them takes a nondeterministic amount of time. Instead of a linear system, fitting a MA(q) model presents us with a quadratic system. Our implementation, which is nonparametric (ie, it assumes no specific distribution for the white noise source), uses the Powell minimization procedure [27, 406–413] to minimize the sum of squares of the $t+1$ prediction errors. The number of iterations necessary to converge is nondeterministic and data dependent.

ARMA(p,q) models: The class of ARMA(p,q) models (autoregressive moving average models) has $z_t = \frac{\theta(B)}{\phi(B)}a_t + \mu$ where $\phi(B)$ has p coefficients and $\theta(B)$ has q coefficients. By combining the AR(p) and MA(q) models, ARMA(p,q) models hope to achieve greater parsimony. From a system designer's point of view, this may be important, at least in so far as it may be possible to fit a more parsimonious model more quickly. Like MA(q) models, however, ARMA(p,q) models take a nondeterministic amount of time to fit to data, and we use the same Powell minimization procedure to fit them.

ARIMA(p,d,q) models: The class of ARIMA(p,d,q) models (autoregressive integrated moving average models) implement Equation 1 for positive integer d. Intuitively, the $(1-B)^d$ component amounts to a d-fold integration of the output of an ARMA(p,q) model. Although this makes the filter inherently unstable, it allows for modeling nonstationary sequences. Such sequences can vary over an infinite range and have no natural mean. Although load clearly cannot vary infinitely, it doesn't have a natural mean either. ARIMA(p,d,q) models are fit by differencing the sequence d times and fitting an ARMA(p,q) model as above to the result.

ARFIMA(p,d,q) models: The class of ARFIMA(p,d,q) models (autoregressive fractionally integrated moving average models) implement Equation 1 for fractional values of d, $0 < d < 0.5$. By analogy to ARIMA(p,d,q) models, ARFIMAs are fractionally integrated ARMA(p,q) models. The details of fractional integration [20,16] are not important here other than to note that $(1-B)^d$ for fractional d is an infinite sequence whose coefficients are functions of d. The idea is that this infinite sequence captures long range dependence while the ARMA coefficients capture short range dependence. Since our sequences exhibit long-range dependence, even after differencing, ARFIMAs may prove to be beneficial models. To fit ARFIMA models, we use Fraley's Fortran 77 code [15], which does maximum likelihood estimation of ARFIMA models following Haslett and Raftery [19]. This implementation is also used by commercial packages such as S-Plus. We truncate $(1-B)^d$ at 300 coefficients and use the same representation and prediction engine as with the other models.

Simple models for comparison: We also implemented two very simple models for comparison, MEAN and BM. MEAN has $z_t = \mu$, so all future values of the sequence are predicted to be the mean. This is the best predictor, in terms of minimum mean squared error, for a sequence which has no correlation over time and we also use it to measure the raw variance of the load signal. The BM model is an AR(p) model whose coefficients are all set to $1/p$. This simply predicts the next sequence value to be the average of the previous p values, a simple windowed mean. p is chosen to minimize mean squared error for $t+1$ predictions. It is important to note that the LAST model, which predicts that all future values will be the same as the last measured value, is simply a BM(1).

Making predictions: After fitting one of the above models, we construct a predictor from it. The predictor consists of the model converted to a uniform form, the predicted next sequence value, a queue that holds the last $p+d$ ($d = 300$ for ARFIMA models) sequence values, and a queue the holds the last q prediction errors. When the next sequence value becomes available, it is pushed onto the sequence queue, its corresponding predicted value's absolute error is pushed onto the error queue, and the model is evaluated ($O(p+d+q)$ operations) to produce a new predicted next sequence value. We refer to this as *stepping* the predictor. At any point, the predictor can be queried for the predicted next k values of the sequence, along with their expected mean squared errors ($O(k(p+d+q)$ operations.)

Computational requirements of models: In implementing an online host load prediction system, we not only care about the predictive performance of the different models described above, but also their computational costs. To assess these costs, we measured the system and user time required to (1) fit a model and create a predictor, and (2) step one measurement into the predictor and extract one set of 30-step-ahead predictions. The machine we used was a 500 MHz Alpha 21164-based DEC personal workstation. The time to fit a model is dependent on the length of the measurement sequence. We measured the costs to fit to 2000 samples, which is roughly the average size sequence of the evaluation presented in the later sections. The representative measurement sequence we used is three hours of the themis trace. A more detailed evaluation of these RPS-based models can be found elsewhere [12].

Figure 5 shows the results of the measurements. It contains six graphs, one for the (a) MEAN, LAST, and AR models, and one each for the remaining (b) BM, (c) MA, (d) ARMA, (e) ARIMA, and (f) ARFIMA models. For each model, we plot several different and interesting combinations of parameter values. For each combination, we plot two bars. The first bar (Fit/Init) plots the time to fit the model and produce a predictor, while the second bar (Step/Predict) plots the time to step that predictor. Each bar is the average of 30 trials, each of which consists of one Fit/Init step and a large number of Step/Predict steps. The y-axis on each plot is logarithmic. We replicate some of the bars from graph to graph to simplify comparing models across graphs, and we also draw horizontal lines at roughly 1 ms and 100 ms, which are the Fit/Init times of AR(16) and AR(512) models, respectively. 1 ms is also the Step/Predict time of an AR(512) predictor.

Of the more sophisticated models, the AR models, even with very high order, are uniquely inexpensive to fit, which would be an important benefit in an online host load prediction system. However, because the Step/Predict time always grows with the number of parameters in the model, a more parsimonious MA, ARMA, or ARIMA model may be more appropriate, even though it may take longer to fit. This is not the case with ARFIMA models, however, which always have very expensive predictors. This is because we multiply out the $(1-B)^d$ term to generate 300 coefficients in the predictor. It is not clear how to avoid this.

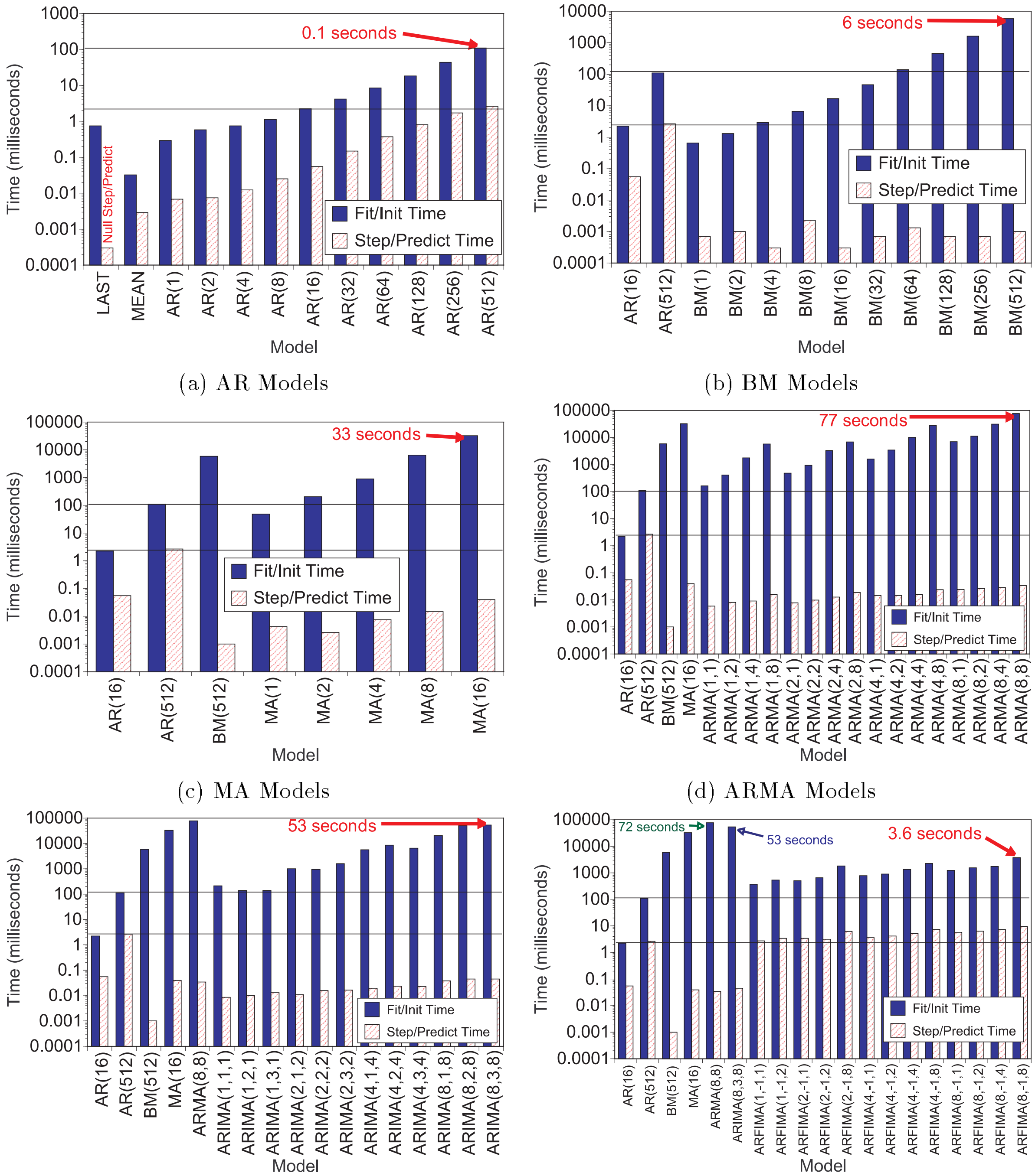
Although the ARFIMA models are considerable more complex than the MA, ARMA, and ARIMA models, Figure 5 shows, quite surprisingly, that they are cheaper to fit than those simpler models. This is because we use a highly-tuned maximum likelihood code that assumes a normal error distribution to fit the ARFIMA model. Using normality-assuming maximum likelihood based modelers for MA, ARMA, and ARIMA models would reduce their Fit/Init times to a bit below those of the ARFIMA models. We decided not to use these because the error distributions we saw were rarely normal.

5. Evaluation methodology

Our methodology is designed to determine whether there are consistent differences in the *practical predictive power* of the different model classes. Other goals are also possible. For example, one could determine the *explanatory power* of the models by evaluating how well they fit data, or the *generative power* of the model by generating new load traces from fitted models and evaluating how well their statistical properties match the original traces. We have touched on these other goals, but do not discuss them here. To assess the practical predictive power of the different model classes, we designed a randomized, trace-driven simulation methodology that fits randomly selected models to subsequences of a load trace and tests them on immediately following subsequences.

In practice, an online host load prediction system has one thread of control of the following form:

```
do forever {
   get new load measurement;
   update history;
   if (some criteria) {
      refit model to history;
      make new predictor;
```



(a) AR Models

(b) BM Models

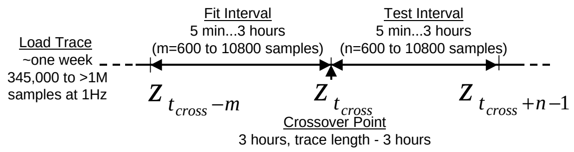
(c) MA Models

(d) ARMA Models

(e) ARIMA Models

(f) ARFIMA Models

Figure 5. Required CPU time for various prediction models, 2000 sample fits.



Model class	Number of parameters
MEAN	none
BM(p) (inc. LAST)	none
AR(p)	p=1..32
MA(q)	q=1..8
ARMA(p,q)	p=1..4, q=1..4
ARIMA(p,d,q)	p=1..4, d=1..2, q=1..4
ARFIMA(p,d,q)	p=1..4, d by fit, q=1..4

Figure 6. Testcase generation.

```
    }
    step predictor;
    sleep for sample interval;
}
```

where `history` is a window of previous load values. Fitting the model to the history is an expensive operation. Once a model is fitted, a predictor can be produced from it. *Stepping* this predictor means informing it of a new load measurement so that it can modify its internal state. This is an inexpensive operation. Prediction requests arrive asynchronously and are serviced using the current state of the predictor. A prediction request that arrives at time t includes a lead time k. The predicted load values $\hat{z}_t^1, \hat{z}_t^2, \ldots, \hat{z}_t^k$ are returned along with model-based estimates of the mean squared prediction error for each prediction.

Evaluating the predictive power of the different model classes in such a context is complicated because there is such a vast space of configuration parameters to explore. These parameters include: the trace, the model class, the number of parameters we allow the model and how they are distributed, the lead time, the length of the history to which the model is fit, the length of the interval during which the model is used to predict, and at what points the model is refit. We also want to avoid biases due to favoring particular regions of traces.

To explore this space in a reasonably unbiased way, we ran a large number of randomized testcases on each of the traces. Figure 6 illustrates the parameter space from which testcase parameters are chosen. A testcase is generated and evaluated using the following steps.

1. Choose a random *crossover point*, t_{cross}, from within the trace.
2. Choose a random number of samples, m, from $600, 601, \ldots, 10800$ (5 minutes to three hours.) The m samples preceding the crossover point, $z_{t_{cross}-m}, z_{t_{cross}-m+1}, \ldots, z_{t_{cross}-1}$ are in the *fit interval.*
3. Choose a random number of samples, n from $600, 601, \ldots, 10800$ (5 minutes to three hours.) The n samples including and following the crossover point, $z_{t_{cross}}, z_{t_{cross}+1}, \ldots, z_{t_{cross}+n-1}$, are in the *test interval.*
4. Choose a random AR, MA, ARMA, ARIMA, or ARFIMA *test model* from the table in Figure 6, fit it to the samples in the fit interval, and generate a predictor from the fitted test model.
5. For $i = m$ to 1, step the predictor with $z_{t_{cross}-i}$ (the values in the fit interval) to initialize its internal state. After this step, the predictor is ready to be tested.
6. For $i = 0$ to $n-1$ do the following:
   - Step the predictor with $z_{t_{cross}+i}$ (the next value in the test interval.)
   - For each lead time $k = 1, 2, \ldots, 30$ seconds, produce the predictions $\hat{z}^k_{t_{cross}+i}$. $\hat{z}^k_{t_{cross}+i}$ is the prediction of $z_{t_{cross}+i+k}$ given the samples $z_{t_{cross}-m}, z_{t_{cross}-m+1}, \ldots, z_{t_{cross}+i}$. Compute the prediction errors $a^k_{t+i} = \hat{z}^k_{t_{cross}+i} - z_{t_{cross}+i+k}$.
7. For each lead time $k = 1, 2, \ldots, 30$ seconds, analyze the k-step-ahead prediction errors a^k_{t+i}, $i = 0, 1, \ldots, n-1$.
8. Output the testcase parameters and the analysis of the prediction errors.

For clarity in the above, we focused on the linear time series model under test. Each testcase also includes a parallel evaluation of the BM and MEAN models in order to facilitate direct comparison with the simple BM model and the raw signal variance.

The lower limit we place on the length of the fit and test intervals is purely prosaic—the ARFIMA model needs about this much data to be successfully fit. The upper limit is chosen to be greater than most epoch lengths so that we can see the effect of crossing epoch boundaries. The models are limited to eight parameters because fitting larger MA, ARMA, ARIMA, or ARFIMA models is prohibitively expensive in a real system. We did also explore larger AR models, up to order 32.

The analysis of the prediction errors includes the following. For each lead time, the minimum, median, maximum, mean, mean absolute, and mean squared prediction errors are computed. The one-step-ahead prediction errors (ie, a^1_{t+i}, $i = 0, 1, \ldots, n-1$) are also subject to IID and normality tests as described by Brockwell and Davis [7, pp. 34–37]. IID tests included the fraction of the autocorrelations that are significant, the Portmanteau Q statistic (the power of the autocorrelation function), the turning point test, and the sign test. Normality was tested by computing the R^2 value of a least-squares fit to a quantile-quantile plot of the values or errors versus a sequence of normals of the same mean and variance.

We ran approximately 152,000 such testcases, which amounted to about 4000 testcases per trace, or about 1000 per model class and parameter set, or about 30 per trace, model class and parameter set. An additional 38,000 testcases were run to enable a fair comparison between the LAST and AR(16) models. Our parallelized simulation discarded testcases in which an unstable model "blew up," either detectably or due to a floating point exception. This amounted to fewer than 5% of the testcases, and was almost entirely confined to the ARIMA and ARFIMA models. The results of the accepted testcases were committed to a SQL database to simplify the analysis discussed in the following section.

6. Results

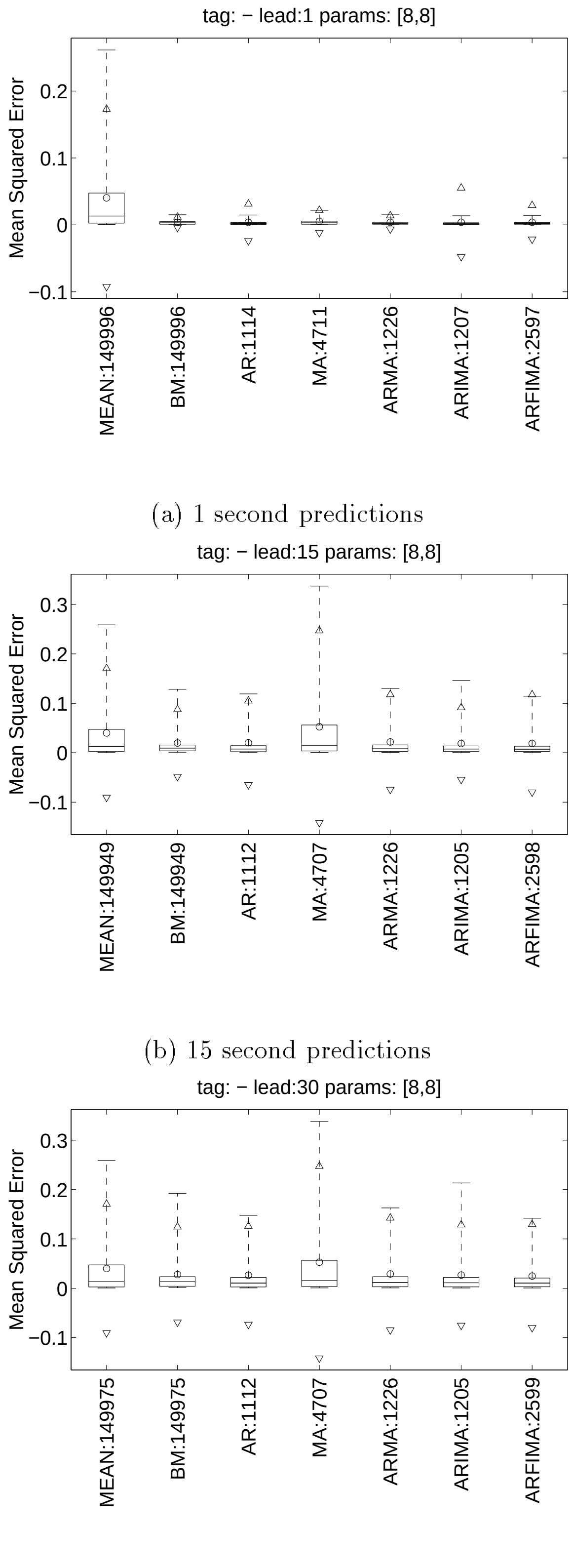
The section addresses the following questions: Is load consistently predictable? If so, what are the consistent differences between the different model classes, and which class is preferable? To answer these questions we analyze the database of randomized testcases from Section 5. For the most part we will address only the mean squared error results, although we will touch on the other results as well. It is important to reiterate the comments of Section 5: we are examining the practical predictive power of the models here, not the their explanatory power, or "fit."

Load is consistently predictable: For a model to provide consistent predictability of load, it must satisfy two requirements. First, for the average testcase, the model must have a considerably lower expected mean squared error than the expected raw variance of the load signal (ie, the expected mean squared error of the MEAN model.) The second requirement is that this expectation is also very likely, or that there is little variability from testcase to testcase. Intuitively, the first requirement says that the model provides good predictions on average, while the second says that most predictions are close to that average.

Figure 7(a) suggests that load is indeed consistently predictable in this sense. The figure is a Box plot that shows the distribution of one-step-ahead (one second) mean squared error measures (ie, the distribution of the measure $\frac{1}{n}\sum_{i=0}^{n-1}(a^1_{t+i})^2$, using the formalisms of Section 5) for 8 parameter models on all of the traces. In the figure, each category is a specific class of model and is annotated with the number of samples for that class. For each class, the circle indicates the expected mean squared error, while the triangles indicated the 2.5th and 97.5th percentiles assuming a normal distribution. The center line of each box shows the median while the lower and upper limits of the box show the 25th and 75th percentiles and the lower and upper whiskers show the actual 2.5th and 97.5th percentiles.

Notice that the expected raw variance (MEAN) of a testcase is approximately 0.05, while the expected mean squared error for *all* of the model classes is nearly zero. In terms of the length of the 95% confidence interval for the running time of a one second task as described in Section 2, this is a reduction from $(2)(1.96)\sqrt{0.05} = 0.87$ seconds to virtually zero for all of the classes of predictive models, including the simple BM model. The figure also shows that our second requirement for consistent predictability is met. We see that the variability around the expected mean squared error is much lower for the predictive models than for MEAN. For example, the 97.5th percentile of the raw variance is almost 0.3 (2.2 second interval) while it is about 0.02 (0.6 second interval) for the predictive models.

Figures 7(b) and 7(c) show the results for 15 second predictions and 30 second predictions. Notice that, except for the MA models, the predictive models are consistently better than the raw load variance, even with 30 second ahead predictions. We also see that MA mod-



(a) 1 second predictions

(b) 15 second predictions

(c) 30 second predictions

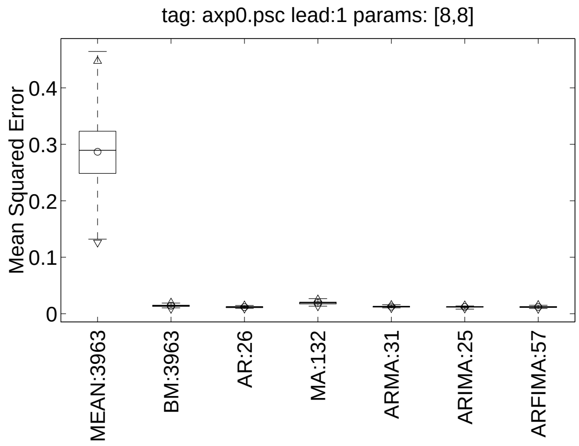
Figure 7. Distributions of mean squared errors for all traces using 8 parameter models.

els perform quite badly, especially at higher lead times. This was also the case when we considered the traces individually and broadened the number of parameters. MA models are clearly ineffective for load prediction.

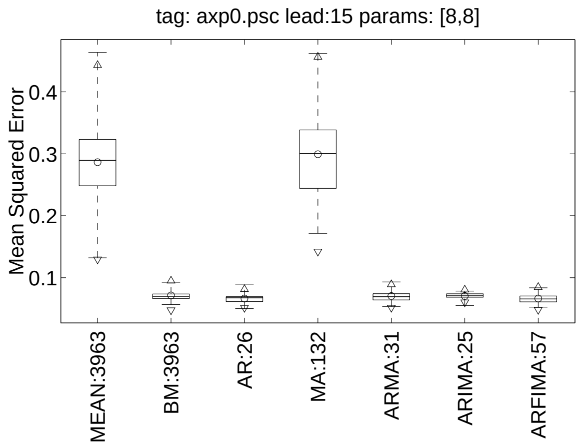
Successful models have similar expected performance overall, but differ on heavily loaded machines: Surprisingly, Figures 7(a)–(c) also show that the differences between the successful models, in aggregate over all the traces, are actually quite small. This is also the case if we expand to include testcases with 2 to 8 parameters instead of just 8 parameters. With longer lead times, the differences do slowly increase.

For more heavily loaded machines, the differences can be much more dramatic. For example, Figure 8(a) shows the distribution of one-step-ahead (one second) mean squared error measures for 8 parameter models on the axp0.psc trace. Here we see an expected raw variance (MEAN) of almost 0.3 (2.2 second confidence interval) reduced to about 0.02 (0.6 second interval) by nearly all of the models. Furthermore, the mean squared errors for the different model classes are tightly clustered around the expected 0.02 value, quite unlike with MEAN, where we can see a broad range of values and the 97.5th percentile is almost 0.5 (2.8 second interval.) The axp0.psc trace and others are also quite amenable to prediction with long lead times. For example, Figures 8(b) and (c) show 15 and 30 second ahead predictions for 8 parameter models on the axp0.psc trace, respectively. With the exception of the MA models, even 30 second ahead predictions are consistently much better than the raw signal variance. These figures remain essentially the same if we include testcases with 2 to 8 parameters instead of just 8 parameters. The use of more sophisticated models is more important on heavily loaded hosts than on lightly loaded hosts.

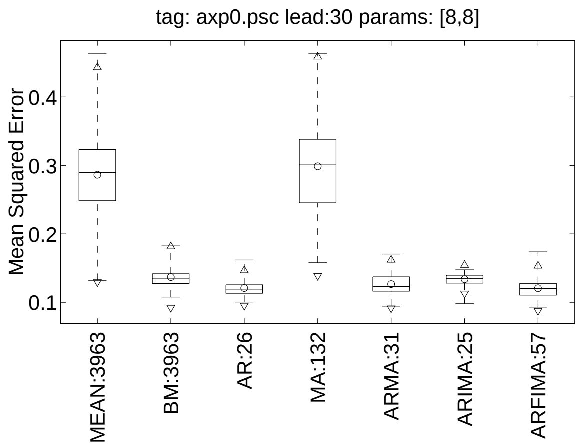
Although the differences in expected performance between the successful models are very small, they are generally statistically significant. We can algorithmically compare the expected mean squared error of the models using the unpaired t-test [21, pp. 209–212], and do ANOVA procedures to verify that the differences we detect are significantly above the noise floor. For each pair of model classes, the t-test tells us, with 95% confidence, whether the first model is better, the same, or worse than the second. We do the comparisons for the cross product of the models at a number of different lead times. We consider the traces both in aggregate and in-



(a) 1 second predictions



(b) 15 second predictions



(c) 30 second predictions

Figure 8. Distributions of mean squared errors for axp0.psc trace using 8 parameter models.

	MEAN	BM	AR	MA	ARMA	ARIMA	ARFIMA
MEAN	=	>	>	>	>	>	>
BM	<	=	=	<	=	=	=
AR	<	=	=	<	=	=	=
MA	<	>	>	=	>	=	>
ARMA	<	=	=	<	=	=	=
ARIMA	<	=	=	<	=	=	=
ARFIMA	<	=	=	<	=	=	=

(a) all traces, one step ahead

	MEAN	BM	AR	MA	ARMA	ARIMA	ARFIMA
MEAN	=	>	>	=	>	>	>
BM	<	=	>	<	=	=	>
AR	<	<	=	<	=	<	=
MA	=	>	>	=	>	>	>
ARMA	<	=	=	<	=	=	>
ARIMA	<	=	>	<	=	=	>
ARFIMA	<	<	=	<	<	<	=

(b) axp0.psc trace, 16 steps ahead

Figure 9. T-test comparisons: (a) all traces aggregated, lead 1, 8 parameters. Longer leads are essentially the same except MA is always worse. (b) axp0, lead 16, 8 parameters.

dividually, and we use several different constraints on the number of parameters.

Figure 9(a) shows the results of such a t-test comparison for the aggregated traces, a lead time of 1, and 8 parameters. In the figure, the row class is being compared to the column class. For example, at the intersection of the AR row and MA column, there is a '<', which indicates that, with 95% confidence, the expected mean squared error of the AR models is less than that of MA models, thus the AR models are better than MA models for this set of constraints. For longer lead times, we found that the results remained essentially the same, except that MA became consistently worse.

The message of Figure 9(a) is that, for the typical trace and a sufficient number of parameters, there are essentially no differences in the expected mean squared error of the BM, AR, ARMA, ARIMA, and ARFIMA models, even with high lead times. Further, with the exception of the MA models, all of the models are better than the raw variance of the signal (MEAN model.)

For machines with higher mean loads, there are more statistically significant differences between the models. For example, Figure 9(b) shows a t-test comparison for the axp0.psc trace at a lead time of 16 seconds for 8 parameter models. Clearly, there is more differentiation here, and we also see that the ARFIMA models do particularly well, as we might expect given the self-similarity result of Section 3. However, notice that the AR models are doing about as well.

The results of these t-test comparisons can be summarized as follows: (1) Except for the MA models, the models we tested are significantly better (in a statistical sense) than the raw variance of the trace and the difference in expected performance is also significant from a systems point of view. (2) The AR, ARMA, ARIMA, and ARFIMA models are significantly better than or equal to the BM model and occasionally the difference in expected performance is also significant from a systems point of view. This is especially the case on more heavily loaded hosts. (3) The AR, ARMA, ARIMA, and ARFIMA models have similar expected performance.

AR models are better than BM and LAST models: Since the AR, ARMA, ARIMA, and ARFIMA models have similar expected mean squared error when evaluated using the methods of the previous sections, the inclination is to prefer the AR models because of their much lower computational requirements. However, should we prefer AR models over BM models, or over their degenerate case, the LAST model? After all, these latter models also have low overhead and are much easier to understand and build. Also, the expected mean squared error of BM is close to that of AR, ignoring the more heavily loaded hosts.

To address this issue, we compared the *normalized* performance of the AR, BM, and LAST models. The metric we compared was the expected percentage reduction in variance—the average value of

$$\frac{testcase\,variance - mean\,squared\,error}{testcase\,variance} \cdot 100\%$$

over all the testcases. The primary advantage of this comparison is that very high variance testcases, which are almost always due to crossing an epoch boundary, do not disproportionately affect the expected percentage reduction in variance (as they do the expected absolute variance we measured in the previous section.) However, comparing two models by this metric only makes sense if their testcases are paired, resulting in matched denominators. Given our experimental setup, this is only possible between the MEAN, BM, and one other model.

Figure 10 uses this process to compare AR models of different orders to BM models. Each graph in the figure plots the average percentage reduction in variance, over all the testcases, as a function of the lead time for AR models of a given order and their corresponding BM models. The performance of the MEAN model is 0% in

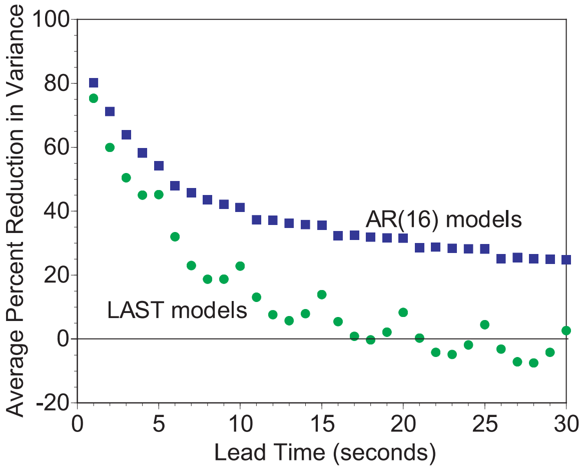


Figure 11. Paired normalized comparisons of AR(16) and LAST models by lead time.

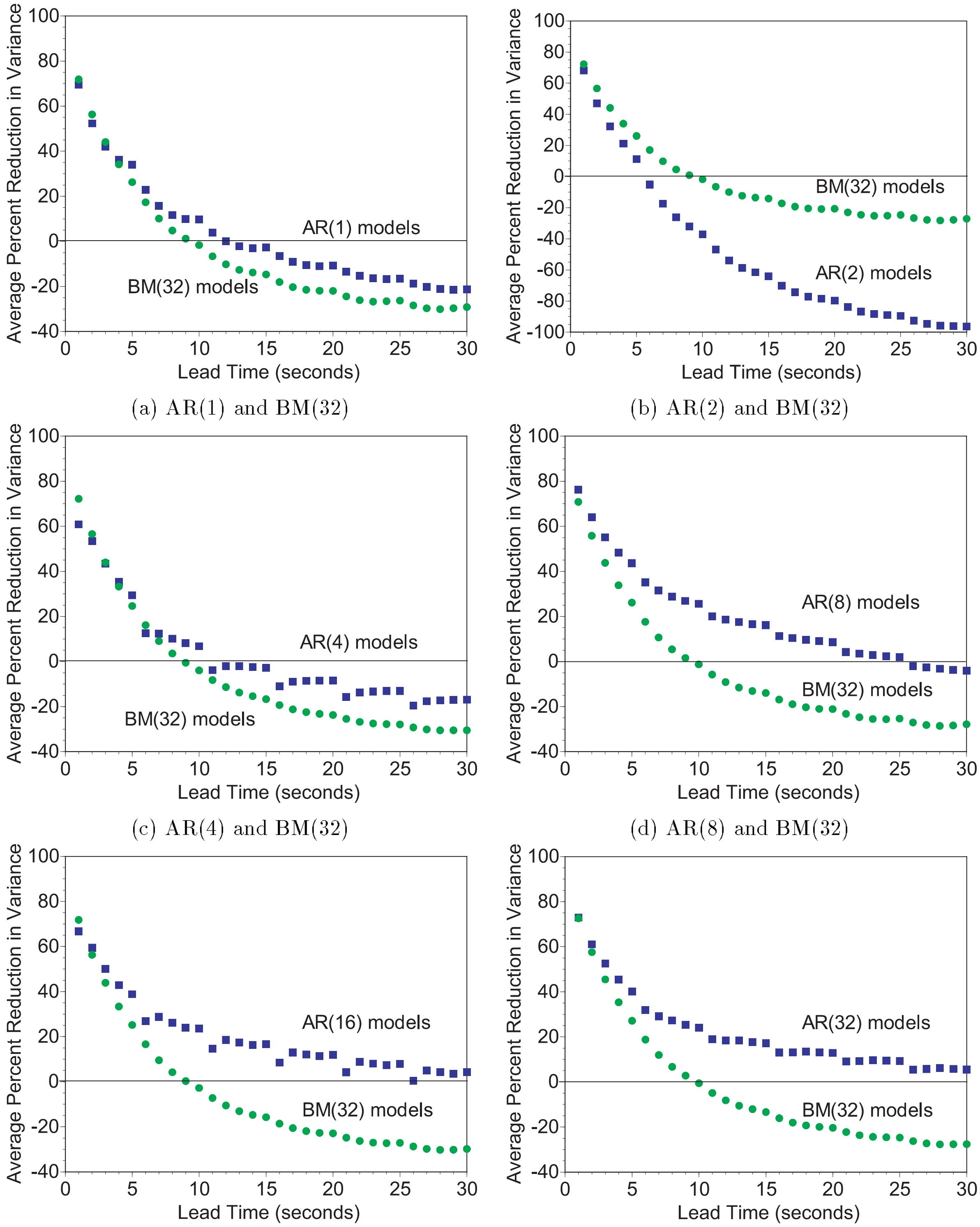
all cases. Each point on a graph encodes approximately 1000 testcases.

As the figure makes clear, AR models of sufficiently high order outperform BM models, especially at high lead times. Notice that for predictions as far ahead as 30 seconds, the AR(16) and AR(32) models provide lower variance than the raw load signal, while the BM models produce prediction errors that are actually more surprising than the signal itself. Indeed, the BM models only seem useful at lead times of 10 seconds or less. Where they work as well as the AR models is at extremely short prediction horizons.

Figure 10 also shows that the performance of the AR models generally increases as their order increases. There are diminishing returns with higher order models, however. Notice that the gain from the AR(16) model to the AR(32) model is marginal. For very low order models, rather odd effects can occur, such as with AR(2) models (Figure 10(b)), which buck the overall trend of improving on lower order models. We noticed that AR models of order less than 5 have highly variable behavior, while AR models of order greater than 16 don't significantly improve predictive performance.

As we discussed in Section 3, host load is produced by an exponential smoothing process in the kernel, which is an AR(1). If the only source of predictability in the host load signal was the exponential smoothing done in the kernel, we would expect that AR models of higher order would not provide better predictability. As we can see, however, higher order models do indeed result in lower mean squared prediction errors.

It is interesting to compare AR(16) models to the



(a) AR(1) and BM(32)

(b) AR(2) and BM(32)

(c) AR(4) and BM(32)

(d) AR(8) and BM(32)

(e) AR(16) and BM(32)

(f) AR(32) and BM(32)

Figure 10. Paired normalized comparisons of AR and BM models by lead time.

simplest possible predictor, the LAST model. We ran an additional set of testcases comparing AR(16) and LAST models to do this. It is possible, of course, to mine the testcases in the original set to do the comparison. However, the testcases we would look at would not be random, but rather those where the BM model had selected LAST as being most appropriate. By running a separate set of testcases, we were able to eliminate this bias.

Figure 11 compares the average percentage reduction in variance, computed as before, of the AR(16) model and the LAST model as a function of the lead time. As we can see, the AR(16) model significantly outperforms the LAST model, especially for lead times greater than five seconds. The AR(16) curve of Figure 11 is different from that of Figure 10(e) because of the use of a different random set of testcases.

AR(16) models or better are appropriate: Clearly, using one of the more sophisticated model classes, other than MA, for load prediction is beneficial. Further, using an AR, ARMA, ARIMA, or ARFIMA model is preferable to the BM model. In the paired comparisons, AR models of sufficiently high order considerably outperform BM and LAST models, especially at longer lead times. AR models of the appropriate order are much less expensive to fit than the ARMA, ARIMA, or ARFIMA models and are of similar or lesser cost to use. In addition, AR models can be fitted in a deterministic amount of time. Because of this combination of high performance and low overhead, AR models are the most appropriate for host load prediction. AR models of order 5 or higher seem to work fine. However, we found the knee in the performance of the AR models to be around order 16. Since fitting AR models of this order, or even considerably higher, is quite fast, we recommend using AR(16)s or better for host load prediction.

Prediction errors are not IID normal: As we described in Section 5, our evaluation of each testcase includes tests for the independence and normality of the one-step-ahead prediction errors. Intuitively, we want the errors to be independent so that they can be characterized with a probability distribution function, and we want the distribution function to be normal in order to simplify computing confidence intervals from it.

For the most part, the errors we see with the different models are not independent or normal to any high confidence level. However, from a practical point of view, the errors are much less correlated over time than the raw signal. For example, for AR(8) models, the Portmanteau Q statistic tends to be reduced by an order of magnitude or more, which suggests that those autocorrelations that are large enough to be significant are only marginally so. Furthermore, assuming normality for the purpose of computing confidence intervals for the running time of short tasks with no priority boosts at high confidence levels seems to be reasonable. If we need to predict the average host load over a longer interval of time, we will sum individual predictions. By the central limit theorem, the distribution of this sum should converge to normality, although it may do so slowly. However, to compute the variance of this sum, it is necessary to use the covariances we mentioned in Section 2.

7. Online host load prediction system

Using RPS, we implemented an online host load prediction system based on the AR(16) model and evaluated its performance. The system was constructed using RPS's prediction components, which are programs that can be composed at run-time using a variety of different communication mechanisms to form prediction systems for periodically sampled, scalar-valued signals. The system includes a component that monitors its prediction errors and mean squared error estimates and forces the model to be refit if these quantities exceed user-specified thresholds. This section summarizes the evaluation of the system, which is presented in more detail elsewhere [12]. RPS and the host load prediction system are publicly available from http://www.cs.cmu.edu/~pdinda/RPS.html.

The maximum rate at which the host load prediction system can operate on a 500 MHz Alpha 21164-based workstation running DUX 4.0D is roughly 730 Hz, which is almost three orders of magnitude faster than our desired rate of 1 Hz. At rates lower than 730 Hz, the mean and median latency from when a measurement is sampled to when its corresponding prediction is ready is 2 ms. The maximum latency we have observed is 33 ms. These measurements indicate that the system provides extremely timely host load predictions.

Figure 12 shows the percentage of the CPU that was in use over time, as measured by vmstat, as we swept

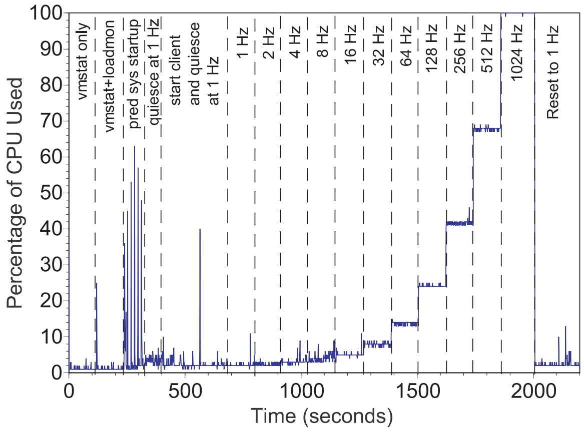


Figure 12. CPU utilization (system and user) of online host load prediction system. The measurement rate is swept from 1 Hz to 1024 Hz.

the measurement rate from 1 Hz to 1024 Hz. The important result is that at the appropriate 1 Hz rate, less than 5% of the CPU is being used. Indeed, considering the baseline CPU utilization measured when only vmstat was running, the overhead is in the noise. In terms of the network utilization, the system, when serving a single client, transfers 1796 bytes per host load measurement. At the appropriate 1 Hz measurement rate, the system clearly uses only miniscule amounts of CPU time and network bandwidth.

8. Related work

In application-level scheduling [5], applications schedule themselves, adapting to the availability of resources, perhaps using a framework such as QuO [32] or Dv [1]. Resource monitoring systems such as Remos [23], the Network Weather Service [30], or Topology-d [26] provide measurements and predictions to help applications make scheduling decisions. This paper characterized one such measurement (the Unix load average), studied the performance of linear models for predicting it, found an appropraite model, and showed that prediction using this model can be done with very low overhead.

While considerable effort has gone into characterizing workloads [14,22,18], the focus has been on load sharing systems [13], which schedule all of jobs in a distributed system, and not on application-level scheduling. An important assumption in load sharing and balancing systems is that current load is a predictor of future load. This work shows that that assumption is valid in a statistically rigorous manner, and also explores the predictive power of more sophisticated predictive models. Mutka and Livny [24] studied workstation availability as a binary function of load average and other measures. Our previous paper [9] and the summary in this paper are the first detailed study of the Unix load average we are aware of.

Although linear time series methods are widely used in other areas, including networking [3,17], little work has been done in using time series methods for host load prediction. Samadani and Kalthofen's work [28] is the closest to ours. They found that small ARIMA models were preferable to single-point predictors (eg, LAST) and Bayesian predictors for predicting load. Their empirical study concentrated on coarse grain prediction (one minute sampling interval) of four traces that measured load as the number of non-idle processes shown by the Unix "ps" command. In contrast, we studied finer grain (one second sampling interval) prediction of the DUX five-second load average on a much larger set of machines using higher order models as well as the ARFIMA class of models. Additionally, our study was randomized with respect to models instead of using the Box-Jenkins heuristic identification procedure. Finally, we reached a different conclusion for our regime, namely that AR models are sufficient.

The Network Weather Service (NWS) uses windowed mean, median, and AR models to predict various resource measures [30] including CPU availability [31]. Our study and the study described in the latter paper are quite complementary. The NWS study confirms our earlier self-similarity results, quantifies how well various measures of CPU availability, including the Unix one-minute load average, correlate with running time, proposes a new hybrid CPU availability measure, and studies the performance of NWS's predictive models for 10 second and five minute predictions of CPU availability measured at a 0.1 Hz rate. In contrast, we evaluated the performance of AR and more complex linear time series models (including one that captures the long-range dependence that self-similarity induces) applied to shorter range prediction (1 to 30 seconds) of a more dynamic load signal (the DUX five-second load average measured at 1 Hz.) In addition, our evaluation used a randomized methodology and a much larger set of traces to gauge the models' predictive power independent of any particular system. Interestingly, both studies reach the conclusion that relatively simple predictive models such as AR are adequate for host load prediction. The conference version of this paper [11] was contemporaneous with the NWS results.

9. Conclusions

We have presented a detailed evaluation of the performance of linear time series models for predicting the Unix five-second load average on a host machine, from 1 to 30 second into the future, using 1 Hz samples of its history. Predicting this load signal is interesting because it is directly related to the running time of compute-bound tasks.

We began by studying the statistical properties of week-long 1 Hz load traces collected on 38 different machines. This study suggested that Box-Jenkins (AR, MA, ARMA, and ARIMA) and ARFIMA models might be appropriate. The computational requirements of these classes of models span a wide range, making some more practical than others for incorporation into an online prediction system. We evaluated the predictive performance of the models, as well as that of a simple windowed-mean predictor by running 190,000 randomized testcases on the load traces and then analyzing the results.

The main contribution of our evaluation is to show, in a rigorous manner, that host load on real systems is predictable to a very useful degree from past behavior by using linear time series techniques. In addition, we discovered that, while there are statistically significant differences between the different classes of models we studied, the marginal benefits of the more complex models do not warrant their much higher run-time costs. We reached the conclusion that AR models of order 16 or higher are sufficient for predicting 1 Hz data up to 30 seconds in the future. These models work very well and are very inexpensive to fit to data and to use. We implemented and evaluated an online host load prediction system based on the AR(16) model. At the 1 Hz rate, the system uses miniscule amounts of CPU time and network bandwidth while providing extremely up-to-date predictions.

For longer tasks, and in the presence of priority boosts, transforming from host load predictions to confidence intervals for task running times is considerably more complex than described in this paper. Our current work addresses this problem and shows how to use the resulting confidence intervals to schedule real-time tasks [10].

References

[1] Aeschlimann, M., Dinda, P., Kallivokas, L., Lopez, J., Lowekamp, B., and O'Hallaron, D. Preliminary report on the design of a framework for distributed visualization. In *Proceedings of the International Conference on Parallel and Distributed Processing Techniques and Applications (PDPTA'99)* (Las Vegas, NV, June 1999), CSREA Press, pp. 1833–1839.

[2] Bassingthwaighte, J. B., Beard, D. A., Percival, D. B., and Raymond, G. M. Fractal structures and processes. In *Chaos and the Changing Nature of Science and Medicine: An Introduction* (April 1995), D. E. Herbert, Ed., no. 376 in AIP Conference Proceedings, American Institute of Physics, pp. 54–79.

[3] Basu, S., Mukherjee, A., and Klivansky, S. Time series models for internet traffic. Tech. Rep. GIT-CC-95-27, College of Computing, Georgia Institute of Technology, February 1995.

[4] Beran, J. Statistical methods for data with long-range dependence. *Statistical Science 7*, 4 (1992), 404–427.

[5] Berman, F., and Wolski, R. Scheduling from the perspective of the application. In *Proceedings of the Fifth IEEE Symposium on High Performance Distributed Computing HPDC96* (August 1996), pp. 100–111.

[6] Box, G. E. P., Jenkins, G. M., and Reinsel, G. *Time Series Analysis: Forecasting and Control*, 3rd ed. Prentice Hall, 1994.

[7] Brockwell, P. J., and Davis, R. A. *Introduction to Time Series and Forecasting.* Springer-Verlag, 1996.

[8] Dinda, P., Lowekamp, B., Kallivokas, L., and O'Hallaron, D. The case for prediction-based best-effort real-time systems. In *Proc. of the 7th International Workshop on Parallel and Distributed Real-Time Systems (WPDRTS 1999)*, vol. 1586 of *Lecture Notes in Computer Science.* Springer-Verlag, San Juan, PR, 1999, pp. 309–318. Extended version as CMU Technical Report CMU-CS-TR-98-174.

[9] Dinda, P. A. The statistical properties of host load. *Scientific Programming 7*, 3,4 (1999). A version of this paper is also available as CMU Technical Report CMU-CS-TR-98-175. A much earlier version appears in LCR '98 and as CMU-CS-TR-98-143.

[10] Dinda, P. A. *Resource Signal Prediction and Its Application to Real-time Scheduling Advisors.* PhD thesis, School of Computer Science, Carnegie Mellon University, 2000. To Appear.

[11] Dinda, P. A., and O'Hallaron, D. R. An evaluation of linear models for host load prediction. In *Proceedings of the 8th IEEE International Symposium on High Performance Distributed Computing (HPDC '99)* (August 1999), pp. 87–96. Extended version available as CMU Technical Report CMU-CS-TR-98-148.

[12] DINDA, P. A., AND O'HALLARON, D. R. An extensible toolkit for resource prediction in distributed systems. Tech. Rep. CMU-CS-99-138, School of Computer Science, Carnegie Mellon University, July 1999.

[13] EAGER, D. L., LAZOWSKA, E. D., AND ZAHORJAN, J. Adaptive load sharing in homogeneous distributed systems. *IEEE Transactions on Software Engineering SE-12*, 5 (May 1986), 662–675.

[14] EAGER, D. L., LAZOWSKA, E. D., AND ZAHORJAN, J. The limited performance benefits of migrating active processes for load sharing. In *SIGMETRICS '88* (May 1988), pp. 63–72.

[15] FRALEY, C. Fracdiff: Maximum likelihood estimation of the parameters of a fractionally differenced ARIMA(p,d,q) model. Computer Program, 1991. http://www.stat.cmu.edu/general/fracdiff.

[16] GRANGER, C. W. J., AND JOYEUX, R. An introduction to long-memory time series models and fractional differencing. *Journal of Time Series Analysis 1*, 1 (1980), 15–29.

[17] GROSCHWITZ, N. C., AND POLYZOS, G. C. A time series model of long-term NSFNET backbone traffic. In *Proceedings of the IEEE International Conference on Communications (ICC'94)* (May 1994), vol. 3, pp. 1400–4.

[18] HARCHOL-BALTER, M., AND DOWNEY, A. B. Exploiting process lifetime distributions for dynamic load balancing. In *Proceedings of ACM SIGMETRICS '96* (May 1996), pp. 13–24.

[19] HASLETT, J., AND RAFTERY, A. E. Space-time modelling with long-memory dependence: Assessing ireland's wind power resource. *Applied Statistics 38* (1989), 1–50.

[20] HOSKING, J. R. M. Fractional differencing. *Biometrika 68*, 1 (1981), 165–176.

[21] JAIN, R. *The Art of Computer Systems Performance Analysis*. John Wiley and Sons, Inc., 1991.

[22] LELAND, W. E., AND OTT, T. J. Load-balancing heuristics and process behavior. In *Proceedings of Performance and ACM SIGMETRICS* (1986), vol. 14, pp. 54–69.

[23] LOWEKAMP, B., MILLER, N., SUTHERLAND, D., GROSS, T., STEENKISTE, P., AND SUBHLOK, J. A resource monitoring system for network-aware applications. In *Proceedings of the 7th IEEE International Symposium on High Performance Distributed Computing (HPDC)* (July 1998), IEEE, pp. 189–196.

[24] MUTKA, M. W., AND LIVNY, M. The available capacity of a privately owned workstation environment. *Performance Evaluation 12*, 4 (July 1991), 269–284.

[25] OBJECT MANAGEMENT GROUP. The common object request broker: Architecture and specification. Tech. Rep. Version 2.0, Object Management Group, July 1995.

[26] OBRACZKA, K., AND GHEORGHIU, G. The performance of a service for network-aware applications. In *Proceedings of the ACM SIGMETRICS SPDT'98* (October 1997). (also available as USC CS Technical Report 97-660).

[27] PRESS, W. H., TEUKOLSKY, S. A., VETTERLING, W. T., AND FLANNERY, B. P. *Numerical Recipes in Fortran*. Cambridge University Press, 1986.

[28] SAMADANI, M., AND KALTHOFEN, E. On distributed scheduling using load prediction from past information. Abstracts published in Proceedings of the 14th annual ACM Symposium on the Principles of Distributed Computing (PODC'95, pp. 261) and in the Third Workshop on Languages, Compilers and Run-time Systems for Scalable Computers (LCR'95, pp. 317–320), 1996.

[29] SUN MICROSYSTEMS, INC. Java remote method invocation specification, 1997. Available via http://java.sun.com.

[30] WOLSKI, R. Forecasting network performance to support dynamic scheduling using the network weather service. In *Proceedings of the 6th High-Performance Distributed Computing Conference (HPDC97)* (August 1997), pp. 316–325. extended version available as UCSD Technical Report TR-CS96-494.

[31] WOLSKI, R., SPRING, N., AND HAYES, J. Predicting the CPU availability of time-shared unix systems. In *Proceedings of the Eighth IEEE Symposium on High Performance Distributed Computing HPDC99* (August 1999), IEEE, pp. 105–112. Earlier version available as UCSD Technical Report Number CS98-602.

[32] ZINKY, J. A., BAKKEN, D. E., AND SCHANTZ, R. E. Architectural support for quality of service for CORBA objects. *Theory and Practice of Object Systems 3*, 1 (April 1997), 55–73.

Biographies

Peter Dinda is finishing his Ph.D. in the School of Computer Science at Carnegie Mellon University, where he is advised by David O'Hallaron. His interests include resource signal prediction, adaptive applications, performance analysis, distributed real-time systems, and high-performance distributed computing.

David O'Hallaron is an Associate Professor of Computer Science and Electrical and Computer Engineering at Carnegie Mellon University. In 1989 he joined the faculty at Carnegie Mellon, where he works on high-performance distributed computing, Internet services, and parallel scientific applications.