An Evaluation of Linear Models for Host Load Prediction

Peter A. Dinda David R. O'Hallaron
Carnegie Mellon University
5000 Forbes Avenue
Pittsburgh, PA 15213

Abstract

This paper evaluates linear models for predicting the Unix one-minute load average on a host machine from 1 to 30 seconds into the future using 1 Hz samples of its history. A detailed statistical study of a large number of load traces leads to consideration of the Box-Jenkins models (AR, MA, ARMA, ARIMA), and the ARFIMA models (due to self-similarity.) These models, as well as a simple windowed-mean scheme, are evaluated by running a large number of randomized testcases on the load traces and then data-mining the results. The main conclusions are that load is consistently predictable to a very useful degree and that the simpler models are sufficient for doing this prediction. Our recommendation is to use AR models of order 16 or better.

1 Introduction

Consider a user-level application which is trying to schedule a compute-bound soft real-time task in a typical distributed computing environment. By using mechanisms such as CORBA [24] or Java RMI [27], the task can be executed on any of the host machines in the network. Using 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 the application is trying to schedule. The computational load on each of the hosts can vary drastically over time.

Because of this dynamicity, 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 host is to meet the deadline.

The running time of a compute-bound task on a host is strongly related to the computational load it encounters during execution on that host. Conceptually, if we could predict the load signal a task would encounter on some

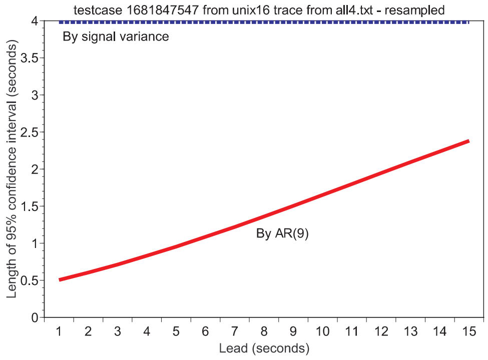


Figure 1: Benefits of prediction: server machine with 40 users and long term load average of 10.

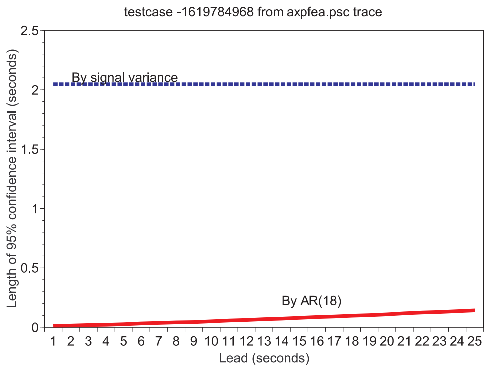


Figure 2: Benefits of prediction: interactive cluster machine with long term load average of 0.17.

host, we could predict the running time the task would have on that host. Better load predictions lead to better running time predictions and thus to smaller confidence intervals. For this reason, we concentrate on host load prediction here.

Better load predictions can lead to drastically smaller confidence intervals on real hosts, both on those with very high overall load, such as shared servers, and those with very low overall load, such as desktop workstations. Figure 1 shows the benefits of good predictions on a heavily loaded server. The figure plots the length of the confidence interval for the running time of a one second task as a function of for how far ahead load predictions are made. The confidence intervals for a predictive AR(9) model (described in Section 4) and for the raw variance of the load signal itself are represented. Notice that for short-term predictions, the AR(9) model provides confidence intervals that are almost an order of magnitude smaller than the raw signal. For example, when predicting 5 seconds ahead, the confidence interval for the AR(9) model is < 1 second while the confidence interval for the raw load signal is about 4 seconds. Figure 2 shows that such benefits are also possible on a very lightly loaded machine.

Clear benefits of this kind strongly motivated this rigorous study of load and load prediction. We wanted to answer the following questions: Is load consistently predictable? If so, what classes of predictive models are appropriate? What are the differences between these different classes? Finally, what class can be recommended for use in a real system?

The study found that 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 findings are surprising because load has complex behavior and exhibits properties such as self-similarity and epochal behavior that suggest that more complex models are appropriate. As far as we are aware, this is the first study to identify these properties of host load and then evaluate the practical predictive power of linear time series models that both can and cannot capture them. It 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 running on distributed systems.

Our measure of host load is the Unix one-minute average, which is closely related to the execution time of compute-bound tasks (Section 2.) We collected a large number of 1 Hz load traces and subjected them to a detailed statistical analysis, which we summarize in Section 3. The 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. However, 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 expensive ARFIMA model class, which explicitly captures long-range dependence, might be more appropriate.

Given these results, we evaluated the predictive power of the AR, MA, ARMA, ARIMA, and ARFIMA model classes, as well as a simple ad hoc windowed-mean predictor called BM and a long-term mean predictor called MEAN. We describe these model classes and our implementations of them in Section 4. The same codebase is used in a proof of concept load prediction system condition object for BBN's QuO distributed object quality of service system [31] and is also being used to implement load prediction for the Remos resource measurement system [22].

Our evaluation methodology, which we describe in detail in Section 5, was essentially to run randomized test-cases on the load traces we had collected and data-mine the results. The testcases (152,000 in all, or about 4000 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 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 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.

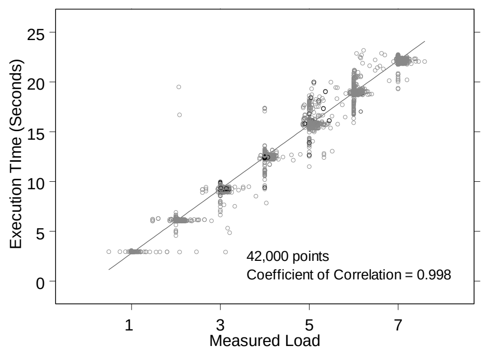


Figure 3: Relationship between average load during execution and execution time

The results of our evaluation are presented in Section 6. We found that load is consistently predictable to a useful degree from past behavior. Except for the MA models, which performed quite badly, the predictive models were all roughly similar, although statistically significant differences were indeed found. These marginal differences do not seem to warrant the use of the more sophisticated model classes because their run-time costs are much higher. Our recommendation is to use AR models of relatively high order (16 or better) for load prediction within the regime we studied.

Section 7 puts our study in the context of other work in the area, and Section 8 summarizes and concludes with a discussion of future directions of this research.

2 Host load and running time

For cpu-bound tasks, the relationship between host load and execution time is very intuitive. Consider Figure 3, which plots execution 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, at identical priority levels, on an otherwise unloaded Digital Unix machine. Each of these tasks sampled the Unix one-minute load average at roughly one second intervals during their execution and at termination printed the average of these samples as well as their execution 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 execution time is almost perfectly linear ($R^2 > 0.99$).

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

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

where t_{now} is when the task begins executing, $t_{nominal}$ is the execution time of the task on a completely unloaded machine, $(1 + z(t))$ is the load experienced during execution ($z(t)$ is the continuous "background" load), and t_{exec} is the task's execution time. Notice that the integral simply computes the inverse of the average load during execution. In practice, we can only sample the load with some noninfinitesimal 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 $\langle z_t \rangle = \ldots, z_{t-1}, z_t, z_{t+1}, \ldots$

If we knew z_{t+k} for $k > 0$ we could directly compute the execution 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 execution 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 (an probably higher) mean squared error than one-step-ahead predictions ($k = 1$), and so on.

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 execution time. The hope is that sophisticated prediction schemes will have much better mean squared errors.

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 is the length of the ready queue maintained by the scheduler. The kernel samples the length of the ready queue at some rate and exponentially averages some number of previous samples to produce a load average which can be accessed from a user program. We wrote a small tool to sample this value on Digital Unix (DUX) machines. By subjecting these 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. We ran this trace collection tool on 38 hosts belonging to the Computing, Media, and Communication Laboratory (CMCL) at CMU and 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 can be found in [8, 9] and the traces themselves can be found on the web at http://www.cs.cmu.edu/ pdinda/LoadTraces.

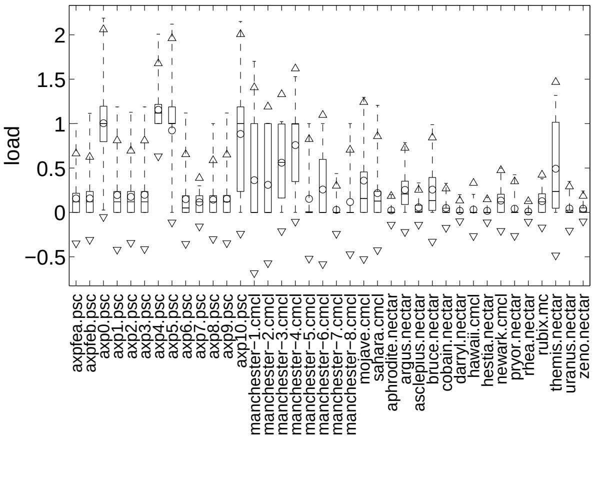


Figure 4: Statistical summary of load traces.

All of the hosts in the August 1997 set were DEC Alpha DUX machines, and they form four classes: *Production Cluster*, 13 hosts of the PSC's "Supercluster", including two front-end machines (axpfea, axpfeb), four interactive machines (axp0 through axp3), and seven batch machines); *Research Cluster*, eight machines in an experimental cluster in our lab (manchester-1 through manchester-8)); *Compute servers*, two high performance large memory machines used by our group as compute servers for simulations and the like (mojave and sahara); and *Desktops*, 15 desktop workstations owned by members of our research group (aphrodite through zeno).

Figure 4 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 of these traces which 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 had 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

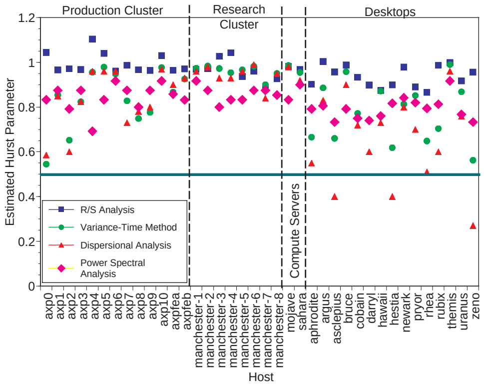


Figure 5: Hurst parameter estimates.

can be as much as two orders of magnitude larger. This high variability suggests that there is considerable opportunity for prediction algorithms.

(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 not unreasonable, as can be seen by comparing the 2.5th and 97.5th percentile whiskers and the corresponding normal percentiles (triangles) computed from the mean and variance in Figure 4.

(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. We estimated their Hurst parameters [19] using four different nonparametric methods [28] which provided the results in Figure 5. The traces' average Hurst parameter estimates ranged 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 suggests that using the fractional ARIMA (ARFIMA) modeling approach [18, 13, 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 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, or that threshold models [29] or other nonlinear or chaotic dynamics methods may be appropriate. We do not investigate these methods in this paper.

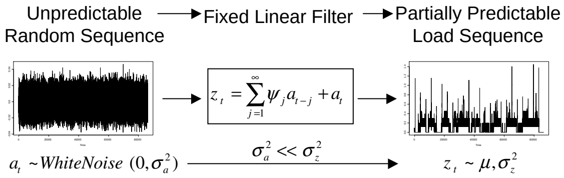


Figure 6: Linear time series models for load prediction.

4 Linear time series models

The main idea behind using a linear time series model in load prediction is to treat our sequence of periodic samples of host load, $\langle z_t \rangle$, as a realization of a stochastic process which 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 6 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 a one-step-ahead prediction for a given set of coefficients is $\hat{z}_{t+1} = \sum_{j=0}^{\infty} \psi_j a_{t-j}$ (since the expected value of $a_{t+1} = 0$), thus the noise sequence is simply the one-step-ahead ("$t+1$" or predictions for a lead time of one) prediction errors, the optimal coefficient values minimize the sum of squares of the one-step-ahead prediction errors.

The general form of the linear time series model as, 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. 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 $\psi(B) = 1 + \psi_1 B + \psi_2 B + \ldots$ 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, 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. 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 n previous observations. The model will then be "fed" the next m observations and asked to make predictions in the process. If coefficients are such that the filter is unstable, then it may explain the n observations very well, yet fail miserably and even diverge (and crash!) when used on the m observations after the fitting.

AR(p) models

The class of AR(p) models (purely autoregressive models) have $z_t = \frac{1}{\phi(B)} a_t + \mu$ where $\phi(B)$ has p coefficients. Intuitively, the output value is a ϕ-weighted sum of the p previous output values. The $t+1$ prediction for a load sequence is the ϕ-weighted sum of the p previous load measurements. The $t+2$ prediction is the ϕ-weighted sum of the $t+1$ prediction and the $p-1$ previous load measurements, and so on.

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.

MA(q) models

The class of MA(q) models (purely moving average models) have $z_t = \theta(B) a_t$ where $\theta(B)$ has q coefficients. Intuitively, the output value is the θ-weighted sum of the current and the q previous input values. The $t+1$ prediction of a load sequence is the θ-weighted sum of the q previous $t+1$ prediction errors. The $t+2$ prediction is the θ-weighted sum of the predicted $t+1$ prediction error (zero) and the $q-1$ previous $t+1$ prediction errors, and so on.

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 procedure 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) have $z_t = \frac{\theta(B)}{\phi(B)} a_t + \mu$ where $\phi(B)$ has p coefficients and $\theta(B)$ has q coefficients. Intuitively, the output value is the ϕ-weighted sum of the p previous output values plus the θ-weighted sum of the current and q previous input values. The $t+1$ prediction for a load sequence is the ϕ-weighted sum of the p previous load measurements plus the θ-weighted sum of the q previous $t+1$ prediction errors. The $t+2$ prediction is the ϕ-weighted sum of the $t+1$ prediction and the $p-1$ previous measurements plus the θ-weighted sum of the predicted $t+1$ prediction error (zero) and the $q-1$ previous prediction errors, and so on.

By combining the AR(p) and MA(q) models, ARMA(p,q) models hope to achieve greater parsimony — using fewer coefficients to explain the same sequence. 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 $d = 1, 2, \ldots$ 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 modelling 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 [18, 13] 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 [12], which does maximum likelihood estimation of ARFIMA models following Haslett and Raftery [17]. 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

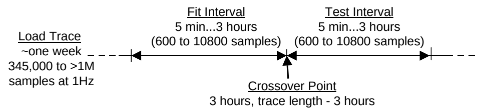
In addition to the Box-Jenkins models and the ARFIMA model described above, we also use two very simple models for comparison, MEAN and BM. MEAN has $z_t = \mu$, so all future values of the sequence are predicted to be the mean. This is the best predictor, in terms of minimum mean squared error, for a sequence which has no correlation over time. It is useful for measuring the raw variance in the load signal and the benefits accrued from exploiting the correlations that exist. 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.

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, it's 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)).

5 Evaluation methodology

Our methodology is designed to determine whether there are consistent differences in *practical predictive power* between 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



Model class	Number of parameters
MEAN	none
BM	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 7: Testcase generation.

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, a load prediction daemon will have 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;
   }
   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 decided to run a large number of randomized testcases on each of the traces. Figure 7 summarizes how a testcase is generated. First, a random crossover point is chosen in the trace. Next, a fit interval with a randomly selected length of five minutes (600 samples) to three hours (10800 samples) that ends at the crossover point is chosen. Then, a test interval is chosen that begins at the crossover point and has a random length from the same range. Finally, a test model (class and number of parameters) is randomly chosen from the table in Figure 7. 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 random testcase is run in the following way: (1) The test model, the BM model, and the MEAN model are fit to the measurements in the fit interval (the "history.") (2)Predictors are produced from the three fitted models and the data in the fit interval is stepped into the predictors to bring them to the crossover point. (3) For each subsequent value in the test interval each of the predictors is stepped with the value, and then asked to generate predictions for $1, 2, 3, \ldots, 30$ seconds into the future. The value and the predictions it produced are saved. (4) The values and predictions produced in the previous step are passed to an evaluation module which, for the actual values and predictions at each lead, computes the minimum, median, maximum, mean, mean absolute, and mean squared prediction errors for each lead time. The values and one-step-ahead prediction errors 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. Our parallelized simulation discarded testcases in which an unstable model "blew up," either detectably or due to a floating point exception. The results of the accepted testcases were committed to a SQL database to simplify the analysis discussed in the following section.

6 Results

The main purpose of this section is to answer the questions we posed in the introduction: is load consistently predictable and, if so, what are the consistent differences between the different model classes, and which class is preferable? To answer these questions we data-mined the database of randomized testcases that we describe in Section 5. We for the most part will address only the mean squared error results, although we will touch on the other results.

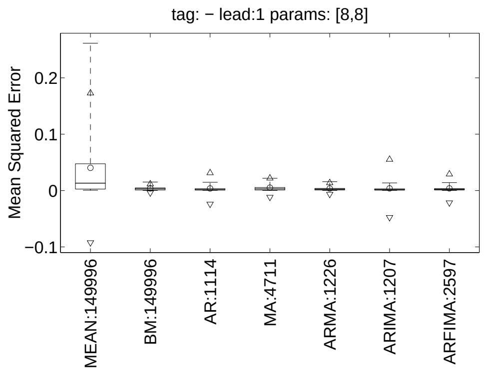


Figure 8: All traces, 1 second lead, 8 parameters

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 better predictions on average, while the second says that most predictions are close to that average.

Figure 8 shows that load is indeed consistently predictable in this sense. The figure is a Box plot that shows the distribution of one-step-ahead mean squared error measures 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 execution 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 excruciatingly simple BM model. The figure also shows that our second requirement for consistent predictability is met. We see that the variabil-

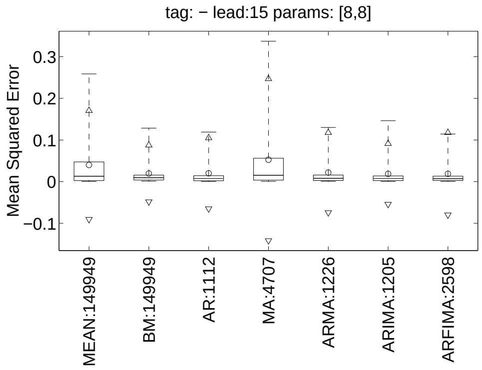


Figure 9: All traces, 15 second lead, 8 parameters

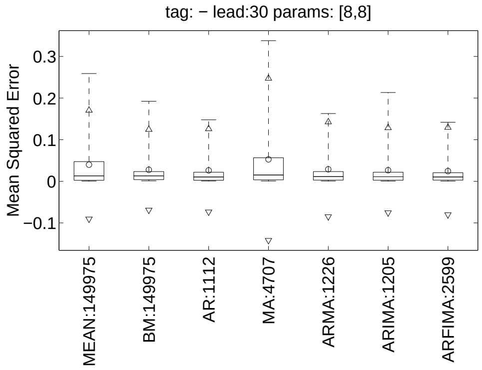


Figure 10: All traces, 30 second lead, 8 parameters

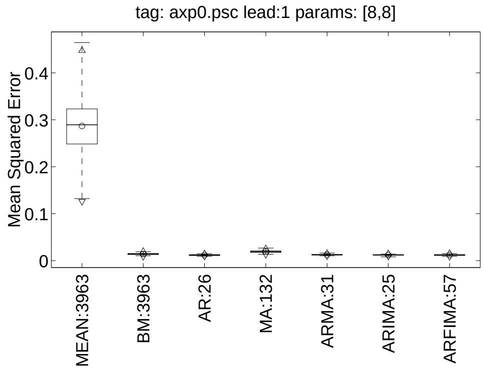


Figure 11: Axp0, 1 second lead, 8 parameters

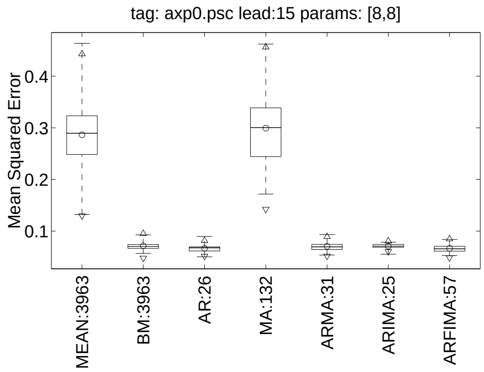


Figure 12: Axp0, 15 second lead, 8 parameters

ity 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 9 and 10 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 models 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 performance

Surprisingly, Figures 8 – 10 also show that the differences between the successful models 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 11 shows the distribution of one-step-ahead 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 12 and 13 show 15 and 30 second ahead predictions for 8 parameter models on the axp0.psc trace. 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.

Although the differences in performance between the successful models are very small, they are generally statistically significant. We algorithmically compared the expected mean squared error of the models using the un-

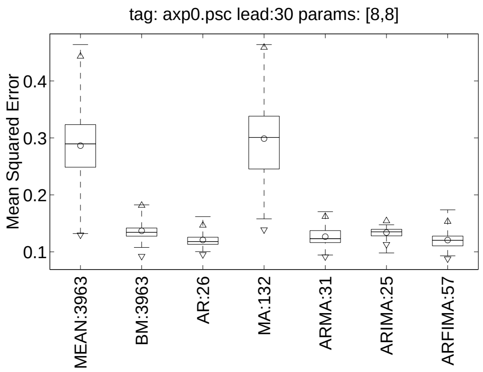


Figure 13: Axp0, 30 second lead, 8 parameters

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

Figure 14: T-test comparisons - all traces aggregated, lead 1, 8 parameters. Longer leads are essentially the same except MA is always worse.

paired t-test [20], pp. 209–212, and did ANOVA procedures to verify that the differences we detected were significantly above the noise floor, which they were. 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 did such comparisons for the cross product of the models at a number of different lead times. We considered the traces both in aggregate and individually, and used several different constraints on the number of parameters.

Figure 14 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 14 is that, for the typical trace and a sufficient number of parameters, there is essentially no difference between the predictive powers of the BM, AR, ARMA, ARIMA, and ARFIMA models, even with high lead times. Further, it shows that, 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 15 shows a t-test comparison for the

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

Figure 15: T-test comparisons - axp0, lead 16, 8 parameters.

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 were significantly better than the raw variance of the trace and the difference in expected performance was significant from a systems point of view. (2) the AR, ARMA, ARIMA, and ARFIMA models were significantly better than or equal to the BM model and occasionally the difference in expected performance was also significant from a systems point of view. (3) The AR, ARMA, ARIMA, and ARFIMA models had similar expected performance.

AR models are appropriate

Clearly, using one of the model classes, other than MA, for load prediction is beneficial. Further, using an AR, ARMA, ARIMA, or ARFIMA model is occasionally preferable to the BM model. Since there were no major differences in the performance of these models, and fitting AR models is much less expensive than the other models (and also takes deterministic amount of time), we recommend the use of AR models. AR models of order 4 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.

Prediction errors are not IID normal

As we described in Section 5, our evaluation of each testcase included tests for the independence and normality of prediction errors. Intuitively, we want the errors to be independent so that they can be characterized with 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 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 computing confidence intervals for high confidence levels, such as 95%, seems to be reasonable. However, since a load prediction system will probably continuously evaluate a fitted model in order to determine when to refit it, it seems reasonable for it to keep a histogram or other more detailed representation of the errors in order to more accurately compute confidence intervals.

7 Related work

In application-centric scheduling [5], applications schedule themselves, adapting to the availability of resources, perhaps using a framework such as QuO [31]. Resource monitoring systems such as Remos [22], the Network Weather Service [30], or Topology-d [25] provide measurements and predictions to help applications make scheduling decisions. This paper characterized one such measurement (the Unix load average) and studied how to predict it.

While considerable effort has gone into characterizing workloads [11, 21, 16], the focus has been on load sharing systems [10], which schedule all of jobs in a distributed system, and not on application-centric scheduling. Mutka and Livny [23] studied workstation availability as a binary function of load average and other measures. Our previous paper [8] 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, 14], little work has been done in using time series methods for host load prediction. Samadani and Kalthofen's work [26] is the closest to ours. They found that small ARIMA models were preferable to single-point predictors 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 Unix one minute 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 [30] uses windowed mean, median, and AR filters to predict various measures of CPU load, including the load average, but no detailed study of its load predictors is available. However, our result that simple prediction algorithms are adequate agrees with their initial results. [1] Hailperin [15] used the statistical periodicity of his sensor-driven workload to predict remote load in his load balancing system.

The AR, MA, ARMA, and ARIMA models have long histories dating back to the 19th century. Box, et al [6] is the standard reference and Brockwell and Davis[7] provide a useful example-driven approach. ARFIMA models are a relatively recent innovation [18, 13]. Bassingthwaighte, et al [2] provide a good intuitive introduction to self-similarity and Beran [4] provides a good introduction to long-range dependence.

[1] Richard Wolski, private communication.

8 Conclusions and future work

We presented a detailed evaluation of the performance of linear time series models for predicting the Unix one-minute 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 the running time of compute-bound tasks can be directly related to it. 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. We evaluated the performance of these classes of models and a simple windowed-mean predictor by running 152,000 randomized testcases on the load traces and then data-mining the results.

The main contribution of our evaluation is 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.

However, it is clear that host load signals are not generated by linear systems. Other techniques, such as threshold autoregressive models [29] or Abarbanel's methodology for chaotic data [1] may be more appropriate. We have applied parts of Abarbanel's approach to our traces with promising results. However, it may be more appropriate for finer or coarser grain load prediction — we are satisfied with simple linear methods for the regime we studied in this paper. We are currently studying bandwidth and latency prediction in local area networks using the evaluation framework described in this paper.

References

[1] ABARBANEL, H. *Analysis of Observed Chaotic Data*. Institute for Nonlinear Science. Springer, 1996.

[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. A. The statistical properties of host load. In *Proc. of 4th Workshop on Languages, Compilers, and Run-time Systems for Scalable Computers (LCR'98)* (Pittsburgh, PA, 1998), vol. 1511 of *Lecture Notes in Computer Science*, Springer-Verlag, pp. 319–334. Extended version available as CMU Technical Report CMU-CS-TR-98-143.

[9] DINDA, P. A., AND O'HALLARON, D. R. The statistical properties of host load. Tech. Rep. CMU-CS-TR-98-143, School of Computer Science, Carnegie Mellon University, July 1998.

[10] 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.

[11] 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.

[12] 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.

[13] 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.

[14] 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.

[15] HAILPERIN, M. *Load Balancing using Time Series Analysis For Soft Real-Time Systems with Statistically Periodic Loads*. PhD thesis, Stanford University, December 1993.

[16] 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.

[17] 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.

[18] HOSKING, J. R. M. Fractional differencing. *Biometrika 68*, 1 (1981), 165–76.

[19] HURST, H. E. Long-term storage capacity of reservoirs. *Transactions of the American Society of Civil Engineers 116* (1951), 770–808.

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

[21] 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.

[22] 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.

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

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

[25] 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).

[26] 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.

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

[28] TAQQU, M. S., TEVEROVSKY, V., AND WILLINGER, W. Estimators for long-range dependence: An empirical study. *Fractals 3*, 4 (1995), 785–798.

[29] TONG, H. *Threshold Models in Non-linear Time Series Analysis*. No. 21 in Lecture Notes in Statistics. Springer-Verlag, 1983.

[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] 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).