Forecasting Network Performance to Support Dynamic Scheduling Using the Network Weather Service[*]

Rich Wolski[†]
Computer Science and Engineering Department
University of California, San Diego
La Jolla, CA 92093-0114

Abstract

The **Network Weather Service** *is a generalizable and extensible facility designed to provide dynamic resource performance forecasts in metacomputing environments. In this paper, we outline its design and detail the predictive performance of the forecasts it generates. While the forecasting methods are general, we focus on their ability to predict the TCP/IP end-to-end throughput and latency that is attainable by an application using systems located at different sites. Such network forecasts are needed both to support scheduling [5], and by the metacomputing software infrastructure to develop quality-of-service guarantees [10, 17].*

Keywords: scheduling, metacomputing, quality-of-service, statistical forecasting, network performance monitoring

1. Introduction

As network technology advances, the resulting improvements in interprocess communication speeds make it possible to use interconnected but separate computer systems as a high-performance computational platform or *metacomputer*. Effective application scheduling (particularly of distributed parallel applications) is fundamental if such metacomputers are to be used successfully. Since the resources composing a metacomputer are shared, contention causes their load and availability to vary over time. As a result, the performance that each resource can deliver to an application also varies with time.

Recent work shows that parallel applications can use non-dedicated metacomputers to achieve high-performance without gang-scheduling or other centralized scheduling policies [5, 11, 15, 30]. To gain the desired levels of performance, however, the scheduling methods described in these works depend on *predictions* of the performance deliverable to the application from the available metacomputing resources. Many of these systems rely on static predictions that are supplied to the scheduler when it is configured and do not change. However, contention for shared resources by competing applications causes the performance any one application can obtain to vary over time. In this paper, we describe a distributed service that continually forecasts the performance that is obtainable from networked resources in a metacomputing system so that parallel application schedulers can adapt to changing load conditions. The service operates a distributed set of sensors from which it gathers readings of the instantaneous conditions. It then uses numerical models to generate forecasts of what the conditions will be for a given time frame. We think of this functionality as being analogous to weather forecasting, and as such, term the service the *Network Weather Service* (**NWS**).

We have developed the NWS for use by schedulers in a networked computational environment. Systems such as those outlined in [15, 30, 29, 10] can use NWS forecasts to parameterize their respective scheduling methods and thereby generate schedules that are sensitive to load variation. In [5, 4] we report on the efficacy this technique for parallel applications in production distributed computing environments. As a result of its success, we are currently implementing versions for two production metacomputing systems: Legion [17] and Globus/Nexus [10]. In [15], the authors report similarly positive results for parallel applications using dynamic performance forecasting as the basis for scheduling, and other work indicates that dynamic information can be used to enhance the performance of world-wide-web applications [7]. In this paper, we focus on the problem of network performance forecasting to support parallel application scheduling. We have developed a prototype of the NWS that forecasts network performance (latency and bandwidth) and available CPU percentage for each machine that it monitors, and advertises that information to all interested

[*] Supported by NSF grant ASC-9308900 and Advanced Research Projects Agency/ITO, Distributed Object Computation Testbed, ARPA order No. D570, Issued by ESC/ENS under contract #F19628-96-C-0020.

[†] email: rich@cs.ucsd.edu, http://www-cse.ucsd.edu/users/rich/

schedulers.

The forecasting methods we have implemented fall into three categories:

- *mean-based* methods that use some estimate of the sample mean as a forecast,
- *median-based* methods that use a median estimator, and
- *autoregressive* methods.

To gauge the effectiveness of each method, we report both the mean square prediction error, and the mean percentage prediction error generated by each method as accuracy measures. While mean-based predictive methods generally yield lower mean square error measures, and median-based methods are better in terms of mean percentage error, the best forecasting technique for each setting is difficult to predict. The system, therefore, tracks the accuracy (using prediction error as an accuracy measure) of all predictors, and uses the one exhibiting the lowest cumulative error measure at any given moment to generate a forecast. In this way, the NWS automatically identifies the best forecasting technique for any given resource.

In the next section (Section 2) we describe the structure and implementation of a prototype NWS we have implemented. Section 3 describes the sensory mechanisms and Section 4 describes the forecasting methods we have currently implemented. In Section 5 we compare NWS measurements with their corresponding forecasts for several different network environments. We conclude in section 6 with an evaluation of the results, and a description of our future research.

2. Structure and Implementation

To serve as a viable tool for scheduling, the Network Weather Service must

- *sense* resource performance throughout the system,
- *forecast* the future performance of each resource, and
- *disseminate* the forecast information to all interested client schedulers.

Unlike the real-world weather service, however, the operation of the NWS can significantly change the conditions which it is attempting to forecast. Moreover, we do not assume that computational or network resources within the system will be devoted exclusively to the NWS as such resources would constitute possible failure points and performance bottlenecks. Our view, instead, is that the NWS must limit its intrusiveness so that its resource consumption does not adversely impact the performance of the applications it is designed to serve. The need to limit the intrusiveness of the NWS influences both the implementation of the overall system and the forecasting techniques we have chosen. Since the problems of non-intrusive resource monitoring [26, 21, 9] and load forecasting [24, 3, 18, 23, 8] both pose open research questions, we have separated the sensory and forecasting functions of the NWS. The resulting modular design is intended to provide a general facility in which a variety of different monitoring and forecasting techniques can be employed easily. Figure 1 depicts the architecture of the system.

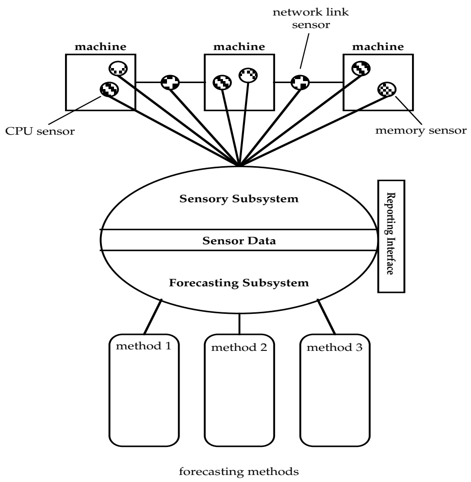

Figure 1. The Structure of the Network Weather Service.

Sensory data is compiled into a logically central (although physically distributed) database to serve as inputs to a collection of forecasting models. Each sensor periodically takes a performance measurement from the resource it is monitoring and stores it with a time stamp in the database. The resulting collection of measurements (ordered by time stamp) form a time series describing the behavior of the resource from which they were taken. That is, each resource is characterized by its own time series. Based on this information history and any *a priori* knowledge of a resource's expected performance response, the NWS forecasting subsystem generates a prediction of what the performance will be for each resource during a given time frame. Once generated, the current forecast data, along with quality measures describing its accuracy (i.e. mean square prediction error, mean percentage prediction error, etc.) are published according to the specifications of an independent reporting interface. Each of the sensory, forecasting, and reporting functions is implemented as a separate subsystem providing a generalizable facility that is easily extended and modified.

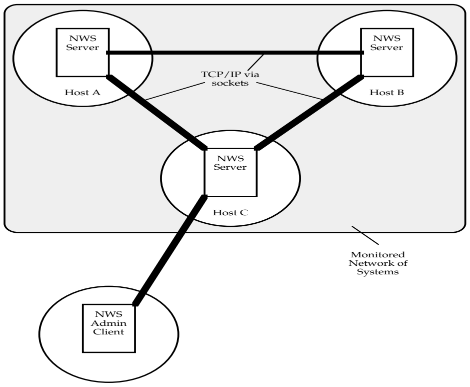

Figure 2. NWS Servers Running on Three Monitored Hosts

3. Sensing

NWS sensors report the observed performance that a resource is able to deliver at the time a measurement is taken. A network link sensor, for example, reports periodic measurements of latency and bandwidth across a particular link. Measurements are taken as close to the application level as possible, since the goal is to forecast the performance an application can actually obtain from each resource. The units of measurement that each sensor uses depend, in general, on the demands of the forecasting models[1].

Each machine to be monitored must execute a copy of the NWS server. Figure 2 depicts three monitored hosts, A, B, and C within the monitored region (shown shaded in the figure). An administrative client utility that controls the system may execute on any machine inside or outside the monitored network, requiring connectivity to at least one of the NWS servers.

Currently, each server maintains a network performance sensor and a CPU availability sensor. All servers in the system share a common list of hosts being monitored, and the TCP port number to which each server is attached. Periodically, each server chooses a host from the list and conducts a communication "experiment" with that host. During an experiment, the round-trip time of a single-word packet is measured. The resulting value is divided by two to yield an approximation of the latency or start-up overhead associated with a communication. Immediately after the latency has been estimated, the initiating server sends a predetermined (and parameterizable) quantity of data and times the transfer. Throughput is then calculated as the data size divided by the transfer time (Equation 1).

$$\textit{throughput} = \textit{data size}/(\textit{data transfer time}) \qquad (1)$$

The resulting measure includes the overhead necessary to initiate a TCP/IP communication stream, which can be significant. To calculate the effective throughput rate, the latency is subtracted from the time recorded for the data transfer, and the result is used as the actual time to transfer the data (see Equation 2).

$$\begin{aligned} &\textit{effective\_throughput} \\ &= \textit{data size/(data transfer time - latency)} \qquad (2) \end{aligned}$$

After an experiment is complete, the latency, throughput, and effective throughput are recorded in an internal database.

Each server also periodically records the local CPU availability using the Unix utilities *vmstat* or *uptime*. If vmstat is available, the sensor parses its output to determine the percentage of time the system is idle, the time spent (by the machine) executing in system space, the percentage of time spent executing user processes, and the number of running processes. Using these measures, it estimates the percentage of time the system is willing to devote to a single application. If *vmstat* is not available, then the CPU monitor uses uptime in the manner described by the local Unix utility reference manual.

4. Forecasting

The NWS operates a set of forecasting methods that it can invoke dynamically, passing as parameters the performance measurements it has taken from each resource. In this section we describe the methods we have included in the current implementation. After each new measurement is taken, it is passed to all of the methods, and a new forecast is generated. That is, for each forecasting method f at measurement time t,

$$\begin{aligned} prediction_f(t) \quad &= \\ METHOD_f(value(t), history_f(t)) & \qquad (3) \end{aligned}$$

where

$$\begin{array}{rcl} value(t) & = & \text{the measured value at time } t, \\ prediction_f(t) & = & \text{the predicted value made by method } f \text{ for measurement } value(t+1), \\ history_f(t) & = & \text{a finite history of measurements, forecasts, and residuals generated previously to time } t \text{ using method } f\text{, and} \\ METHOD_f & = & \text{forecasting method } f. \end{array}$$

[1] We have based an initial sensory subsystem implementation on the TCP/IP socket functionality provided by the *netperf* network performance utility, although we have modified the code substantially. Netperf proved to be a robust and powerful substrate for our purposes, and we encourage those interested to visit the netperf World Wide Web site at [27] for further information.

The values supplied by the sensory subsystem are treated as a time series by the forecasting methods, and each method maintains a history of previous activity and accuracy information. In particular,

$$err_f(t) = value(t) - prediction_f(t-1) \tag{4}$$

is the error residual associated with a measurement and a prediction of that measurement generated by method f.

The current implementation generates a forecast every time a measurement is taken. Since each method is evaluated whenever a new datum is available, we restrict ourselves to those methods we can implement with limited computational complexity. However, an alternative implementation we are considering generates forecasts only when they are requested by a client; this approach may make more computationally complex methods feasible.

4.1. Mean-based Methods

One class of predictors that we have investigated uses arithmetic averaging (as an estimate of the mean value) over some portion of the measurement history to predict the value of the next measurement. The running average, defined as

$$RUN\_AVG(t) = \frac{1}{t+1}\sum_{i=0}^{t} value(i) \tag{5}$$

uses the average of the measurement taken at time t with all previous measurements as a predictor of the measurement to be taken at $t+1$.

Since the running average considers the entire history of measurements when making each forecast, the weight given to each measurement decreases linearly with time. If the most recent values better predict the next measurement, then an average taken over a fixed-length history (thereby fixing the weight given to each measurement) will be a better predictor. The fixed-length or "sliding window" average is calculated as

$$SW\_AVG(t,K) = \frac{1}{K+1}\sum_{i=t-K}^{t} value(i) \tag{6}$$

where $K \geq 0$ is an integer specifying the number of samples to consider in the window. Note that for $K = 0$, SW_AVG uses the last measurement only as a predictor. That is,

$$LAST(t) = SW\_AVG(t,0) \tag{7}$$

Recent work by Harchol-Balter and Downey [20] indicates that this is a useful predictor for CPU resources, hence we include it as a separate method.

The choice of K for SW_AVG may be difficult to determine *a priori* for each resource, and in fact, may vary over time. To set K dynamically so that it adapts to the time series, we employ a gradient-descent strategy. Let $K(t)$ be the value of K at time t, and

$$err_i(t) = (value(t) - SW\_AVG(t, K(t)+i))^2.$$

Then we define

$$\begin{aligned} & ADAPT\_AVG(t) \\ = & \begin{cases} SW\_AVG(t,K(t)-1) & \text{if } \min\limits_{i=-1,0,1} err_i(t) = err_{-1}(t) \\ SW\_AVG(t,K(t)) & \text{if } \min\limits_{i=-1,0,1} err_i(t) = err_0(t) \\ SW\_AVG(t,K(t)+1) & \text{if } \min\limits_{i=-1,0,1} err_i(t) = err_1(t) \end{cases} \end{aligned} \tag{8}$$

and

$$\begin{aligned} & K(t+1) \\ = & \begin{cases} K(t)-1 & \text{if } \min\limits_{i=-1,0,1} err_i(t) = err_{-1}(t) \\ K(t) & \text{if } \min\limits_{i=-1,0,1} err_i(t) = err_0(t) \\ K(t)+1 & \text{if } \min\limits_{i=-1,0,1} err_i(t) = err_1(t) \end{cases} \end{aligned} \tag{9}$$

The value of K is adjusted at each time step in the direction that yields the lowest error. We use a measure of the square error in $err_i(t)$ arbitrarily. It is also possible to use a measure of the absolute percentage error, but our initial experiments indicate that the results are similar. Note that the value of $K(t)$ must be carried as part of the history for $ADAPT\_AVG$, and that $K(0)$ is set to some reasonable starting value. We also arbitrarily restrict K to be between a predetermined maximum and minimum. Setting a maximum threshold limits the computational complexity of the predictor; the minimum value prevents it from becoming "stuck" in a local minimum. In the experiments presented in the next section, we set $5 <= K <= 50$.

Stochastic gradient or recursive prediction error estimators are powerful predictive techniques with recursive formulations [25]. For example, modern implementations of the TCP/IP protocol include a dynamic predictor of end-to-end round-trip time based on stochastic gradient filter [28]. We follow the exposition of the technique provided in [22] which includes a description of a very efficient implementation for the Unix kernel. We define

$$\begin{aligned} & GRAD(t,g) \\ = & \ (1-g) * GRAD(t-1,g) + g * value(t) \end{aligned} \tag{10}$$

for a *gain* $(0 < g < 1)$. The choice of g controls the accuracy with which $GRAD$ estimates the mean value of the time series and the lag time until it converges to a stable estimate. $GRAD$ oscillates randomly about the true average with a standard deviation $\sigma_{GRAD} = g * \sigma_{value(t)}$. Hence a larger value of g will yield a more widely varying estimate. However, $GRAD$ converges exponentially with time constant $1/g$ to the true mean. If the time series is not stationary, $GRAD$ must reconverge as the mean moves. The convergence rate must be faster than the drift in the mean or the predictor will fail to converge. Empirically, a value of 0.05 works well, although we expect to study the problem of finding an appropriate g further. We have experimented with techniques to dynamically adapt g on the fly, but have yet to identify an effective method for the resources we currently monitor with the NWS.

4.2. Median-based Methods

The median value can also serve as a useful predictor, particularly if the measurement sequence contains randomly-occurring, asymmetric outliers. Our presentation of these techniques follows the exposition in [19] and [12]. The median over a sliding window of fixed length whose leading edge is the most recent measurement is used as the forecast for the next measurement. That is, we define

$$\begin{aligned} Sort_K = &\ \text{the sorted sequence of the } K \text{ most recent} \\ &\ \text{measurement values,} \\ Sort_K(j) = &\ \text{the } jth \text{ value in the sorted sequence,} \end{aligned}$$

and

$$\begin{aligned} &MEDIAN(t, K) \\ = &\begin{cases} Sort_K((K+1)/2) & \text{if } K \text{ is odd,} \\ \frac{Sort_K(K/2) + Sort_K(K/2+1)}{2} & \text{if } K \text{ is even.} \end{cases} \end{aligned} \tag{11}$$

As with SW_AVG the choice of K may be difficult to determine. We, therefore, include an adaptive median filter that is analogous to $ADAPT\_AVG$ (Equation 8).

$$\begin{aligned} &ADAPT\_MED(t) \\ = &\begin{cases} MEDIAN(t, K(t)-1) & \text{if } \min\limits_{i=-1,0,1} err_i(t) = err_{-1}(t) \\ MEDIAN(t, K(t)) & \text{if } \min\limits_{i=-1,0,1} err_i(t) = err_0(t) \\ MEDIAN(t, K(t)+1) & \text{if } \min\limits_{i=-1,0,1} err_i(t) = err_1(t) \end{cases} \end{aligned}$$

where $K(t)$ is the value at time t and

$$err_i(t) = (value(t) - MEDIAN(t, K(t)+i))^2.$$

$K(t+1)$ is then determined by Equation 9.

Median filters are attractive because they will reject the effects of sharply outlying data points or “impulses” from the forecasts they produce. They lack some of the smoothing power of the averaging based methods, however, resulting in forecasts with a considerable amount of jitter [19]. It is possible to combine the positive advantages of both classes of methods in the form of an α-trimmed mean filter that averages the central $K - 2 * \alpha * K$ values within a sliding window of size K for ($0 < \alpha < 0.5$). We define

$$T = \lfloor \alpha * K \rfloor$$

for window size K, and the trimmed mean to be

$$\begin{aligned} &\text{TRIM\_MEAN(t,K,T)} \\ &= \sum_{j=T+1}^{K-T+1} \frac{1}{K - 2 * T} Sort_K(j). \end{aligned} \tag{12}$$

It is possible to consider gradient adaptation of α in the same manner that we adapt K for $ADAPT\_AVG$ and $ADAPT\_MED$ but the relationship between α and K is not obvious.

4.3. Autoregressive Models

Recent work [3, 18] has shown that aggregate internet packet traffic can be effectively modeled by autoregressive, integrated, moving average (ARIMA) models. Fitting these models to a specific time series requires the solution to a system of potentially non-linear simultaneous equations, making them difficult to use in a dynamic setting. However, fitting a purely autoregressive (AR) model requires only the solution to a strictly linear system of equations that can be solved recursively via the Levinson Recursion [19]. The general form of a pth-order autoregressive model is

$$AR(t, p) = \sum_{i=0}^{p} a_i * value(t - i). \tag{13}$$

If the time series is stationary, then the sequence $\{a_i\}$ that minimizes the overall error can be determined by the solution to the linear system

$$\sum_{i=0}^{N} a_i * r_{i,j} = 0 \qquad j = 1, 2, \ldots, N \tag{14}$$

where $r_{i,j}$ is the autocorrelation function for the series of N measurements taken. The Levinson Recursion requires a set of partial correlation (PARCOR) coefficients which can also be derived recursively. Burg [6] and more recently Haddad and Parsons [19] describe a recursive algorithm for calculating both the PARCOR and autoregression coefficients from which we derive our current implementation. We omit the details of the algorithm here due to space constraints, but our implementation follows [19] closely. The algorithm takes time $O(p \cdot N)$ for N measurements, which becomes prohibitive when N is the length of the entire time series. We, therefore, calculate the $\{a_i\}$ coefficient sequence over a sliding window of the K most recent measurements, rather than the entire series of size N. That is, after each measurement is taken, we recompute the autoregressive coefficients $\{a_i\}$ using only the previous K measurements as an approximation of the complete time series.

The choice of parameters p and K are determined by the computational complexity the NWS is willing to tolerate. Making K as large as possible (as close to the size of the history as possible) will yield the best fit, but making K too large causes the execution cost of each forecast to be prohibitive. The value of p should be set according to the decay of the autocorrelation function $r_{i,j}$, the values for which are not computed explicitly by the method[2]. Since the autocorrelations can be computationally expensive to compute, we choose arbitrary fixed values of $p = 15$ and $K = 60$. In future implementations, we plan to derive p algorithmically based on estimates of the autocorrelation values, and K based on p and a maximum computational complexity threshold.

4.4. Dynamic Predictor Selection

Choosing the correct predictive method for each resource that the NWS monitors is difficult. Further, it may be that a particular resource conforms to the assumptions of one method for a period of time, and then changes its behavior so that it is best modeled by a different method. Rather than attempting to choose the correct method *a priori*, our initial implementation maintains *all* of the predictive methods simultaneously, for each resource. Then it uses the error measure calculated in Equation 4 to produce an overall fitness metric for each method. The method exhibiting the best overall predictive performance at any time t is used to generate the forecast of the measurement at time $t + 1$. In the initial implementation of the NWS, we use the mean square prediction error

$$MSE_f(t) = \frac{1}{t+1}\sum_{i=0}^{t}(err_f(i))^2 \qquad (15)$$

and the mean percentage prediction error

$$MPE_f(t) = \frac{1}{t+1}\sum_{i=0}^{t}|(err_f(i)|/value(i) \qquad (16)$$

as fitness metrics for each method f at time t. We then define

$$MIN\_MSE(t) = predictor_f(t) \qquad (17)$$

if $MSE_f(t)$ is the minimum over all methods at time t

and

$$MIN\_MPE(t) = predictor_f(t) \qquad (18)$$

if $MPE_f(t)$ is the minimum over all methods at time t.

That is, at time t, the method yielding the lowest mean square prediction error is used as a forecast of the next measurement by MIN_MSE. Similarly, the forecasting method at time t yielding the lowest overall mean percentage prediction error becomes the MIN_MPE forecast of the next measurement. In a scheduling context, it is unclear which fitness metric — mean square error or mean percentage error — will ultimately yield a better schedule. Indeed, the fitness of each forecasting technique may be application-specific. Therefore, the current system maintains and reports both mean square error and mean percentage error, allowing a specific scheduler to choose either.

Location	Host Type
UCSD Parallel Comp. Lab*	Sparc10
UCSD Parallel Comp. Lab	Sparc5
San Diego Supercomputer Center	Alpha
California Institute of Technology	Hypersparc
University of Oregon	Power Challenge
National Center for Supercomputer App.	Power Challenge

Table 1. Host Locations and Types

5. Forecasting Network Performance

In this section, we present measurements and corresponding forecasts of latency and throughput for network connections between machines. During the experimental period, the NWS also monitored and predicted CPU availability using the same forecasting methods. Not surprisingly, network performance proved to be the more difficult of the two to predict as the CPU measurements were slowly varying by comparison. We, therefore, use the network performance data to illustrate the forecasting functionality of the NWS.

We monitored the TCP/IP connectivity between the hosts shown in Table 1 using the prototype NWS over a 24 hour period, and dynamically forecast the latency and throughput between each pair of hosts. We chose this collection of systems so that we could study the quality of the existing internet connectivity with respect to geographic proximity. In particular, we were interested in identifying representative

[2] The autoregressive model is applicable if the decay in the autocorrelation function is exponential and the value of p is set to the duration of the decay [16]. Our current implementation of the NWS does not attempt to determine the suitability of AR for a particular resource. Instead, it assumes that the autoregressive model is applicable, and tracks the prediction error, using AR only if the error is lower than other competing predictors (see section 4.4).

examples of connectivity for different plausible metacomputing settings. To do so, we report data on the connectivity between the Sparc10 located in the Parallel Computation Lab (PCL) at UCSD (marked with an "*" in Table 1) and the other five systems.

Each of these parings is intended to serve as a representative example. The two PCL machines are connected to the same ethernet segment representing a *intra-lab* connection. The PCL and the San Diego Supercomputer Center (SDSC) are located approximately one-quarter mile apart on the UCSD campus representing the connectivity in a *campus-wide* setting. Caltech is located in Pasadena California, approximately 120 miles north of San Diego representing *intra-state* connectivity. The connection between UCSD and the University of Oregon, located in Eugene, Oregon, represents *inter-state* connectivity, and the connection to the National Center for Supercomputing Applications (NCSA) in Urbana, Illinois represents *transcontinental* connectivity.

All of the data were collected between 6:00 PM on Wednesday, September 18, 1996, and 6:00 PM the following day. The NWS was initiated at the beginning of the experimental period so that the forecasters would have access to no previous information (i.e. all start-up and calibration effects would be visible). Measurements were taken at roughly 30 second intervals, and a latency measurement immediately preceded each throughput measurement. The throughput was measured using a 64K byte data transfer with 4K socket buffers at both the sending and receiving ends. We report all throughput measurements in units of megabits per second (mbits/s), and latency in milliseconds (ms).

5.1. PCL Throughput

In Table 2 we summarize the overall accuracy of each forecasting method when forecasting throughput in the PCL over the 24 hour measurement period. The stochastic gradient ($GRAD$) predictor generates the lowest mean square prediction error, and the trimmed mean ($TRIM\_MEAN$) is best in terms of lowest mean percentage prediction error (shown boldface in the table). MIN_MSE shows the ability of the NWS to determine the best predictive method (in terms of mean square error) dynamically without knowing *a priori* that $GRAD$ would perform best. Similarly, MIN_MPE shows the NWS's tracking of mean percentage error. Both of these methods yield error rates that are relatively close to the respective minima, although for this series, all of the forecasting methods except $LAST$ and AR perform reasonably well.

In Figure 3a we show the time series of throughput measurements for the intra-PCL connection. The PCL ethernet segment we monitored is isolated from general internet traf-

Predictor	**MSE**	**MPE**
RUN_AVG	0.5274	0.0927
SW_AVG	0.5041	0.0902
LAST	0.7892	0.1066
ADAPT_AVG	0.5214	0.0925
MEDIAN	0.5386	0.0901
ADAPT_MED	0.5337	0.0896
TRIM_MEAN	0.5130	**0.0893**
GRAD	**0.4903**	0.0895
AR	0.7139	0.0992
MIN_MSE	0.5136	0.0901
MIN_MPE	0.5417	0.0906

Table 2. Forecasting Method Error Statsitics for PCL Throughput Measurements

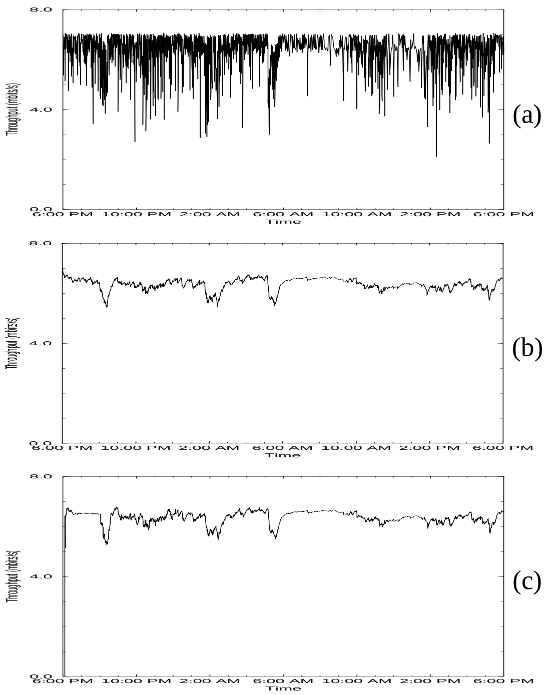

Figure 3. PCL Throughput Time Series (a), $GRAD$ Predictions (b), and MIN_MSE Predictions (c)

fic by a gateway. Even though it is used only by PCL machines, it still displays considerable performance variation. Predictions made by $GRAD$ are shown in Figure 3b and by MIN_MSE in Figure 3c. Except during the initial part of the experiment, the prediction curves are identical. That is, even though we did not know ahead of time that $GRAD$ would be most accurate, the MIN_MSE predictive method automatically identifies it as having the minimum mean square prediction error. The reason that they do not have identical mean square error statistics is that MIN_MSE requires some time to recognize $GRAD$ as the best predictor.

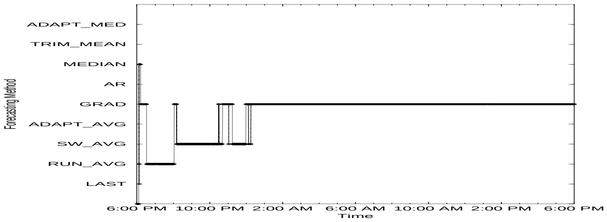

Figure 4. Predictor Selection for MIN_MSE

Figure 4 shows predictor selection as a function of time for MIN_MSE. The heavy horizontal lines indicate which predictor MIN_MSE used at any given point in time. The dotted vertical lines show when the predictor switched from one method to another. Notice that after an initial start-up period (lasting from 6:00 PM until approximately 1:00 AM), MIN_MSE uses the values generated by $GRAD$ for the remainder of the experiment. Since the NWS is intended to be a continuously available service, such a lengthy calibration or start-up period does not pose a serious problem. The results are similar for MIN_MPE; during start-up it switches several times before identifying $TRIM\_MEAN$ as the most accurate predictor in terms of mean percentage error.

5.2. PCL Latency

For the PCL latency measurements, the median-based forecasters generate predictions having the lowest mean percentage error, and very nearly the lowest mean square error. RUN_AVG, generating a mean square error of 0.71, is slightly better than $MEDIAN$ (0.74) in terms of mean square error, but $MEDIAN$ generates a little under half of the mean percentage error (19.8% for RUN_AVG versus 9.6% for $MEDIAN$). Figure 5a shows the time series of latency measurements and in Figure 5b we compare the predictions generated by RUN_AVG (dotted line) to those generated by $MEDIAN$ (solid line).

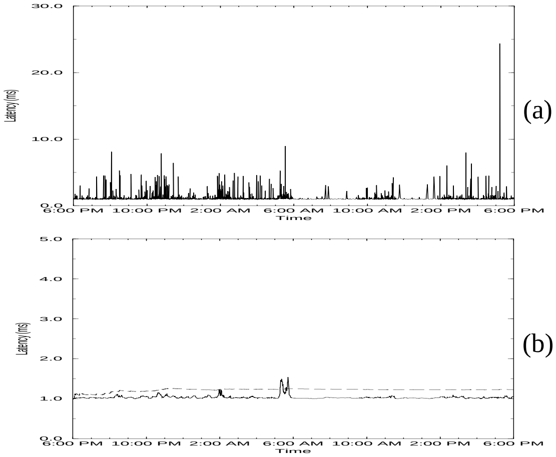

Figure 5. (a) Latency Time Series for PCL, (b) RUN_AVG (dotted) vs. $MEDIAN$ (solid)

In contrast with the throughput time series (Figure 3a), the latency measurements show intermittent outliers departing from an almost uniform value of about 1 ms (Figure 5a). These outliers do not generally form a trackable trend (their duration is short) and they differ from the stable value by an order of magnitude. As such, a median-based forecasting method will reject them in favor of the uniform tendency. Since the outliers all constitute longer latencies (there are no measurements shorter than 1 ms), a mean-based method will be drawn in the direction of the outliers. Figure 5b depicts this relationship. The solid line shows the forecasts generated by $MEDIAN$ and the dotted line above it in the figure are the forecasts made by RUN_AVG. Note that we have changed the scale of the graph to make the difference easier to discern. Since the RUN_AVG forecasts are closer to the outliers when they occur, RUN_AVG yields a lower square error measure and, consequently, a lower overall mean square error. However, RUN_AVG consistently differs from the uniform 1 ms value, so its cumulative mean percentage error is higher. Since the error terms are not squared when calculating mean percentage error, $MEDIAN$ does not accumulate as much error when it encounters an outlier.

Note that $LAST$ exhibits poor forecasting performance for this time series because the presence of an outlier does not indicate that another outlier will follow. Note also that MIN_MSE and MIN_MPE correctly identify and track the best predictor according mean square error and mean percentage error respectively.

5.3. Summary and Analysis of Network Forecasting Performance

A more detailed description and analysis of the forecasting performance can be found in [31]. Tables 3 and 4 summarize the performance of the best predictors for each setting. Notice that $GRAD$ is the overall best predictor of throughput if mean square error is used as an accuracy measure. It fails to yield the lowest error for only the PCL-

Connection	Minimum MSE	Minimum MPE
PCL	$GRAD$	$TRIM\_MEAN$
PCL-SDSC	$GRAD$	$ADAPT\_MED$
PCL-Caltech	$GRAD$	$MEDIAN$
PCL-Oregon	$GRAD$	$ADAPT\_MED$
PCL-NCSA	$LAST$	$LAST$

Table 3. Summary of Best Forecasters for Throughput

Connection	Minimum MSE	Minimum MPE
PCL	$GRAD$	$MEDIAN$
PCL-SDSC	$TRIM\_MEAN$	$MEDIAN$
PCL-Caltech	RUN_AVG	$MEDIAN$
PCL-Oregon	$TRIM\_MEAN$	$MEDIAN$
PCL-NCSA	$GRAD$	$MEDIAN$

Table 4. Summary of Best Forecasters for Latency

NCSA connection, but for that connection it is ranked fifth, (not including MIN_MSE and MIN_MPE in the ranking). In general, though, the mean-based predictors tend to outperform the median-based ones, for throughput time series in this study, if mean square error is used to measure prediction accuracy. Analogously, $MEDIAN$ is the most accurate predictor of latency, in terms of mean percentage error, for each set of latency measurements. MIN_MSE and MIN_MPE correctly track the leading predictor in each case without knowing ahead of time which will be most successful.

Notice also that $LAST$ is not a good predictor of network performance (particularly of latency) *except* for the cross-country internet throughput measurements. In that experiment, however, it performs best. We believe that this result supports those reported in [3] which demonstrate the ability of autoregressive models to correctly reflect aggregate traffic patterns in certain wide-area network environments. In particular, the authors analyze packet data taken from the gateway between SDSC and the NSFNET backbone. The PCL-to-NCSA TCP connection we monitored traverses this gateway. Since we are also measuring the effects of protocol and buffer processing on each end of a connection, we expected aggregate packet behavior to dominate in those settings where network paths include many heavily congested gateways. For the PCL-to-NCSA throughput measurements, indeed, AR performs only slightly worse than $LAST$ as a predictor. The performance of AR is competitive with the other forecasters, in terms of mean square error, for the PCL-to-Oregon throughput series as well.

We summarize these results by noting that:

- if mean square error is the accuracy measure used to judge the fitness of a forecasting method, a *stochastic gradient* predictor is a good choice for forecasting throughput over most internet connections;
- if mean percentage error is used as an accuracy measure, a *sliding-windowmedian* is a good choice of forecasting method for latency, given current internet technology;
- the best predictor of each performance characteristic (latency and throughput in this study) is, in general, not obvious and varies from resource to resource;
- the dynamically-selecting predictive methods successfully track the best predictor in each case yielding forecast error rates close to the minimum.

6 Conclusions and Future Work

To predict the performance of resources in a metacomputing environment, we have developed the Network Weather Service. It operates an arbitrary set of performance sensors, and dynamically generates forecasts from the periodic readings it takes. Determining the most appropriate forecasting method for each resource *a priori* is difficult. Indeed, in the absence of a perfect generating model, the best forecasting method for any particular resource may change over time.

In this work, we illustrate the end-to-end TCP/IP throughput and latency performance an application can obtain between the UCSD Parallel Computation Lab, and a variety of geographically dispersed computing sites. The NWS is able to make dynamic short-term forecasts for both of these communication characteristics, although the accuracy of the forecasts varies from site to site. More importantly, the system can correctly identify the best method "on the fly" based on a running tabulation of prediction error. Since we have designed the system to be extensible, we can incorporate a multitude of techniques from which it can choose the best for any given resource and any given time.

Our work with the NWS is very much in its formative stages. We plan to investigate how the system can incorporate modeling techniques which require a computationally-intensive "fitting" phase. The ARIMA models described in [3], the self-similarity analysis outlined in [24], and the semi-nonparametric techniques discussed in [13, 14], all provide immediately promising avenues of investigation. We would like to discern the relationship between the computational complexity devoted to making a forecast its accuracy. We also plan to integrate other sensory mechanisms such as those described in [7], and to investigate how groups of forecasts may be composed to yield higher-level performance characteristics.

As of this writing, second generation implementations of the NWS are underway for the Globus/Nexus[10] and Legion[17] metacomputing systems. These versions will be initially deployed as part of the GUSTO (Globus UbiquitouS Testbed) [10] and DOCT (Distributed Object Computational Testbed) [1] metacomputing testbeds. We plan to use these implementations both to investigate metacomputing scheduling via AppLeS [4, 2] and the development of general quality-of-service mechanisms.

Acknowledgements

We would like to thank Francine Berman, Chandra Krintz, and Jenny Schopf at UCSD and Allen Downey at UCB and SDSC for their comments, suggestions, and criticisms, all of which were invaluable.

References

[1] Distributed object computation testbed. http://www.sdsc.edu/DOCT/QuadPage.html.

[2] AppLeS. http://www-cse.ucsd.edu/groups/hpcl/apples/apples.html.

[3] S. Basu, A. Mukherjee, and S. Kilvansky. Time series models for internet traffic. Technical Report GIT-CC-95-27, Georgia Institure of Technology, 1996.

[4] F. Berman and R. Wolski. Scheduling from the perspective of the application. In *Proceedings of High-Performance Distributed Computing Conference*, 1996.

[5] F. Berman, R. Wolski, S. Figueira, J. Schopf, and G. Shao. Application level scheduling on distributed heterogeneous networks. In *Proceedings of Supercomputing 1996*, 1996.

[6] J. Burg. *Maximum Entropy Spectral Analysis*. PhD thesis, Stanford University, 1975.

[7] R. Carter and M. Crovella. Dynamic server selection using bandwidth probing in wide-area networks. Technical Report TR-96-007, Boston University, 1996.

[8] M. Crovella and A. Bestavros. Self-similarity in world wide web traffic: Evidence and possible causes. In *Proceedings of the 1996 ACM Sigmetrics Conference on Measurement and Modeling of Computer Systems*, 1996.

[9] M. Crovella and T. LeBlanc. Parallel performance prediction using lost-cycles analysis. In *Proceedings of Supercomputing 1994*, 1994.

[10] T. DeFanti, I. Foster, M. Papka, R. Stevens, and T. Kuhfuss. Overview of the i-way: Wide area visual supercomputing. *International Journal of Supercomputer Applications*, To Appear.

[11] A. Dusseau, R. Arpaci, and D. Culler. Effective distributed scheduling of parallel workloads. In *Proceedings of SIGMETRICS/Performance*, May 1996.

[12] N. Gallagher and G. Wise. A theoretical analysis of the properties of median filters. *IEEE Transactions ASSP*, December 1981.

[13] R. Gallant and G. Tauchen. Snp: A program for nonparametric time series analysis. In *http://www.econ.duke.edu/Papers/Abstracts/abstract.95.26.html*.

[14] R. Gallant and G. Tauchen. Seminonparametric estimation of conditionally constrained heterogeneous processes: Asset pricing applications. *Econometrica 57*, pages 1091–1120, 1989.

[15] J. Gehrinf and A. Reinfeld. Mars - a framework for minimizing the job execution time in a metacomputing environment. *Proceedings of Future general Computer Systems*, 1996.

[16] C. Granger and P. Newbold. *Forecasting Economic Time Series*. Academic Press, 1986.

[17] A. S. Grimshaw, W. A. Wulf, J. C. French, A. C. Weaver, and P. F. Reynolds. Legion: The next logical step towrd a nationwide virtual computer. Technical Report CS-94-21, University of Virginia, 1994.

[18] N. Groschwitz and G. Polyzos. A time series model of long-term traffic on the nsfnet backbone. In *Proceedings of the IEEE International Conference on Communications (ICC'94)*, May 1994.

[19] R. Haddad and T. Parsons. *Digital Signal Processing: Theory, Applications, and Hardware*. Computer Science Press, 1991.

[20] M. Harchol-Balter and A. Downey. Exploiting process lifetime distributions for dynamic load balancing. In *Proceedings of the 1996 ACM Sigmetrics Conference on Measurement and Modeling of Computer Systems*, 1996.

[21] J. Hollingsworth, B. Miller, and J. Cargille. Dynamic program instrumentation for scalable performance tools. In *Proceedings of SHPCC 1994*, 1994.

[22] V. Jacobson. Congestion avoidance and control. In *Proceedings of SIGCOMM '88*, volume 18, August 1988.

[23] S. Keshav. A control-theoretic approach to flow control. In *Proceedings of SIGCOMM '91*, volume 24, August 1991.

[24] W. e. a. Leland. On the self-similar nature of ethernet traffic. *IEEE/ACM Transactions on Networking*, February 1994.

[25] L. Ljung and T. Soderstrom. *Theory and Practice of Recursive Identification*. MIT Press, 1983.

[26] A. Malony, D. Reed, and H. Wijshoff. Performance Measurement Intrusion and Perturbation Analysis. *IEEE-TPDS*, 3(4):433–450, July 1992. Available as Tech. Report CSRD-923, University of Illinois, Center for Supercomputing Research and Development. Reprinted in IEEE CS Press Tutorial, *Monitoring and Debugging Distributed and/or Real-Time Systems*, Jeffrey Tsai and S. Yang (Eds.), pp. 77–94, 1995.

[27] Netperf. http://www.cup.hp.com/netperf/netperfpage.html.

[28] P. e. a. Postel. Transmission control protocol specification, 1981. ARPA Working Group Requests for Comment DDN Network Information Center, SRI International, Menlo Park, CA, RFC-793.

[29] T. Tannenbaum and M. Litzkow. The conder distributed processing system. *Dr. Dobbs Journal*, February 1995.

[30] J. Weissman and A. Grimshaw. A framework for partitioning parallel computations in heterogeneous env ironments. *Concurrency: Practice and Experience*, 7(5), August 1995.

[31] R. Wolski. Dynamically forecasting network performance using the network weather service. Technical Report TR-CS96-494, U.C. San Diego, October 1996. available from http://www.cs.ucsd.edu/users/rich/publications.html.