Implementing a Performance Forecasting System for Metacomputing: The Network Weather Service*

Rich Wolski
Neil Spring
Chris Peterson†

July 10, 2002

Abstract

In this paper we describe the design and implementation of a system called the **Network Weather Service** (NWS) that takes periodic measurements of deliverable resource performance from distributed networked resources, and uses numerical models to dynamically generate forecasts of future performance levels. These performance forecasts, along with measures of performance fluctuation (e.g. the mean square prediction error) and forecast lifetime that the NWS generates, are made available to schedulers and other resource management mechanisms at runtime so that they may determine the **quality-of-service** that will be available from each resource.

We describe the architecture of the NWS and implementations that we have developed and are currently deploying for the Legion [13] and Globus/Nexus [7] metacomputing infrastructures. We also detail NWS forecasts of resource performance using both the Legion and Globus/Nexus implementations. Our results show that simple forecasting techniques substantially outperform measurements of current conditions (commonly used to gauge resource availability and load) in terms of prediction accuracy. In addition, the techniques we have employed are almost as accurate as substantially more complex modeling methods. We compare our techniques to a sophisticated time-series analysis system in terms of forecasting accuracy and computational complexity.

1 Introduction

Fast networks have made it possible to connect distributed, heterogeneous computing resources to form a high performance computational platform, or **metacomputer**. While

*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. Chris Peterson supported in part by ARPA/NCCOSC (Naval Command Control & Ocean Surveillance Center) N66001-96-C-8523.

†email: rich@cs.ucsd.edu, nspring@cs.ucsd.edu, cpeterso@microsoft.com

metacomputing offers tremendous potential performance, realizing that potential depends, in part, on the ability to manage the effects of resource contention on application performance. In particular, resource allocation and scheduling decisions must be based on **predictions** of the performance each resource will be able to deliver to an application during a specified time frame. Fixed estimates based on manufacturer's performance specifications are typically inadequate as they fail to reflect the performance loss due to fair sharing and contention. Moreover, these contention effects vary dynamically as competing applications vary their resource demands.

We have designed and implemented a system that takes periodic measurements of the currently deliverable performance (in the presence of contention) from each resource and uses numerical models to generate forecasts of future performance levels dynamically. Forecast data is continually updated and distributed so that resource allocation and scheduling decisions may be made at run time based on **expected** levels of deliverable performance. To the extent that network performance conditions can be thought of as the "network weather", this functionality is roughly analogous to weather forecasting and hence we term the system the **Network Weather Service** (NWS).

Since the NWS measures and forecasts performance deliverable to the application level, it must be implemented using the same communication and computation mechanisms that applications use so that forecasts accurately reflect the true performance an application can expect to obtain. Initially, we have developed separate implementations of the NWS using sockets (based on the netperf [15] utility) and for the Globus/Nexus [8] and Legion [13] metacomputing environments, each of which provides a software infrastructure that supports high-performance distributed and parallel computing. As part of the AppLeS (Application-Level Schedulers) project [2, 1] we are developing scheduling agents that make decisions based on application-level performance estimates. The functionality of the NWS is motivated by the requirements of these agents. In addition, quality-of-service guarantees in shared network environments (e.g. The Internet) are difficult to achieve. NWS forecasts provide statistical estimates of available service quality from each resource, as well as the degree to which those estimates are likely to be accurate [17].

In this paper, we focus on the architecture and implementation of the Legion and Globus/Nexus Network Weather Service versions. Section 2 describes the high-level architecture of the system and discusses some of the details specific to the Legion and Nexus implementations (the socket version is described more completely in [16]). In section 3, we present comparative forecasting results and we conclude with an evaluation of the current system and a description of future research in section 4.

2 Architecture

In this section, we present some high-level design issues that shaped the architecture of the Network Weather Service (NWS). The NWS was designed as a modular system to provide performance information for distributed application scheduling. Some of its forecasting models (described more completely in [17]) require long-term history information. As such, we

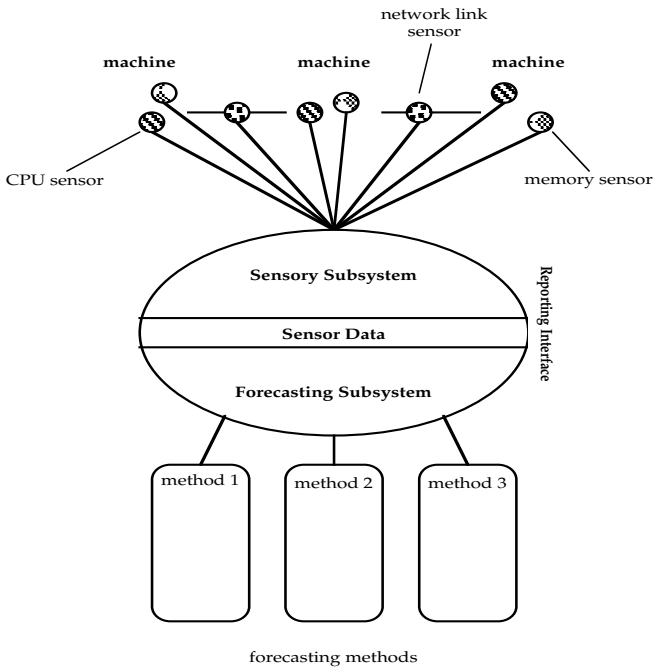


Figure 1: The Logical Structure of the Network Weather Service.

have designed the system to be persistent with the intention that it be a continually available service within the metacomputing environment. Since workstation users must retain autonomy over their own machines, and the chances of resource failure scale with the size of the computing environment, the system must be robust with respect to resource failure. Furthermore, metacomputers are dynamically changing in structure and composition. Resources may be added, deleted, or modified (upgraded, reconfigured, etc.) under the control of their respective owners and managers. The NWS, therefore, must be dynamically reconfigurable to accommodate changes in the underlying metacomputing system.

We have separated the Network Weather Service functionality into three modules:

- a **sensory subsystem** that monitors system-wide resource performance levels,
- a **forecasting subsystem** that predicts future conditions and passes this information to
- a **reporting subsystem** that disseminates the forecast information in various formats.

Figure 1 depicts this logical organization. Measurement data is collected independently from the resource sensors and stored in a physically distributed database by the sensory subsystem. Forecasting models are applied to measurement histories (which may be treated as time series) to generate predictions. The forecasting subsystem also tracks prediction error and maintains accuracy information for each prediction so that the quality of each prediction

may be gauged. External programs, such as user applications, system schedulers, or quality-of-service mechanisms can access forecast information generated by the NWS through the public interface exported by the reporting subsystem.

2.1 Sensory Subsystem

The information given to a scheduler should represent deliverable performance that resources can provide to an application. For this reason, measurements of resource performance are taken at the application level, using the facilities provided by the underlying resource management system. By using the same facilities that are available to an application, NWS measurements are subject to the overheads imposed by the resource management system. Lower level monitoring tools may not capture the effect of such overheads at the application-level, particularly with respect to contention.

The Network Weather Service distinguishes between passive sensors and active sensors. A passive sensor, such as the CPU availability sensor, exercises an external system utility and scans the utility's output to obtain information describing a number of resources. For example, memory usage and CPU availability can be tracked by executing the Unix utilities **vmstat** on each machine and processing the output. It is important that the external utilities be non-privileged with respect to an application. If the information provided by the NWS can only be obtained by a privileged process, it would not only violate local security constraints, but it would not truly represent what is visible to a typical application.

An active sensor, on the other hand, must explicitly measure the availability of the resource it is monitoring. To test a resource, an active sensor will conduct a **performance experiment** that is intended to be representative of a typical resource access. Again, such accesses must be made with standard user privileges so that they reflect true application-level performance.

2.1.1 Sensing the Network

The NWS currently monitors both process-to-process latency and throughput throughout the system using an active sensor. Figure 2 details the performance experiment conducted by NWS network sensors. Network latency is the minimum transit delay when transmitting a message. The NWS approximates the one-way message latency as one-half of the round-trip time for an arbitrarily small message.

Network throughput is defined to be the effective rate at which bits can be sent from one process to another. To perform a throughput experiment, a message of significant size is sent between processes and the time required to complete the transfer and receive an acknowledgement is recorded on the sending side. The length of this message is parameterizable, depending on the speed of the connection, the physical proximity of the machines, and the degree of intrusiveness a particular connection can support. In the current implementations, this size is set by the NWS administrator, although we are considering ways in which it can

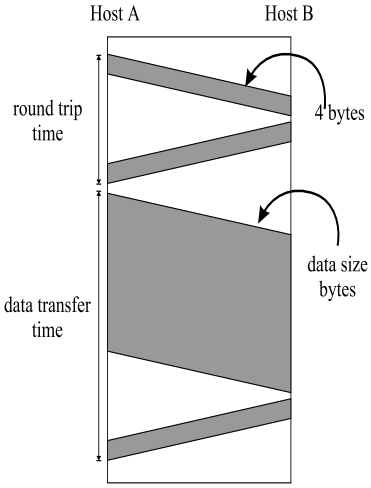


Figure 2: Throughput and Latency Experiments.

be determined automatically by the NWS itself. Throughput is calculated as

$$\textbf{effective throughput} = \frac{\textbf{data size}}{\textbf{data transfer time - predicted round-trip time}}$$

where the **predicted round-trip time** comes from the forecast for latency between the two processes.

The sensory subsystem must control all network experiments so that they do not consume an appreciable portion of the resource they monitor. Setting the transfer size to be large yields lower prediction errors (e.g. high quality point estimates of available throughput). However, the throughput that is consumed by the experiment is unavailable for application use while the experiment is underway. That is, the experiment, itself, may contend with the application. Alternatively, a small transfer size will yield less accurate predictions, but intrude less on the available network performance. Deriving the best transfer size value for each network setting is the subject of our current work. The implementations we have deployed currently use transfer sizes that we have observed to work well in practice.

The sensory subsystem must also take care not to inadvertently measure the effects of its own probes as ambient contention. For example, simultaneous throughput experiments conducted by separate hosts attached to a single ethernet segment will interfere. Figure 3 depicts the effects of this interference. Each data point represents the throughput (in megabits per second) that was obtained by a pair of processes performing 64 kilobyte transfers over an ethernet segment using the Mentat [12] communication primitives available with Legion [13][1]. The time at which each experiment was conducted is represented along the horizontal axis so that the graph shows a time series of measurements. One measurement was made every ten seconds over a twenty-four hour period. At the time corresponding to the vertical line

[1]The Mentat Programming Language is an object oriented dialect of C++ which allows program-level access to the Legion metacomputing software infrastructure.

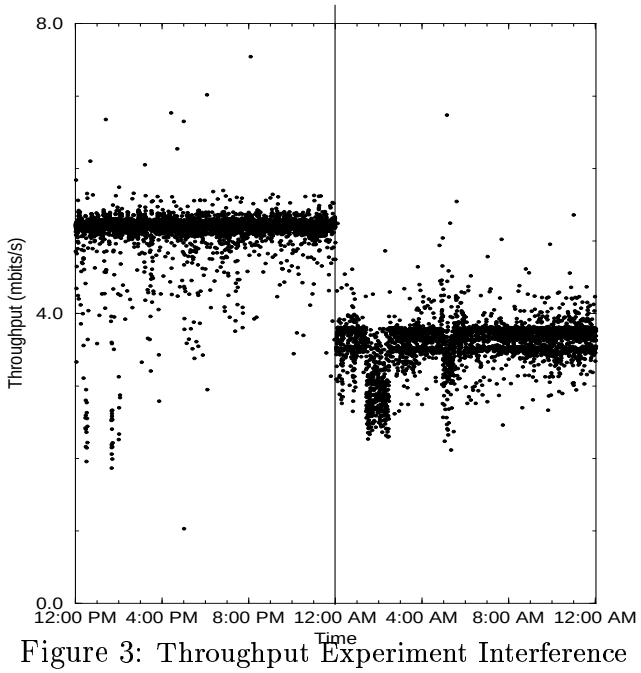


Figure 3: Throughput Experiment Interference

in the figure, (approximately twelve hours after the beginning of the time series) we started a second set of periodic measurements (also using 64 kilobyte transfers) between a different pair of hosts connected to the same ethernet segment. To measure the worst-case effect, we endeavored to make the network probes collide as much as possible. The difference between the left hand side of the figure and the right results from the collision of, and subsequent contention between, simultaneous network experiments.

Note that the contention effects depicted in Figure 3 affect the forecast that the NWS should report to an application scheduler. If one probe is periodically active, the time series on the left hand side of the figure should be used to generate a forecast. If two probes are periodically active, the right hand side data should be used. Visually, it is clear that any reasonable forecasting technique will yield different predictions for the two different time series. The right hand time series is particularly problematic. It is largely bimodal, but a periodogram [11, 14] fails to reveal predictable cycles within the data. That is, the measurements do not switch periodically between one mode and the other. Consequently, to make a reasonable forecast from this data, an application of time series analysis requires four distributions: one for each mode, and one each describing the transitions between modes. Deriving these distributions and then using them effectively to generate forecasts is considerably more complex than in the unimodal case. Further, if the two probes are allowed to contend randomly, the resulting series will be some combination of the two shown in the figure. As more hosts are added, the effect becomes more pronounced and less predictable.

There are several ways to address the problem of controlling the possible interference between network probes. The method we have chosen to implement first is a token passing scheme that ensures mutual exclusion between experiments. When a network sensor receives a token, it is entitled to conduct a single network performance experiment and then to pass

the token on to a successor. Since only one performance experiment can be active at a time, collisions between experiments are avoided. In addition, we incorporate a simple token-recovery scheme in case of a resource failure. If the token cannot be passed to a host, that host is automatically configured out of the system until its continued functionality can be verified. If a system crashes while holding the token, a pre-elected "master" detects token-death via a timeout. The token master can be switched dynamically between hosts should the master, itself, fail.

It is obvious that this token-passing method for ensuring non-interference between experiments does not scale. While we are investigating ways to filter the effects of possibly contending network experiments, we are planning to implement a large-scale NWS as a hierarchy of potentially synchronizing pools of hosts. For example token passing instances (called **cliques** in our terminology) can be organized as a tree where each level in the tree forms a clique. The leaves of the tree represent individual machines. In each clique (corresponding to a level in the tree), an individual representative node is designated to also participate in the clique above it (at the next lower level). Data up to the nearest common ancestral clique between two hosts need only be considered when estimating the network performance between them. We will also consider methods other than token-passing for controlling interference between network experiments within a clique, in future implementations.

2.1.2 Sensing the Machines

To measure the availability of a machine, the Network Weather Service uses a passive sensor module. Currently, the CPU sensor starts the Unix **vmstat** system utility as a background process and periodically scans the output. The vmstat output is parsed to pick out the number of running processes, and the percentage of the total time the system is spending in user and supervisor state respectively. The Network Weather Service computes the percentage of the time that a processor will be available to each running process as

$$\begin{aligned} availableCPU &= T_{idle} + (T_{user}/rp) \\ &\quad + (T_{user} * T_{system}/rp) \end{aligned}$$

where

T_{idle}	=	percentage of time cpu is idle
T_{user}	=	percentage of time cpu is executing user code
T_{system}	=	percentage of time cpu is executing executing system code
rp	=	runnable processes

Using this value, a scheduler can compute the CPU slowdown a process will experience due to contention. The rationale for this formula is that a new job (running with standard priority) should be entitled to all of the idle time, and a fair share of the available user state time. Our experience has been that system cycles (represented T_{system} in the equation) are shared fairly in proportion to the amount of time the system, as a whole, spends executing in

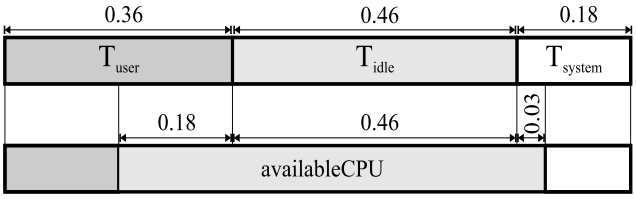


Figure 4: Available CPU calculation. rp = 2 runnable processes. $\boldsymbol{T_{user}} = 0.36$, of which half (0.18) would be usable to a second process. $\boldsymbol{T_{idle}} = 0.46$, all of which is available for computation. $\boldsymbol{T_{system}} = 0.18$, of which 0.03 can be shared. The total **availableCPU** = 0.67.

user state. While this empirically derived formula has worked well for some applications [3], the NWS can easily accommodate more sophisticated techniques such as those described in [5].

2.2 Reporting Subsystem

Network Weather predictions are accessible via an exported reporting interface. The intrusiveness of the interface (i.e. the network and computational resources required to disseminate forecast information) is a key concern in its design. Stored data is distributed across the system to balance the storage load and to avoid a bottleneck at the interface. Each host maintains a copy of the current state of the network, and the one-step-ahead predictions of future conditions. To amortize the overhead associated with exchanging state, each NWS server sends a copy of its local state as part of every throughput communication experiment it conducts. The data stored at each host is shown in Figure 5. Storing a global image of the current and predicted state of the network on each host allows clients to access this data from a number of different sources, each of which has a copy. Time series history information is stored by individual hosts, and is transferred only by the request of an interested client. This form of on-demand transfer minimizes the amount of data stored on each host, while still providing detailed forecast information to clients efficiently. Access to both remote and local data is provided through the communication primitives of the underlying resource management system.

Users can view NWS weather reports using the World Wide Web. The Network Weather Service on each machine continuously generates a publicly accessible HTML file containing a "snapshot" of the most recent forecasts. With their web browers, users can watch current and predicted network conditions fluctuate as the Network Weather Service monitors the network.

3 Results

To forecast resource performance, the NWS treats periodic measurements taken from a particular sensor as a time series, and then uses different statistical models to predict the

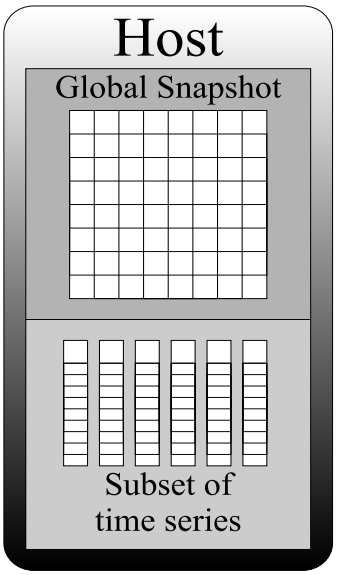


Figure 5: Data stored on each host.

next value in the series. In this section, we report the performance of the forecasting system (in terms of prediction accuracy) for several different resources. We compare measurements and forecasts made using the Legion and Globus/Nexus systems to consider the effect of the underlying resource management system on performance forecasting. We also provide a brief investigation of the relationship between the computational complexity associated with generating a forecast and its accuracy.

3.1 Forecasting Subsystem

The Network Weather Service uses a number of predictive algorithms to anticipate performance fluctuations. Sensory data is ordered by time stamp so that the forecasting models may treat each prediction history as a time series. In [16, 17] we detail the specific algorithms that are part of the current NWS implementations. From the perspective of the forecasting subsystem's implementation, each forecasting model is an independent module that imports and exports a common interface. When a forecast is required, the NWS evaluates a set of different forecasting models and then automatically chooses between them based on the accuracy history of each model. Currently, the models that are included are computationally cheap to compute so that the system can adapt them dynamically to changing conditions. We discuss this issue further in Section 3.3.

3.2 Process-to-Process Throughput

We monitored the effective throughput using the Legion version of the NWS between two adjacent sun workstations on an ethernet segment in the Parallel Computation Lab (PCL)

Predictor	MAE
LAST	0.28
RUN_AVG	0.23
SW_AVG	0.23
MEDIAN	0.18

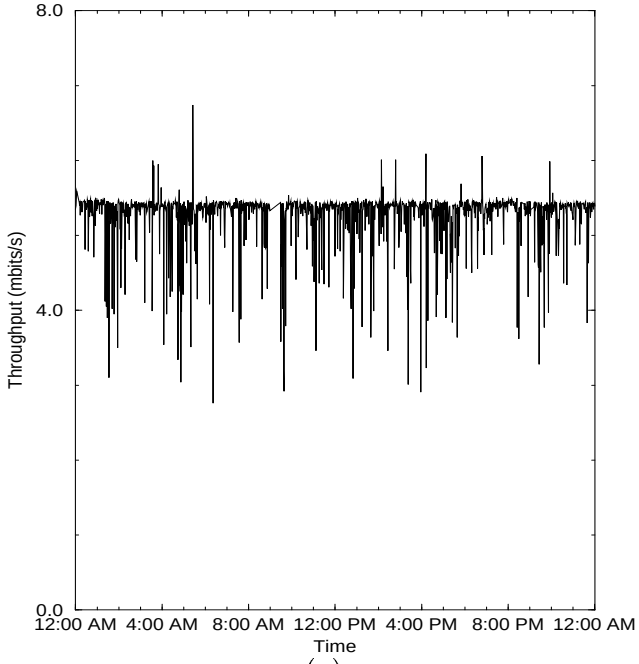
Table 1: Forecasting Error for PCL Throughput using Legion. Lower values indicate better predictor performance.

at UCSD. Once every sixty seconds over a 24 hour period (starting at midnight on Tuesday, February 4, 1997), the NWS moved a 64 kilobyte array via a Mentat method invocation between the two hosts and timed the transfer. Figure 6(a) shows the time series of measurements and Figure 6(b) shows the corresponding predictions made by the NWS. The units of measurement are megabits per second and the mean absolute error (MAE) for the prediction is **0.18**. That is, during the measurement period, at each time step the prediction of a measurement differed from the actual measurement it was predicting by an average of **0.18** megabits per second. Table 1 summarizes the predictive performance of several different forecasting methods. The NWS currently supports a variety of forecasting techniques, the details of which are more completely described in [16, 17]. In the interest of brevity, we demonstrate its functionality using only forecasters that are based on the common summary statistics shown in the table. The complete battery of techniques includes these simple ones shown in the table as well as other, more sophisticated methods. For the purposes of illustration, however, we restrict ourselves here to a simple set.

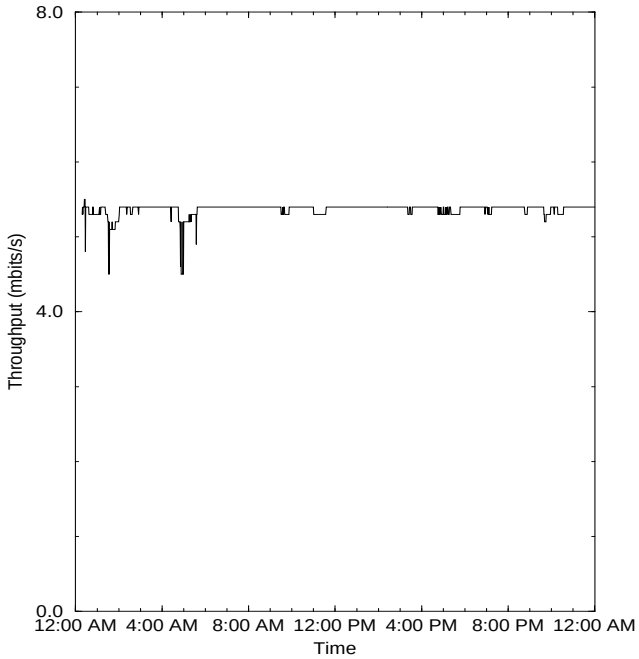
The ***LAST*** predictor uses the last measurement as a prediction of the next measurement. ***RUN_AVG*** keeps a running tabulation of the average measurement and uses that as a prediction at each time step. ***SW_AVG*** uses the average of the current measurement and the previous 20 measurements (a sliding window of 21 measurements) as a predictor of the next value and ***MEDIAN*** uses the median over the same sliding window as a predictor. By tracking the prediction error made by each predictor, the NWS was able to identify ***MEDIAN*** as the most accurate (yielding the lowest average error).

Note that ***LAST***, by comparison, is not a good predictor yielding an absolute prediction error that is an average of 0.1 megabits per second (55%) greater than that generated by ***MEDIAN***. In general, our experience has been that the last measurement is a poor predictor of future network performance. Performance monitoring systems often use current measurement data as an estimate of the available throughput. Clearly, even for a predictable network like the one shown in Figure 6, simple probes measuring current conditions are poor indicators of the performance that will be available in the next time step.

In Figure 7 we show throughput measurements and predictions generated using Nexus remote service requests between processes running on workstations in the UCSD PCL and at the San Diego Supercomputer Center (SDSC). As in the previous experiment, the NWS moved 64 kilobytes of data every 60 seconds and timed the transfer. The resulting throughput between the PCL and SDSC was recorded in units of megabits per second. Figure 7(b) shows

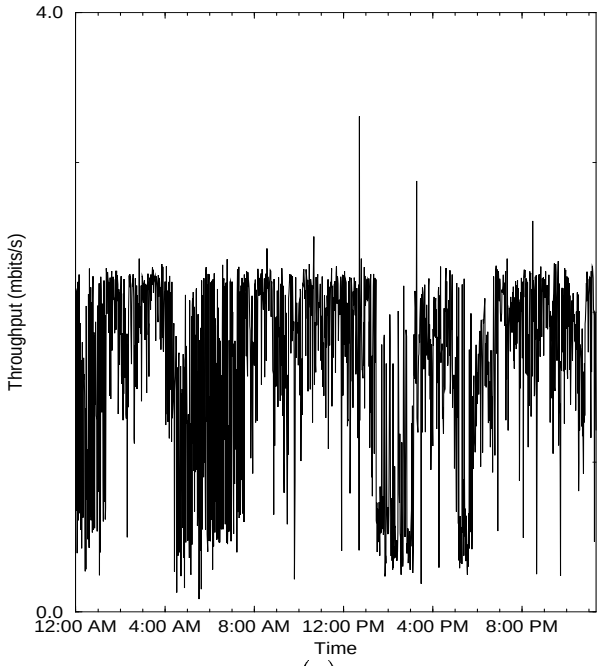


(a)

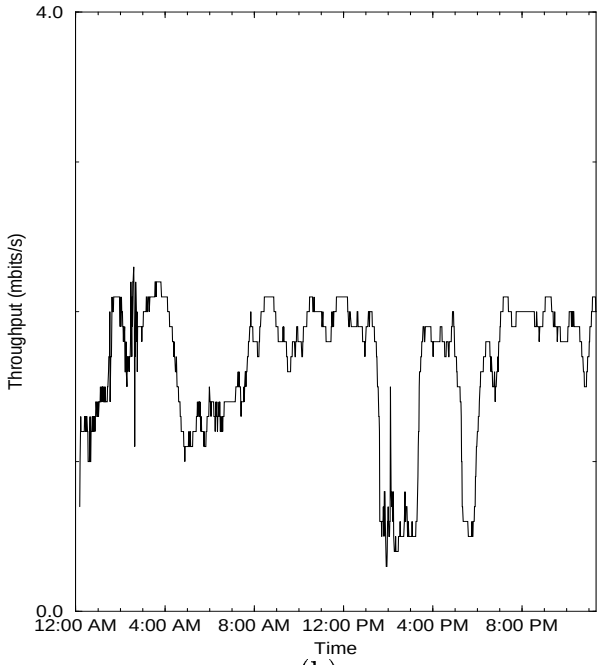


(b)

Figure 6: Legion Throughput Measurements (a) and Predictions (b) in the PCL



(a)



(b)

Figure 7: Nexus Throughput Measurements (a) and Predictions (b) from the PCL to SDSC

Using Nexus	
Predictor	**MAE**
LAST	0.46
RUN_AVG	0.49
SW_AVG	0.39
MEDIAN	0.38

Using Legion	
Predictor	**MAE**
LAST	1.40
RUN_AVG	1.40
SW_AVG	1.30
MEDIAN	1.30

Table 2: Forecasting Error for PCL to SDSC Throughput

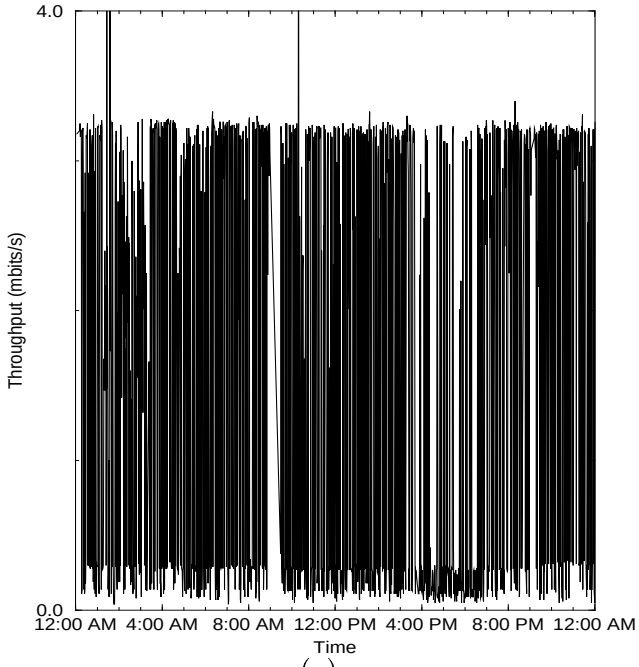
the predictions made by the NWS and Table 2 summarizes the accuracy of each predictor in terms of mean absolute error. In this case, the average absolute error in megabits per second generated by ***MEDIAN*** is **0.08** megabits per second lower than ***LAST*** and **0.11** megabits per second lower than ***RUN_AVG***. The forecasts shown in Figure 7(b) are a combination of ***SW_AVG*** and ***MEDIAN*** as the NWS switched back and forth between them depending on which yielded the lowest MAE at any given point in time. This combinational forecast also yielded a mean absolute prediction error of **0.38** megabits per second.

In Figure 8 we show throughput measurements and predictions for the same PCL-to-SDSC communication link using Legion. Compared to the equivalent set of Nexus measurements shown in Figure 7a, the Legion measurements vary through a wider range. A scatter plot of the Legion measurement data (shown in Figure 8c) reveals a multimodal distribution of measurements. We attribute this multimodality to the use of UDP as Legion's underlying messaging protocol. The timeout value for a lost packet (due to gateway congestion) is one second in the current Legion prototype implementation[2] causing a substantial loss of throughput when a packet is lost. At present, the NWS is unable to predict in which mode a successive measurement will fall at any given moment. Neither a periodogram [11] nor a state-transition analysis [4] yield exploitable predictive information. Consequently, the forecast errors for Legion throughput measurements, shown in Table 2, are higher than for Nexus. Note that the NWS distributes its quantification of forecasting error so that schedulers and quality-of-service mechanisms, such as those proposed in [6], can consider the performance predictability of a resource. If, for example, both the Nexus and Legion messaging systems were available to an application communicating between the PCL and SDSC, a scheduler might choose to use Nexus for the communication due to its greater predictability. Conversely, in a local area setting where packet loss is rare, Legion's messaging system might be more appropriate.

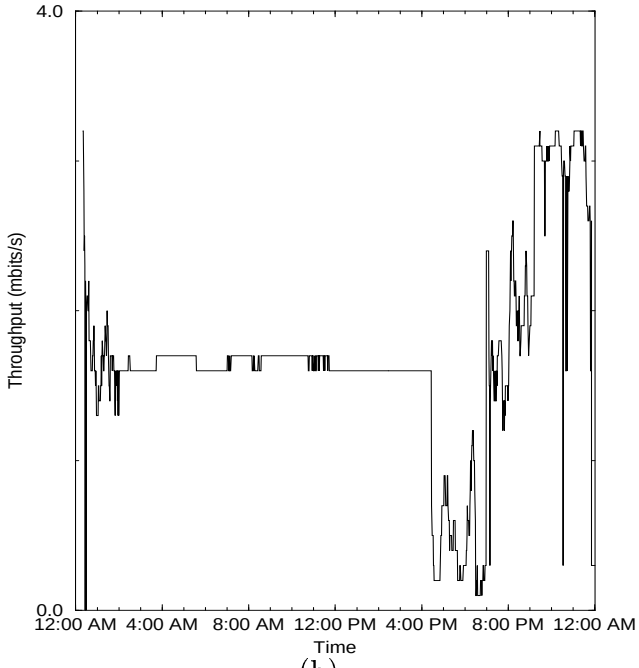
3.3 Forecasting Complexity versus Accuracy

The choice of computationally simple forecasting techniques is not only motivated by our desire to build initial prototypes with which to experiment, but also by our experiences with the use of more elaborate statistical methods in a dynamic setting. Specifically, it is often the

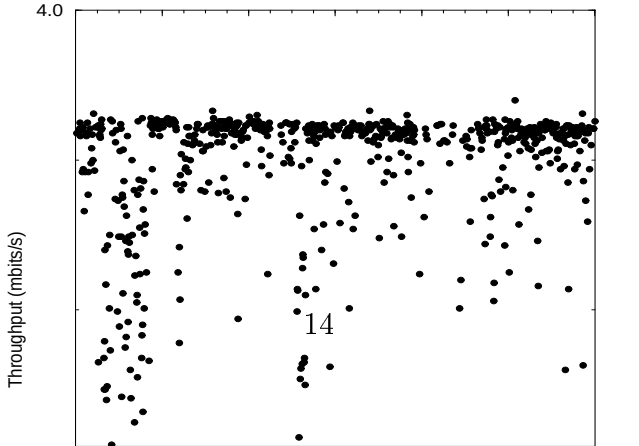
[2]Legion is currently being reimplemented. In the next release of the system, it is our understanding that more robust communication protocols will be available.


4.0
0.0
Throughput (mbits/s)
12:00 AM 4:00 AM 8:00 AM 12:00 PM 4:00 PM 8:00 PM 12:00 AM
Time

(a)


4.0
0.0
Throughput (mbits/s)
12:00 AM 4:00 AM 8:00 AM 12:00 PM 4:00 PM 8:00 PM 12:00 AM
Time

(b)


4.0
Throughput (mbits/s)

Predictor	Description
RUN_AVG	running average
SW_AVG	sliding window avg.
LAST	last measurement
ADAPT_AVG	adaptive window average
MEDIAN	median filter
ADAPT_MED	adaptive window median
TRIM_MEAN	α-trimmed mean
GRAD	stochastic gradient
AR	autoregression
MIN_MAE	adaptive minimum absolute error

Table 3: Summary of Forecasting Methods

case that the additional accuracy provided by a sophisticated, but computationally complex, forecasting technique cannot be amortized effectively.

For example, consider the time series shown in Figure 9(a) which depicts a series of process to process throughput measurements between two workstations connected via ethernet within the UCSD Parallel Computation Laboratory. Each measurement represents the throughput observed by timing a 64 kilobyte transfer between the two workstations using TCP/IP sockets. One measurement was taken every thirty seconds over a twenty-four hour period beginning at 6:00 PM on Thursday, September 19, 1996.

To investigate the relationship between model complexity and accuracy, we compare the predictive performance of a model fitted using Semi-Nonparametric Time Series Analysis (SNP) developed by Gallant and Tauchen [10, 9] to the current set of NWS forecasting methods (shown in Table 3). A complete description of each NWS technique is given in [16]. To make a forecast using these methods, the NWS runs them **all** every time a forecast is required, and then uses the method that yields the minimum absolute error (denoted ***MIN_MAE*** in the table) as the published forecast. In this way, the forecasting system can identify which method has been most accurate over time and use that method for subsequent forecasts. Moreover, if conditions change and another method becomes more accurate, the NWS notices the change and switches methods dynamically. Since all methods must be evaluated every time a forecast is required, we include only those for which we have a computationally efficient implementation.

Alternatively, forecasting using SNP is a two-phase process. First, a model is fit to the time series using an iterative search method that can be computationally complex. Once fit, the model computes the one-step-ahead conditional probability density using a small number of lags (previous measurements). From the conditional density, the system calculates a conditional expected value for the next measurement which we use as a forecast.

From a classical time-series analysis perspective, SNP is a powerful technique. The model fitting phase considers stationary autoregression in various forms (both with Gaussian and non-Gaussian noise terms), autoregressive, conditionally heteroscedastic (ARCH) models

Method	MAE	Cost/Pred (ms/pred)
SNP	0.437	1090 ms/pred
NWS	0.450	7.29 ms/pred

Table 4: Comparison of Forecasting Performance

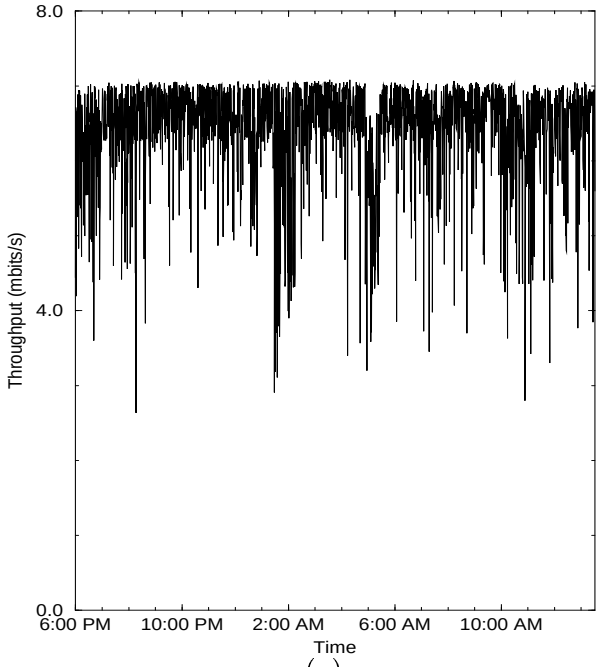
(with and without Gaussian noise), and general non-linear processes with heterogeneous innovations.

Table 4 compares the performance of forecasts generated using SNP, with the techniques shown in Table 3 that are implemented by the NWS.

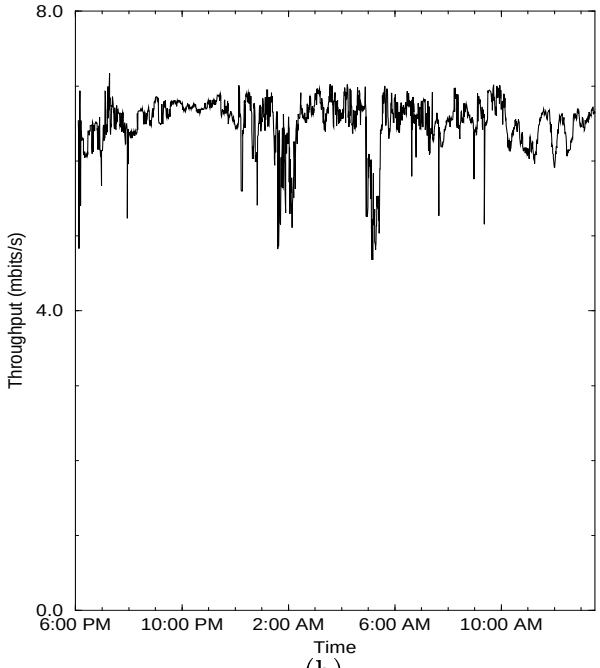
We ran both experiments on a dedicated SPARC 20 workstation and all time series data fit into real memory. The timing data in the table reflects the average time required to generate a forecast in milliseconds. In the case of SNP, however, it does not reflect the time required to fit a model. That is, these times reflect the difference between using the adaptive technique we have incorporated within the NWS (which requires no model-fitting phase) versus **evaluating** the model fit with SNP each time a forecast is required. Although generating a forecast is three orders of magnitude slower using the more complex SNP model, it still reasonable with respect to a 30 second measurement cycle. That is, if a forecast is generated every time a measurement is taken, waiting approximately 1 second for that forecast (assuming a 30 second cycle) is probably reasonable. Figures 9(b) and 9(c) show the predictions made by each method.

The SNP model we used to generate the predictions shown in Figure 9(c) took almost 21 hours of dedicated CPU time on the SPARC 20 to fit. We used a trace of the previous twenty-four hour period to fit a general non-linear model with heterogeneous innovations. While the delay associated with evaluating the SNP model (approximately 1 second per forecast on a Sparc 20) might be amortizable, a 21 hour model fitting time is not. Performance enhancements to the SNP code itself (e.g parallelization) and faster resources might reduce this cost, but as Table 4 shows, the improvement in accuracy gained by the more complex technique is small. The average forecasting error generated by the NWS dynamic method is 0.02 megabits per second greater that that generated with SNP. Since the values being forecast are on the order of 6 megabits per second, the small improvement in accuracy does not seem to warrant the much larger complexity associated with generating and then evaluating the SNP model. It may be, however, that for other measurement time series less complex SNP models will perform better. However, for process-to-process TCP/IP performance over an ethernet (which is relatively predictable — note the small MAE associated with each prediction) the dynamic NWS techniques work very nearly as well with much less computational complexity.

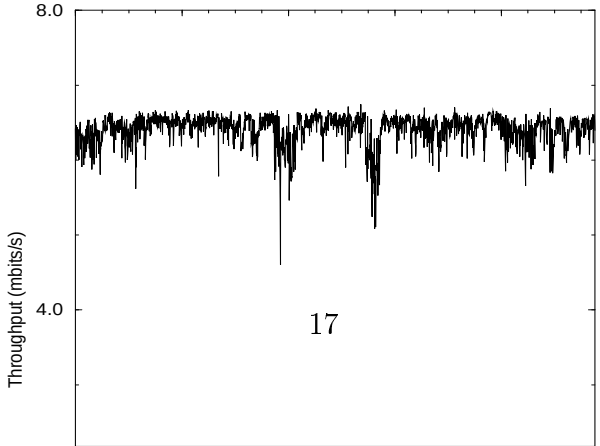
Clearly, much work is to be done to find the balance point between the complexity associated with dynamically generating a forecast and its accuracy. We are certain that there are many ways to streamline and improve the cost of model fitting with SNP, for example, some of which are outlined by its authors (see [9]). However, since the improvement in forecast accuracy is slight, we have chosen to incorporate a more dynamic and computationally cheap


8.0
4.0
0.0
Throughput (mbits/s)
6:00 PM
10:00 PM
2:00 AM
6:00 AM
10:00 AM
Time

(a)


8.0
4.0
0.0
Throughput (mbits/s)
6:00 PM
10:00 PM
2:00 AM
6:00 AM
10:00 AM
Time

(b)


8.0
4.0
Throughput (mbits/s)

methodology in the NWS, at present. Effectively incorporating off-line modeling techniques like SNP and other time-series systems in a manner which is cost effective (given the dynamic nature of metacomputing systems) is the subject of our present research.

4 Conclusions and Future Work

In a metacomputing environment, scheduling and quality-of-service mechanisms must have access to predictions of deliverable resource performance to mitigate the effects of contention. We have implemented a system called the Network Weather Service that collects periodic performance measurements and generates statistical forecasts, dynamically, based on time-series analysis techniques. The system is intended to be a ubiquitous service within a metacomputer, providing forecast information to all interested schedulers, quality-of-service facilities, and users.

To provide accurate performance forecasts, the measurements required to parameterize the forecasting models must be as non-intrusive as possible. Performance experiments that sense the available performance at any given time must not interfere with each other, or inaccurate readings will be incorporated into the generated forecasts. Furthermore, since forecast information may be used dynamically (i.e. to support dynamic scheduling), the interface to the system also must be lightweight and non-intrusive. These requirements motivate the design of the NWS architecture, and the implementations we have constructed for TCP/IP and Unix sockets, and the Globus/Nexus, and Legion metacomputing environments.

To make forecasts, the NWS automatically identifies and combines different predictive strategies from a set of potentially useful models. It chooses those models that, at any given time, have accumulated the lowest aggregate prediction error. Our experience indicates that this dynamic method of forecasting model selection works well. Moreover, using simple forecasting techniques like those outlined in Section 3 yields more accurate predictions than those generated from measurements of current conditions alone. Both forecast data and accuracy measures are made available to NWS clients so that the predictability of a resource may be considered. Further, while the forecasting techniques we describe are computationally simple, our early experiences with them indicate that they perform almost as well as those which are considerably more complex. The lightweight nature of the forecasting methods we have implemented makes them appropriate for dynamic computational settings.

Our future work will focus on developing new sensory mechanisms and new forecasting techniques. Predicting the time a job will wait in a batch queue is especially important in large-scale computational settings, for example. To make such predictions, we need to incorporate more sophisticated sensors and forecasting models. We are also planning to develop a scalable version of the system that can be deployed over large numbers of resources, based on a hierarchical organization of NWS resource clusters.

References

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

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

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

[4] M. Devarakonda and R. Iyer. Predictability of process resource usage: A measurement-based study on unix. **IEEE Transactions on Software Engineering**, (12), December 1989.

[5] S. Figueira and F. Berman. Modeling the effects of contention on the performance of heterogeneous applications. In **Proc. 5th IEEE Symp. on High Performance Distributed Computing**, August 1996.

[6] I. Foster, J. Geisler, C. Kesselman, and S. Tuecke. Managing multiple communication methods in high-performance networked computing systems. **Journal of Parallel and Distributed Computing**, 40:35–48, 1997.

[7] I. Foster and C. Kesselman. Globus: A metacomputing infrastructure toolkit. **International Journal of Supercomputer Applications**, 1997. to appear.

[8] I. Foster, C. Kesselman, and S. Tuecke. The nexus approach to integrating multithreading and communication. **Journal of Parallel and Distributed Computing**, 1997. to appear.

[9] 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`.

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

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

[12] A. Grimshaw. Easy-to-use object-oriented parallel programming with mentat. **IEEE Computer**, May 1993.

[13] 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.

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

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

[16] 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.

[17] R. Wolski. Dynamically forecasting network performance to support dynamic scheduling using the network weather service. In **Proc. 6th IEEE Symp. on High Performance Distributed Computing**, August 1997. to appear.