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

(Extended Abstract)

submitted to SC97

UCSD Technical Report TR-CS97-540

Rich Wolski
Neil Spring
Chris Peterson*

May 20, 1997

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 [11] 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.

1 Introduction

Fast networks have made it possible to connect distributed, heterogeneous computing resources to form a high performance computational platform, or *metacomputer*. While metacomputing offers tremendous potential performance, effective application scheduling is critical to realize that performance. Schedulers make decisions about which and when resources will be used by the application and as such, scheduling decisions must be based on *predictions* of the performance each resource will be able to deliver when it eventually performs the work it has been assigned. Since the metacomputer is a shared system, contention for system resources will cause the deliverable performance to vary dynamically. A scheduler for this environment requires predictions of resource performance that can account for variability caused by resource contention. We have designed and implemented a system that takes periodic measurements of the currently deliverable performance (in the presence of con-

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

tention) from each resource and uses numerical models to generate forecasts dynamically of future performance levels. Forecast data is continually updated and distributed so that dynamic schedulers and other run time mechanisms can use the latest predictions. 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).

The NWS measures and forecasts performance deliverable to the application level. As such, 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 for the Globus/Nexus [8] and Legion [11] 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 NWS functionality 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 [13].

In this paper, we focus on the architecture and implementation of the Legion[1] 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. In section 3, we present comparative forecasting results for both implementations and we conclude with an evaluation of the current system and a description of future research in section 4.

[1]We developed the implementation of the NWS for Legion that we describe in this paper using the prototype Legion environment based on Mentat [10].

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 [13]) require long-term history information. As such, we 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 mea-

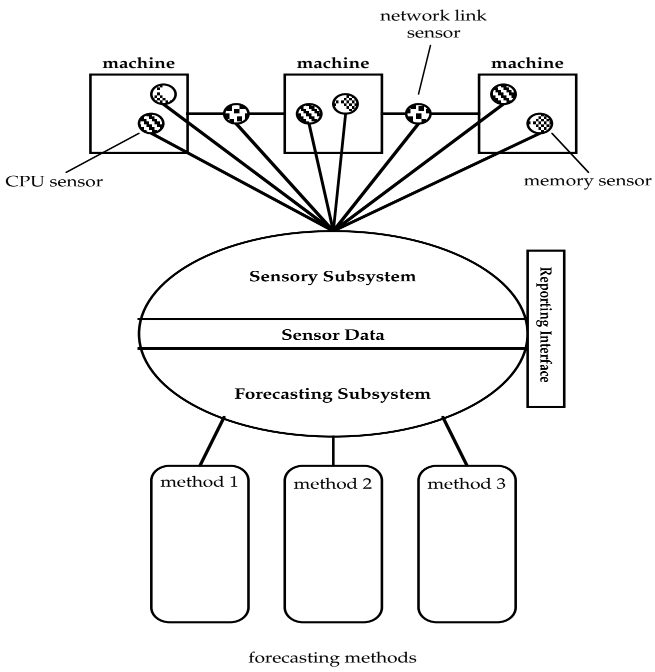


Figure 1: The Logical Structure of the Network Weather Service.

surement histories (which may be treated as time series) to generate predictions. The forecasting subsystem tracks prediction error and maintains accuracy information for each prediction. 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* and *uptime* on each machine and processing the output.

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*. For example, a sensor may access a resource in a representative way and monitor the elapsed time required to complete the access.

2.1.1 Sensing the Network

The NWS currently monitors both process-to-process latency and throughput throughout the system. 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 be determined automatically by the

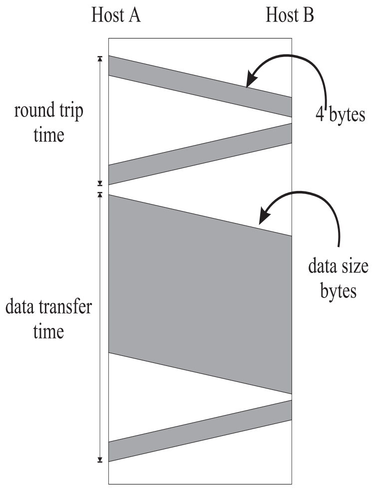


Figure 2: Throughput and Latency Experiments.

NWS itself. Throughput is calculated as

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

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

Note that simultaneous experiments over the same network may adversely affect both network performance and the quality of the measurements made. If performance experiments collide, the resulting measurements reflect the load introduced by contention between the experiments themselves. It is therefore necessary to employ a distributed method for ensuring mutual exclusion so that performance experiments do not interfere. In the current implementations, we use a token passing scheme to entitle a particular sensor to perform a network experiment. When a sensor receives a token, it is entitled to conduct a single network performance experiment and then to pass the token on to a successor. Token routing is configured statically into the system.

In a controlled, local area environment with well synchronized clocks, and well behaved networks, it would be possible to distribute network experiments based on a time-slicing approach, where each pair of hosts is assigned a particular fraction of a minute with which to perform a network experiment. However, even assuming that the added complexity of implementing distributed synchronized clocks and using these clocks to control network performance experiments avoided contention during normal operation, as soon as packets are dropped and the expected lifetime of an experiment is exceeded, contention results. By passing a token, however, the system is self-clocking and stable. 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 time-out. The token master can be switched dynamically between hosts should the master, itself, fail.

This method for ensuring non-interference between experiments does not scale. We are currently investigating ways in which a number of NWS instances may be organized hierarchically to provide scalability. 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.

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

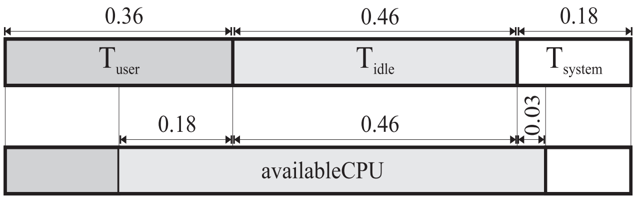


Figure 3: Available CPU calculation. rp = 2 runnable processes. T_{user} = 0.36, of which half (0.18) would be usable to a second process. T_{idle} = 0.46, all of which is available for computation. T_{system} = 0.18, of which 0.03 can be shared. The total $availableCPU = 0.67$.

Service computes the "percentage of the processor" 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

The resulting value can be used to 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 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 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 [12, 13] 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. In Section 3 we describe the forecasting methods that we consider in this paper more completely.

2.3 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. Each host maintains a copy of the current state of the network, which is exchanged during communication experiments to amortize the overhead of state exchange. The data stored in the host object on each host is shown in Figure 4. Storing a global image of the current and predicted state of the network on each host allows clients to access this data without the need to assemble a snapshot of the global state of the network by communicating with a set of hosts before making a scheduling decision. Time series history information is stored by individual host objects, 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 time series information to clients. Access to both remote and local data is provided through the communication primitives of the underlying resource management system.

Users can easily view NWS weather reports using the World Wide Web. The Network Weather Service on each machine continuously generates

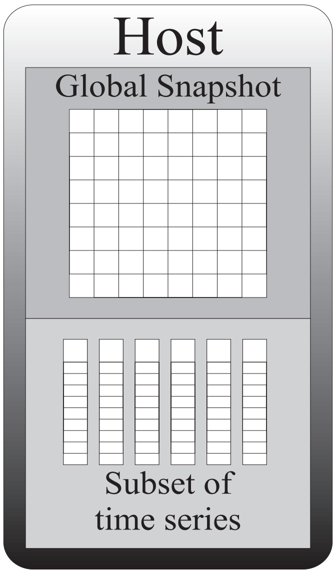


Figure 4: Data stored on each host.

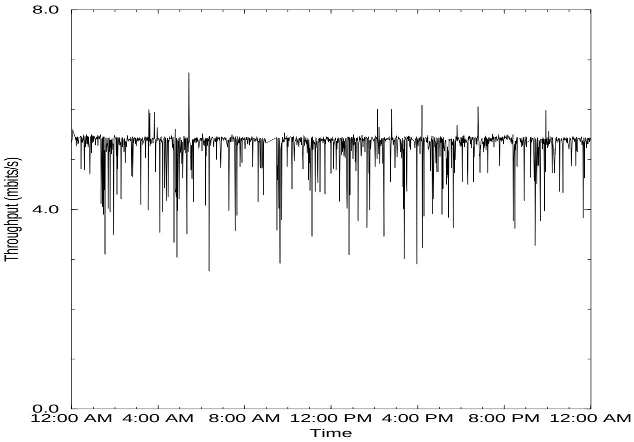
a publicly accessible HTML file containing a "snapshot" of the most recent forecasts. With their web browers, users can watch current and predicted network conditions fluctuate as the Network Weather Service monitors the network.

3 Results

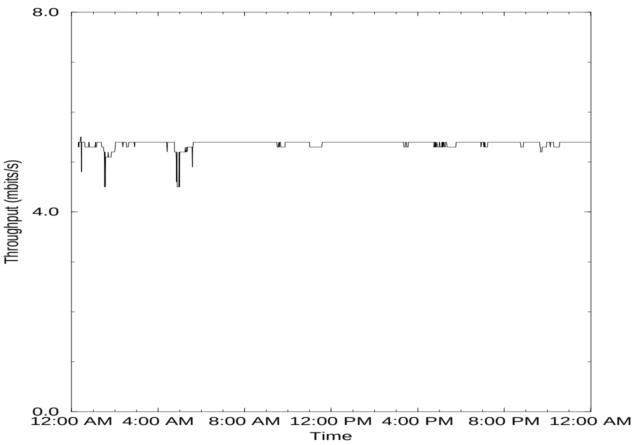
To forecast resource performance, the NWS treats periodic measurements taken from a particular sensor as a time series, and then uses different statistical models to predict the next value in the series. In this section, we report the performance of the forecasting system (in terms of prediction accuracy) for several different resources. We also 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.

3.1 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) at UCSD. Once every sixty seconds over a 24 hour period (starting at midnight on Tuesday, Februrary 4, 1997), the NWS moved a 64 kilobyte array via a Mentat[2] method invocation between the two hosts and timed the transfer. Figure 5(a) shows the time series of measurements and Figure 5(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.



(a)



(b)

Figure 5: Legion Throughput Measurements (a) and Predictions (b) in the PCL

[2]The Mentat Programming Language is an object-oriented language based on C++ supported by Legion [10].

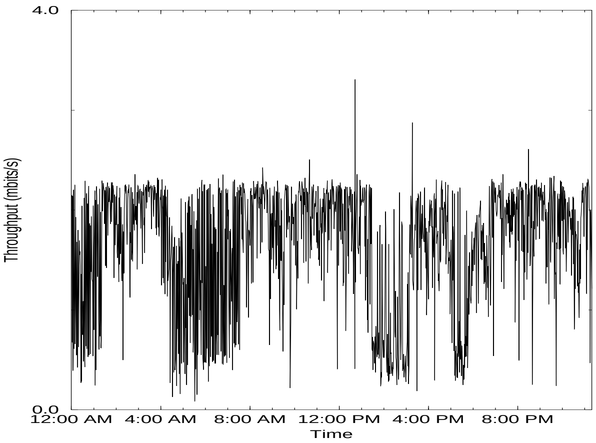
Predictor	MAE
$LAST$	0.28
RUN_AVG	0.23
SW_AVG	0.23
$MEDIAN$	0.18

Table 1: Forecasting Error for PCL Throughput using Legion

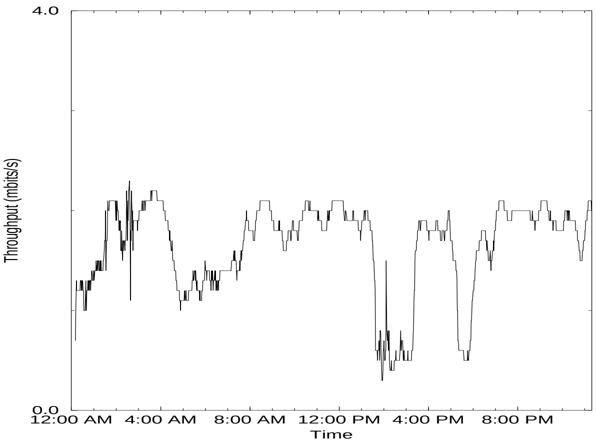
That is, during the measurement period, at each time step the prediction of a measurement differed from the actual measurement it was predicting by an average of 0.18 megabits per second. Table 1 summarizes the predictive performance of several different forecasting methods. The NWS currently supports a variety of forecasting techniques, the details of which are more completely described in [12, 13]. In the interest of brevity, we demonstrate its functionality using only forecasters that are based on common summary statistics. 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 a prediction error that is an average of 0.1 greater than that generated by $MEDIAN$. In general, our experience has been that the last measurement is a poor predictor of future network performance. Often, performance monitoring systems use current measurement data as an estimate of the available throughput. In this case, the use of the current measurement yields a prediction that is 55% less accurate than the one chosen by the NWS. Clearly, even for a predictable network like the one shown in Figure 5, simple probes measuring current conditions are poor indicators of the performance that will be available in the next time step.

In Figure 6 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



(a)



(b)

Figure 6: Nexus Throughput Measurements (a) and Predictions (b) from the PCL to SDSC

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 6(b) shows the pre-

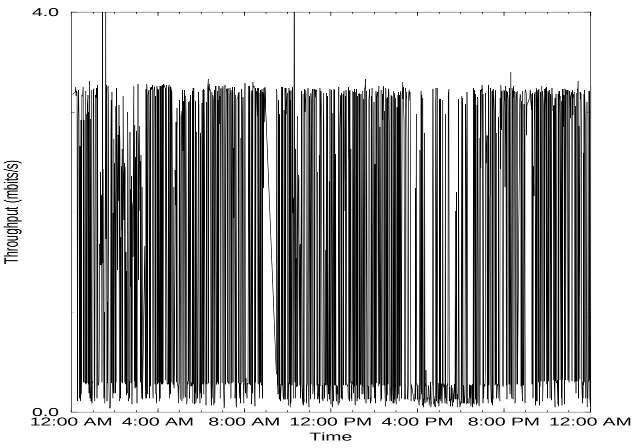
Using Nexus	
Predictor	**MAE**
LAST	0.46
RUN_AVG	0.49
SW_AVG	0.39
MEDIAN	0.38

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

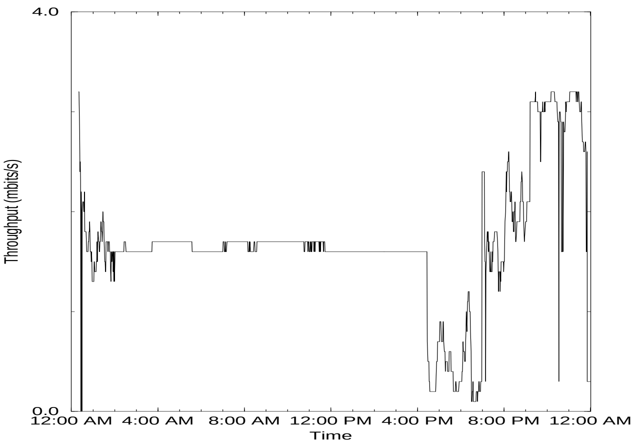
Table 2: Forecasting Error for PCL to SDSC Throughput

dictions 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 (21%) lower than $LAST$ and 0.11 (29%) lower than RUN_AVG. The forecasts shown in Figure 6(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.

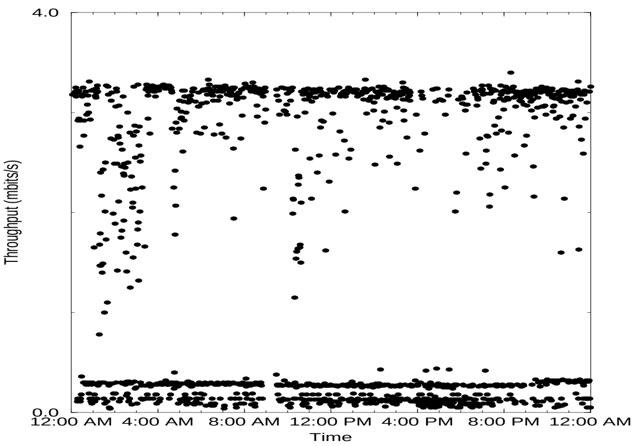
In Figure 7 we show throughput measurements and prediction for the same PCL-to-SDSC communication link using Legion. Compared to the equivalent set of Nexus measurements shown in Figure 6a, the Legion measurements vary through a wider range. A scatter plot of the Legion measurement data (shown in Figure 7c) 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/Mentat prototype implementation[3] 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 [9] nor a state-transition analysis [4] yield exploitable predictive



(a)



(b)



(c)

Figure 7: Legion Throughput Measurements (a), and Predictions (b), and a Scatter Plot of the Measurements (c) from the PCL to SDSC

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

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.

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 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 with the Legion and Globus/Nexus implementations 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.

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

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

[11] 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.

[12] 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.

[13] 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.