Predicting Grid Resource Performance On-line

Rich Wolski[1], Graziano Obertelli[1], Matthew Allen[1], Daniel Nurmi[1], and John Brevik[1]

Computer Science Department University of California, Santa Barbara

Summary. In this paper, we describe methods for predicting the performance of Computational Grid resources (machines, networks, storage systems, etc.) using computationally inexpensive statistical techniques. The predictions generated in this manner are intended to support adaptive application scheduling in Grid settings, and on-line fault detection. We describe a mixture-of-experts approach to non-parametric, univariate time-series forecasting, and detail the effectiveness of the approach using example data gathered from "production" (i.e. non-experimental) Computational Grid installations. See footnote[1] for information on sources of support for this work.

1 Introduction

Performance prediction and evaluation are both critical components of the Computational Grid [20, 8] architectural paradigm. In particular, predictions (especially those made at run time) of available resource performance levels can be used to implement effective application scheduling [13, 38, 42, 12, 43, 9]. Because Grid resources (the computers, networks, and storage systems that make up a Grid) differ widely in the performance they can deliver to any given application, and because performance fluctuates dynamically due to contention by competing applications, schedulers (human or automatic) must be able to predict the deliverable performance that an application will be able to obtain when it eventually runs. Based on these predictions, the scheduler can choose the combination of resources from the available resource pool that is expected to maximize performance for the application.

[1]This work was supported, in large part, by grants from the National Science Foundation, numbered CAREER-0093166, EIA-9975020, ANI-0213911, and ACI-9701333. In addition, the infrastructure development for public release that is discussed has been supported by the NSF National Partnership for Advanced Computational Infrastructure (NPACI) and the NASA Information Power Grid project.

Making the performance predictions that are necessary to support scheduling typically requires a compositional model of application behavior which can be parameterized dynamically with resource information. For example, consider the problem of selecting the machine from a Grid resource pool that delivers the fastest execution time for a sequential program. To choose amongst a number of available target platforms, the scheduler must predict the execution speed of the application code on each of the platforms. Grid infrastructures such as Globus [19, 15] provide resource catalogs in which static, and therefore precisely known, attributes (such as CPU clock speed) are recorded. Therefore the simplest approach to selecting the best machine is simply to query the catalog for all available hosts and then to choose the one with the fastest clock rate.

There are several assumptions that underlie this simple example. One assumption is that the clock speeds of the various available CPUs can be used to rank the eventual execution speeds of the program. Clock speed correlates well with execution performance if the machine pool is relatively homogeneous. One of the basic tenets of the Grid paradigm, however, is that a wide variety of resource types are available. If, in this example, a floating point vector processor is available, and the application vectorizes well, a slower-clocked vector CPU could outperform a faster general purpose machine making clock-speed an inaccurate predictor of application performance. Conversely, if it is a scalar integer code, a high-clock-rate vector machine might under-perform a slower commodity processor.

A second assumption is that the CPU is the only resource that needs to be considered as a parameter to the application model. If the input and output requirements for the program are substantial, the cost of reading the inputs and generating the outputs must also be considered. Generating estimates of the time required for the application to perform I/O is particularly difficult in Grid settings since the I/O usually traverses a network. While static CPU attributes (e.g. clock speed) are typically recorded for Grid resources, network attributes and topology are not. Moreover, at the application level, the network performance estimates that are required are end-to-end. While it is possible to record the performance characteristics of various network components, composing those characteristics into a general end-to-end performance model has proved challenging [36, 16, 53, 17, 6, 30, 37].

However, even if a model that effectively composes application performance from resource performance characteristics is available, the Grid resource pool cannot be assumed to be static. One of the key differentiating characteristics of Computational Grid computing is that the available resource pool can fluctuate dynamically. Resources are *federated* to the Grid by their resource owners who maintain ultimate local control. As such, resource owners may reclaim their resources, may upgrade or change the type and quantity of resource that is available, etc. making "static" resource characteristics (e.g. the amount of memory supported by a machine) potentially time-varying.

Even if resource availability is slowly changing, resource contention can cause the performance that can be delivered to any single application component to fluctuate much more rapidly. CPUs shared among several executing processes deliver only a fraction of their total capability to any one process. Network performance response is particularly dynamic. Most Grid systems, even if they use batch queues to provide unshared, dedicated CPU access to each application rely on shared networks for inter-machine communication. The end-to-end network latency and throughput performance response can exhibit large variability, both in local area and wide area network settings. As such, the resource performance that will be available to the program (the fraction of each CPU's time slices, the network latency and throughput, the available memory) must be *predicted* for the time frame that the program will eventually execute.

Thus, to make a decision about where to run a sequential program given a pool of available machines from which to choose, a scheduler requires

- a performance model that correctly predicts (or ranks) execution performance when parameterized with resource performance characteristics, and
- a method for estimating what the resource performance characteristics of the resources *will be* when the program executes.

In this chapter, we focus on techniques and a system for meeting the latter requirement. In particular, we discuss our experiences in building and deploying the *Network Weather Service* (NWS) [52, 49, 50, 35] — a robust and scalable distributed system that monitors and predicts resource performance on-line. The predictions the NWS makes are based on real-time statistical analysis of historically observed performance measurement data. Typically deployed as a Grid middleware service, the system has been used been used extensively [38, 12, 48, 3, 41, 51, 43, 9] to provide resource performance forecasts to Grid schedulers. We describe the architecture of the NWS, the statistical techniques that have proved successful from our collaborations with various Grid scheduling projects, and some of the lessons we have learned from building and deploying a Grid information system capable of managing dynamic data in real-time.

2 Requirements for Grid Performance Monitoring and Forecasting

As a Grid service, the NWS (and any other system that serves dynamically changing performance data) must fulfill a demanding set of requirements. The system must be able to run continuously so that it can gather a history of available performance from each monitored resource. At the same time, the fluctuations in performance and availability that it is tracking cannot impede its function. Network failures, for example, cannot cause NWS outages even

though the NWS may be using those network links that have failed to gather and serve performance data.

The performance monitoring system must also avoid introducing false correlations between measurements. For example, the typical method for measuring host availability is to use some form of "heartbeat" message to renew a soft-state availability registration [21]. Hosts send a message periodically to a central server to indicate their availability and missing heartbeats indicate host failure. While this architecture method is robust if the central server is running on a highly available system, it inextricably convolves network failure and host failure. That is, a missing heartbeat or set of heartbeats could be because the host has failed, or because the network linking the host to the central server has failed. For hosts within a cluster, the problem is especially acute. If the network partitions between a cluster and the soft-state registration server, the cluster hosts will appear to have failed when, in fact, they can communicate with each other and with any hosts on the same side of the partition.

Of all Grid services, the ones with the most restrictive performance requirements are precisely those that monitor performance. If client applications and services are to use the performance data served by the performance system, in some sense the system must run "faster" than these clients so that the needed data is immediately available. If it is not, clients may waste more time waiting for performance data from the resources they intend to use than they will gain from having the performance data in the first place. That is, the data must be gathered and served in time for it to be useful. Few other Grid services must operate under such restrictive performance deadlines.

Moreover, the standard technological approaches that have been developed for serving data across a network typically are not optimized to handle dynamically changing data. Most extant systems are designed under the assumption that the rate of queries for the data is substantially higher than the rate at which the data changes. For static resource attributes such as processor type, operating system and revision level, static memory capacity, etc. this assumption is reasonable. As an example, queries for operating system type and revision level (which are critical to support for automatic resource discovery) should occur at a higher rate than the administrative OS upgrade frequency in any reasonable setting. However, when historical resource performance is to be used to gauge resource suitability, particularly with respect to load and availability, the opposite data access pattern is typical. Resources update the information base with periodic performance measurements much more frequently than queries are made. Thus query-optimized systems, if not architected to support more frequent updates than queries, may have trouble coping with the update load introduced by the need to constantly gather performance measurements.

The need to monitor Grid resources constantly without perturbing those resources requires the monitoring system to be ubiquitous yet mostly invisible to users and administrators. Further, a resource monitoring process that is

has a noticeable impact on running applications will not and should not be tolerated. These issues imply a monitoring system that is powerful enough to provide useful information and yet remain lightweight enough to not have significant impact on resource performance.

Finally, the Grid performance information system must be able to meet the daunting engineering challenges described in this section at a relatively large scale. While the debate about the feasibility of Internet-wide Grid computing continues, at present Grid systems containing tens of thousands of hosts generating millions of individual performance histories are being deployed. To be effective, Grid performance monitoring systems must be able to operate at least on this scale, in the wide area, while respecting the constraints placed upon resource usage by each resource owner.

3 The Network Weather Service Architecture

It is, perhaps, easiest to think of the NWS as a Grid application designed to measure performance and service availability. Resource sensors must be deployed and executed on a large, heterogeneous, and distributed set of resources with widely varying levels of responsiveness and availability. Due to the volatile nature of Grid environments, the NWS is necessarily designed to be a portable and scalable with functional mechanisms for load balancing, redundancy, and failure handling. In this section, we describe the individual components of the NWS as well as the mechanisms which have enabled it to be successfully deployed on Grid architectures around the world and to be compatible with or work within the most common Grid infrastructure (Condor [46], Globus [19, 15], GrADS [7, 23], etc.).

The NWS is composed of three persistent components and a suite of user interface tools. The set of persistent entities that compose a minimal NWS installation includes one of each of the following: *nameserver*, *memory* and *sensor*. NWS installations typically include many sensor components, one on each machine that is to be monitored. An installation also includes one or more memory processes, depending on the scale of the installation, and a single nameserver. Each sensor process is responsible for gathering resource information, which is then stored over the network to a memory, the location of which is registered in the nameserver along with other system control information. The relationship between these components is shown in *figure 1*, and will be more thoroughly explained in the following subsections.

In addition to these persistent components, NWS installations include interface tools which allow users to search, extract, and request forecasts of measurement data. Tools also exist which allow an NWS administrator to control the running state of the entire installation from a single point on the network. These tools are covered in depth at the end of this section.

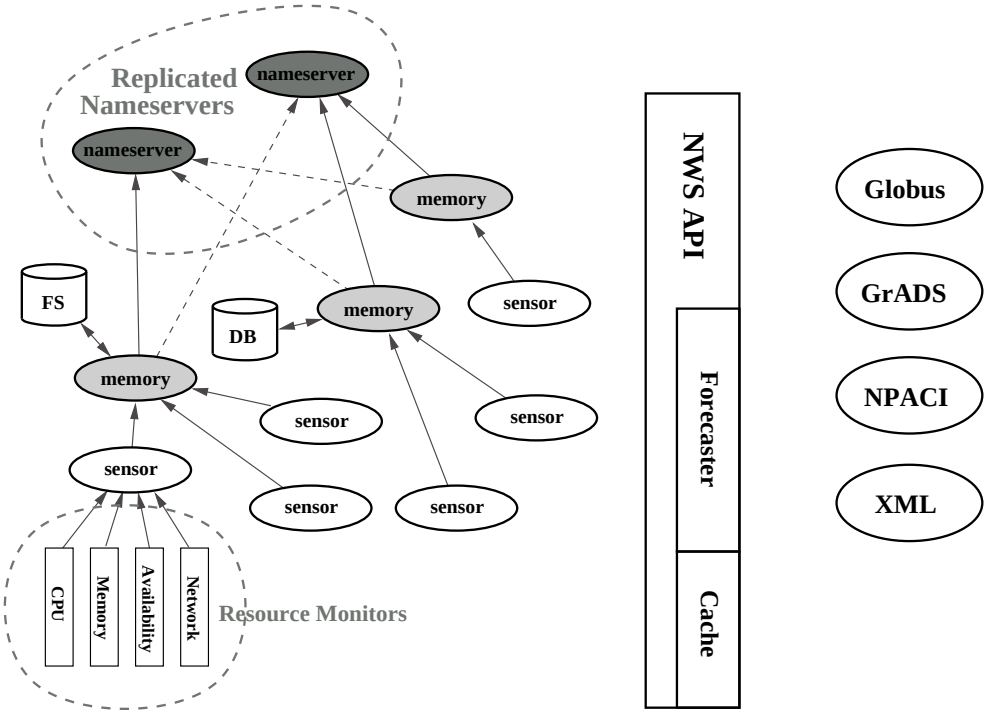

Fig. 1. Overview of an example NWS installation.

3.1 Nameserver

The nameserver can be considered the phone book of the NWS–it keeps a record of every host and activity in the system. As a single source of control and information, NWS users and other NWS entities use the nameserver and stored registrations to perform lookup, search, store, and many other control tasks. Essentially, if an activity, host, or process exists in an NWS installation, information about it can be gleaned from the data stored in the nameserver.

Although there can be multiple nameserver processes there is only one logical nameserver for each NWS installation. All other NWS components such as memories and sensors are in periodic contact with the nameserver to keep the view of the system current and controllable.

To combat the inevitable down time of a nameserver or machine on which a nameserver is running, a robust failover mechanism has been built into the system. In order to provide robustness, the nameserver is architected with mirroring capabilities: two or more nameservers, on separate hosts, can be directed to keep their data synchronized. In this case every update request coming to one nameserver will be forwarded to all others. The nameservers also implement a queue of update requests to tolerate temporary network

failures. Nameservers that fail permanently are removed from the mirroring process.

The primary datum kept by the nameserver is called a *registration*. Each registration is a set of flexible key/value attribute pairs, with only a few keys required to construct a valid registration. The required keypairs in every registration are *name*, *objectclass*, *timestamp* and *expiration*. The former two are used to describe the type of registration and are provided by the registering host, and the latter two are used for management and are added by the nameserver upon receipt. The nameserver offers fast search capabilities and updates on the registrations by keeping them ordered in memory and periodically saving a backup to stable storage. Apart from the required keypairs, NWS hosts are free to add new attributes containing whatever control information they require to operate.

Of the required keypairs, *objectclass* is the highest level and the only keypair that define the content of the registration itself. Currently, objectclass supports the following values and additional information:

- **nwsHost**: every host registers itself with the nameserver. *hostType* indicates whether the NWS host is a memory, nameserver of sensor, *ipAddress* is the ip address of the nwsHost as reported by gethostbyname or forced from the command line, *port* is the TCP port the host is listening on, *started* is the time when the host was executed, *owner* is the login name of the user that started the host, *version* is the NWS version, and *flags* are the options passed to configure upon NWS compilation flags. Other keypairs reflect specific host details (*systemType, releaseName, machineArch, CPUCount,* etc.).
- **nwsSkill**: every NWS sensor registers a list of its capabilities (called *skills*). It contains the *skillName*, the *option* that can be used when starting an activity, and an informative list of what the options take as arguments (integer, string, ...).
- **nwsControl**: currently there are two different controls defined by the *controlName*: *periodic* and *clique*. This objectclass also defines the *host* bound to the control, the *option* that is passed to the control, and the *skillName* that can be started under this control.
- **nwsActivity**: experiments that are being run in the NWS system are all registered with the nameserver. Objects of this type contain the *controlName* that started the activity, the *host* running the activity, the *skillName* used for this activity, and the *option* that the skill uses.
- **nwsSeries**: collections of measurements are called *series*. Objects of this type contain the *host* which ran the experiments, the *activity* which is generating the series, the measured *resource*, the NWS *memory* which stores the series, the measuring unit for this resource (*label*) and the *option* used for this skill.

The nameserver's responsibility is to store small, independent data items and make them available to users. As a result, it is optimized to make searching

and correlating data quick and simple. However, this design is not conducive to storing large sets of data like measurement series. This data is stored by another process that is designed to deal with the information's specific nature. This component, called the memory, is described in the next section.

3.2 Memory

The memory server is responsible for housing measurement data produced by sensors within an NWS installation. The memory receives measurements from sensors and other sources and organizes them into a collection called a *series*. It makes these series available to users through a well defined interface.

Memories are a very flexible part of the NWS infrastructure and can be used in whatever way is appropriate to the scale of the installation. Users interested in minimizing the network traffic used to save measurement data can create a memory on each machine or administrative domain housing sensors. Alternatively, to reduce the cost of retrieving data from a single source, a memory can be place on a nearby central host capable of handling a large number of sensors and measurement series.

The memory registers every series that it is responsible for with the nameserver. In the case of the failure of a replicated nameserver, the memory knows how to contact and utilize backup nameservers. Without the presence of any functional nameserver, it can operate independently–storing measurement data and series registrations from newly started sensors. If sensors establish new series with a memory while the nameserver is inaccessible, the memory caches their registrations and forwards them when the nameserver becomes available again.

Upon restart a memory checks if there are older series in stable storage. If any exist, it creates a limited registration and sends it to the nameserver. This mechanism allows the system to access series that are no longer updated by active sensors but are still addressable by the memory.

By default, memories store measurement data using the file system. Each series is associated with a file named with the fully qualified series name. These files are managed as circular queues with a size determined by a user parameter. The first line of the file contains data for managing the circular queue. Each series measurement is stored in a fixed length, human readable buffer containing the arrival timestamp, sequence number, expiration timeout, and the timestamp/measurement pair sent by the sensor. As the circular queue becomes full, old values are overwritten.

If data is stored in the file system in this way, the memory keeps a cache of the most frequently accessed series in resident memory to minimize the performance hit of going to the file system. To keep update operations safe, the cache is write-through. Although the cache reduces IO load and increases performance for read operations, the fact that it does not cache write operations results in substantial IO overhead from writes being performed on disk files.

Larger installations typically exploit the feature of NWS memories which, allowing multiple memory instances within an installation, significantly reduces IO load on any one host running a memory process.

While memories usually store a large enough backlog data to make accurate resource forecasts, some applications require a longer trace of data. In these cases memories can use a database instead of a circular queue filled flat file. In the database, a new table is generated for each series the memory is handling. Each set of measurements is stored in the table with the same information as the flat file design. Data is stored for as long as the database administrator decides to keep the history.

When data is requested from an NWS installation, the memory process is responsible for providing it. The memory makes no effort to interpret user requests, so users usually talk to the nameserver to discover the name of a series and the memory that houses it. The primary source of data for the memory is the sensor process, which is responsible for running on-line performance tests. This process is described in the next section.

3.3 Sensor

The NWS sensor component is responsible for gathering resource information from machines, coordinating low level monitoring activities, and reporting measurements over the network to an NWS memory.

On each machine that houses monitored resources, a single sensor process is deployed. Since single machines house multiple resources, each sensor process has the capability of spawning child processes for measuring each unique resource. Sensors are typically measuring resources available to normal users, so the NWS sensor should be executed using normal user permissions. Running sensors with system privilege is, in fact, discouraged. Starting them can be done manually, through automated execution systems (cron, etc), or at system startup.

To account for unforeseen complications that may cause various resource measurement processes to fail or block, the sensor separates its administrative and measurement components into separate processes. The original parent process is responsible for accepting control messages and starting measurements while child processes are created to perform the actual measurements. If this is not desirable, this feature can be disabled, leaving only one process to handle both measurements and control messages.

If the network between the sensor and memory fails or the memory process becomes temporarily unavailable, the sensor process will begin caching resource measurements until such time when the memory becomes available. In this way, the sensor is capable of maintaining a consistent view of measured resources without gaps incurred by network or process failures.

The introduction of firewalls often adversely affect distributed systems. NWS sensors can be instructed to use a specified port when conducting network experiments, allowing an administrator to open only 2 ports in the fire-

wall: one to control the sensor and the other to allow the sensors access to one another while taking network measurements.

A sensor is instructed to start monitoring a resource using a specific skill with some well specified options. An activity is the process of using a skill at specific interval. An activity generates one or more series measurement, and a single sensor is capable of running any number of activities. The current NWS sensor implementation includes the following pre-defined skills [2]:

- **availabilityMonitor**: measures time since machine last booted.
- **cpuMonitor**: measures the fraction of the CPU available and the current CPU load. Accepts a *nice* level as options.
- **diskMonitor**: measures available disk capacity of a specified disk. Accepts a *path* as option.
- **filesystemMonitor**: monitors performance of specified a file system. Accepts multiple options including *path*, *fstype* (block/char), *fstmpdir*, *fssize* and *fsbufmode* (instruct skill to attempt to avoid file system buffer cache using various methods).
- **startMonitor**: registers the numbers of seconds since the sensor started.
- **tcpMessageMonitor**: monitors bandwidthTcp and latencyTcp to a *target* host. It accepts options to set the *buffer* size of the socket, the *message* size to be used and the total experiment *size*.
- **tcpConnectMonitor**: measures the time it takes to establish a TCP connection with a *target* host.
- **memorySpeedMonitor** (experimental): measures attainable memory speed (random or sequential access).

In addition to pre-defined skills, the sensor has been architected to make the addition of novel user defined skills fairly straightforward. A user who wishes to add a new skill needs only to implement a function for measuring a resource of interest, and can rely on existing mechanisms for caching, communication, and control making the process of adding a new skill as simple and efficient as possible.

Many resources, like CPU, memory, etc, are measured on a single machine. Other resources, in particular network resources, require two hosts to participate in the experiment. Because the NWS uses active network probes, simultaneous test could interfere with each other. To deal with these different types of measurements, the NWS uses two methods to determine when measurements will be taken.

Periodic skills

Periodic skills need to be run at specific time interval and are independent (thus running these skills on different hosts at the same time doesn't cause interference in the measurement). Upon starting such skills, the *period* option

[2] Due to system limitations, not all skills are available on all architectures

is used to determine how many seconds pass between experiments. Most predefined skills are periodic skills, since measuring CPU, memory, disk and other independent resources has no effect on other hosts measuring the same resources.

Clique skills

NWS cliques are used to provide a level of mutual exclusion within a group of hosts so that their measurement activities do not interfere with each other. This is a best-effort mutual exclusion mechanism. Upon the start of a clique activity a token is generated and circulated within the members of the clique. A member can take measurements only if it has the token. Once the member has finished taking all the needed measurements, the token is passed to the next clique member.

Because the network can partition or hosts can fail, the token can get lost. To account for this, the clique protocol implements a mechanism to regenerate the token if knowledge of it is lost. Every clique has a *leader* (by default, the member which starts the token) which keeps track of the time needed to circulate the token. If the leader doesn't receive the token within a reasonable length of time, it regenerates the token and starts a new circulation. Also, if a member of the clique sees a long enough delay between tokens, it becomes the leader and starts a new instance of the token. The clique is timed out after a few multiples of the clique periodicity.

The token system is *best effort* because it considers taking measurements at the right frequency over strict mutual exclusion. The clique protocol ensures that the sensors take their network measurements at roughly the periodicity asked by the activity. Members can starts taking measurements without holding the token if too much time has elapsed. If the token is then received after the sensor's timeout, the token is passed along without taking the measurements. Mechanisms are in place to eliminate multiple tokens circulating at the same time (for example when a network partition is restored).

3.4 Design considerations

The NWS is expected to provide access to useful data for a large set of heterogeneous and faulty systems. As a result, it is required to be robust, portable, and scalable. Furthermore, sensor processes are run on the machines they are monitoring. If they have high resource requirements, they are likely to degrade application performance and to interfere with their own measurements.

Failure is a complicating factor in the design of NWS processes since they cannot disregard their responsibilities because a processes they report to is unavailable. Passive failure detection is accomplished using heartbeat messages between dependent processes. Heartbeats are used to detect expired registrations and failures in replicated nameservers. Also, the NWS relies heavily on timeouts to aggressively avoid deadlock during communication among NWS

processes. Components measure the length of time to send data and receive heartbeats for each host they interact with. By using forecasting techniques (described in section 4), the processes use these measurements as a timeseries to compute a perdition and error value. These two pieces of information are combined to form an expected upper bound. These bounds are used to time-out network communication, determine lost clique tokens, and notice which processes have not sent a heartbeat message.

The NWS has a number of mechanisms, detailed in the sections describing each component, for handling the failure of the processes they depend on. First, nameserver replication adds some robustness to the NWS's central point of failure. Additionally, memories and sensors all cache registrations that could not be sent to the nameserver. This means that these processes can be started even when the nameserver has failed, and they can also accommodate temporary nameserver failures. Lastly, sensors cache measurement data so that measurement are not lost when memories fail. These caches can hold a large number measurements, and can store almost an hour of CPU availability before they start to lose information.

There are a handful of portability issues that have been addressed for the NWS as well. For one, timing out socket communication is C is not a trivial task. Early versions used alarm signals to interrupt blocking communication system calls. This method is not portable for all OSes and does not interact well with threaded processes. Therefore, the NWS can be configured at compile time to use non-blocking sockets, disabling the use of the alarm signal. Other portability issues come from the use of threads, which are notoriously different across architectures and OSes. Therefore, forking is used in places where threads might be used. To allow users to implement processes that use the NWS within threads, the NWS libraries can be built with an option to add mutexes to synchronize internal calls.

Monitoring the network performance of a set of hosts, requires to take $O(n^2)$ measurements, obviously posing scalability concerns at some level. Observing that most likely there are clusters of machines tightly connected (fast local networks) which are as a group connected with wide area networks, we make the assumption that the statistical properties of the link from machines in a cluster to machines of another cluster are somewhat similar. Hence we do not require all individual measurements from all machines within separate clusters, but can instead elect one (or few) machines from each cluster and start a *superclique* among these selected machines. Newer versions of NWS provide a *caching* mechanism that understands this operation and provides a logical view of an all to all performance matrix of TCP network measurements. This caching mechanism can be seen in *figure 2*.

The NWS cache provides another scalability feature. Accessing $O(n^2)$ series requires a user to contact the memory $O(n^2)$ times, thus increasing the time of when the data is effectively available to unacceptable levels. To address this problem, we have made the assumption that what is really needed is the single prediction instead of the entire history. The cache works as proxy, col-

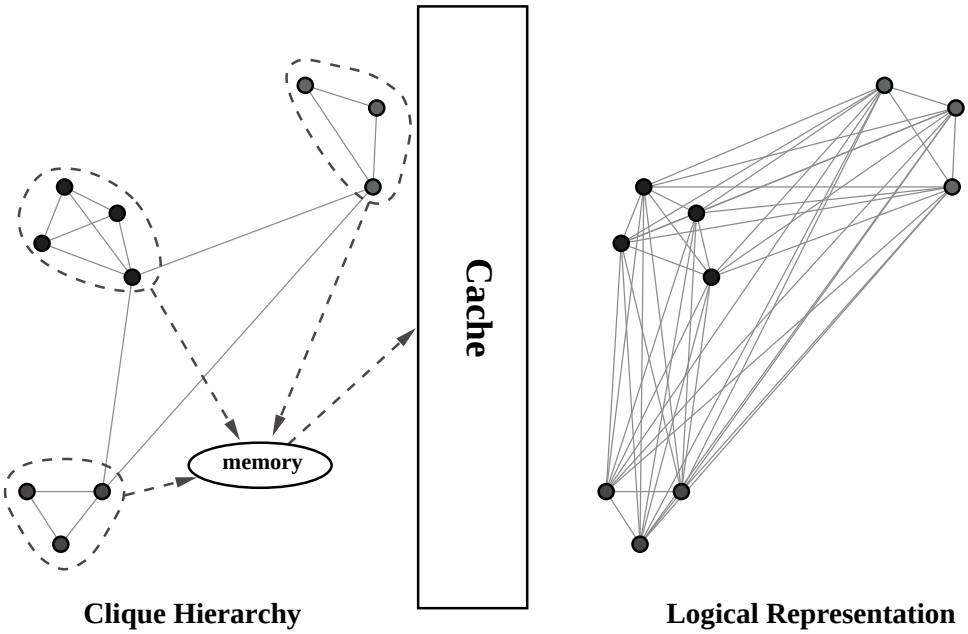

Fig. 2. Hierarchical clique with connectivity cache reduces both the number of experiments taken and connectivity graph creation when cliques contain a large number of hosts separated by wide area networks.

lecting the data from the memory and generating the forecasts, then returns the $O(n^2)$ forecasts in one call, avoiding the prohibitive $O(n^2)$ connection cost.

Finally, the components themselves are designed to be able to scale to a large number of hosts. The biggest liability is the nameserver, since it is the most centralized component. The requirements on the nameserver, however, are extremely low, so this poses little problem. The worst observed example was a nameserver running on a common desktop linux machine that served more than 50,000 registrations and hundreds of hosts. In the case that nameservers are frequently accessed, they can be replicated so different hosts and users can depend on different nameservers. The requirements of memories are not as light as nameservers, so they cannot serve nearly as much information. However, memories are very flexible about where they can be placed, so large systems can easily support a large number of memories.

Sensors have been designed to be lightweight and as non intrusive as possible on the host being measured. Only under excessive monitoring, due to misconfiguration, slower or less robust systems may be taxed (when the periods of large cliques, file system sensors, and CPU sensors excessively short). Typically, sensors uses between 2-4 megabytes of system memory depending on the number of experiments they run. They spend most of their time waiting for control messages without using the host's processor.

3.5 User interface

There are three main interface applications that are used to interact with an NWS installation: *nws_search*, *nws_extract* and *nws_ctrl*. These provide the core command-line interface with the NWS processes.

The nws_search program allows a user to search through the registrations that a nameserver has available. It uses a syntax reminiscent of LDIF and all the usual operators can be used ($\&, |, =, >=, <= \ldots$). For convenience, shortcuts have been added that allow users to list standard things like sensors, series, or skills without knowing the registration structure.

The nws_extract program allows the user to retrieve measurement data (series) from an NWS installation. The user specifies the nameserver, the resource they are interested in, and the hosts whose data they want to retrieve. nws_extract will first query the nameserver to find which series name matches the user request, then lookup in the matched object and the contact information of the associated memory, which is then contacted for data retrieval. The series of data is then fed to the forecaster and the measurements, forecasts, and respective errors are then presented to the user. If the nameserver is unknown, nws_extract can query the first sensor asking to report which nameserver it is using. If the user knows the series names and the memory storing them, using -M and -S they can bypass the nameserver and query the memory directly.

Finally, nws_ctrl allows the user to control processes in an NWS installation. Most importantly, it allows administrators to modify behavior which is usually specified through command line options at start time. However, there are also a handful of other commands. The following actions are understood by nws_ctrl:

- **test**: performs a simple test of health of an nwsHost. The nwsHost can be *dead* (no connection was made), *unresponsive* (connection was made but there was no response from the sensor), *sick* (the sensor is reachable but it cannot talk to its nameserver), or *healthy* (everything is functioning as expected).
- **register**: instructs the nwsHost to use a different nameserver for registration of objects. This allows the administrator to replace, restart, or move the nameserver process without redeploying the entire NWS installation. If the command is given to a nameserver, it will begin mirroring with the target.
- **memory**: instructs the given sensor to send all new measurements to a different NWS memory.
- **halt**: stops the nwsHost.
- **log** toggles the verbosity of logging on the specified nwsHost.
- **skill**: asks a sensor to run a particular skill with specified options. Unlike an activity, the results are not taken continuously or sent to the memory but are instead reported directly to the user at their terminal.

- **add/remove**: adds or removes a member from a currently running clique. The user needs to specify a member of the clique and the clique name, and the sensor will restart the modified clique with the same options but a different list of members.
- **ping**: runs a single network experiment (tcpMessageMonitor) between the host running the command and the remote sensor, reporting the results directly to the user.
- **start/stop**: asks a sensor to start or stop an activity.

All these processes make use of the well defined NWS-API to retrieve information and change the behavior of NWS processes. These functions are available to users through the *nws_api* library. It is therefore possible to include the functionality of these programs into a user's application with relative ease. In fact, these programs are invaluable examples of how to interface with the NWS at an application level.

4 The NWS Forecasting Methodology

The forecasting methodology used by the NWS assumes that each resource performance characteristic can be measured quantitatively. Each resource can be described by a stream of performance measurements, and predictions of future measurement values are the quantities that are of interest. Notice that useful qualitative information may be difficult to incorporate under this assumption. For example, it may be possible to know that "less" bandwidth will be available to a desktop machine which is typically used by a person who frequently downloads images from the Internet than from a machine used by a person who typically works locally. The NWS approach is to gather performance measurements from both machines and then predict future measurement values so that the predictions can be compared quantitatively. For some Grid applications, simply knowing "less" or "more" resource will be available may be enough to develop an effective schedule. The advantage of using quantifiable resource characterization, however, is that the information is more easily encoded for use by an automatic scheduler. That is, it may be difficult for a scheduling agent to parse and compare the qualities of a resource, but forecast quantities can almost always be compared if the units are compatible.

A second important assumption made by the NWS forecasting method is that performance measurements can be gathered non-intrusively. In particular, any load that the performance monitors introduce does not have a measurable effect on the resource being monitored.

Finally, because the methods are time series based, they assume that the characteristics being measured have an instantaneous value that can be sampled at any given point in time. Not all quantifiable performance characteristics which are useful for scheduling easily conform to this model. For example,

it is useful to predict the duration of time that a resource will be available based on previous availability history. Availability, in a time series form, is a series of binary values indicating available or unavailable at a particular time. Thus, the measurement levels are bimodal. While Markov-based models are adept at predicting modality, time-series analysis tends to be less effective. It is possible to incorporate state-transition models into the NWS forecasting framework, but at present they are not used by the system.

Dynamic Model Differentiation

Rather than relying on a single model, the NWS uses a mixture-of-experts approach to forecasting. A set of forecasting models are configured into the system, each having its own parameterization. Given a performance history of previously observed measurement values, each model is exercised to generate a forecast for every measurement value based only on the measurement values that come before it. That is, given a performance history of N values, a forecast is generated for each. To generate a forecast for measurement k, only values up to measurement $k-1$ will be presented to each forecasting model, for all values $1 \leq k \leq N$. We term this method of replaying a performance history to generate a forecast for each known measurement value *postcasting*.

Postcast errors are generated for each forecasting method by differencing each measurement with the forecast generated for it. By aggregating the postcast errors, each method is assigned an overall accuracy score for the complete history up to the point in time when the forecast is generated. When a single forecast is required, the NWS forecasting system applies the postcasting procedure to all of the configured prediction models using the most recent performance history available, and ranks each prediction model in terms of its accuracy. The most accurate model is then chosen to make the requested forecast. Each time a forecast is requested, the NWS recalculates the accuracy ranking using the most recently gathered history. The NWS constantly gathers measurement data from sensors that it controls. Thus, the performance histories that it uses are, typically, up-to-date at the time a forecast is requested from the system, and the forecaster choice takes into account the "fresh" historical data.

This method of differentiating between competing models based on previously observed accuracy has several advantages. The first is that it is non-parametric. Each individual model may have a specific parameterization, but the complete technique simply takes the constantly-updated performance history gathered by the NWS as its only input. A second potential advantage is that it is possible for the system to adapt to changing conditions in the cases where the performance response series is non-stationary. For example, if an exponential smoothing predictor with a gain factor of 0.01 is the most accurate predictor at one point in time, and conditions change so that a sliding window median predictor with a window size of 10 becomes the most accurate (due to a change in the series dynamics) the system will switch predictors if the

change is persistent enough to cause the aggregate error ranking to change. If, however, the forecasters have been exposed to an extensive performance history before the change point, it may take a great deal of time for the better method to garner a lower aggregate error.

To improve the response of the overall technique to changes in the underlying dynamics of each measurement series, the NWS forecasting subsystem also selectively limits the amount of history during postcasting to determine if "old" data is harming accuracy. During the dynamic model selection phase, a postcast is conducted using all previously available data. In addition, the system conducts postcasts using different windows of previous data (always starting with the most recent data and working backwards in time) and records the "winning" forecaster for each window size. The number of postcast limiting windows and their sizes are fixed at compile time, but can be changed via configuration parameters when the forecasting subsystem is built. Each window size of previous history is subsequently treated as a separate forecaster and a final accuracy tournament determines which forecaster will be used.

The pseudocode shown in Figure 3 summarizes how NWS forecasts are generated from a given measurement trace. The effect of using this method

```
input: T: measurement trace
       F: set of forecasting models that take a trace of fixed size and produce
          a forecast of next value
       W: a set of integer window sizes to limit postcasting

for each window size in W + (entire history)
       for each forecaster in F
           postcast current forecaster over current window size in T
             (window size slides over all of T)
        record aggregate error for current forecaster
        end for
        record forecaster with lowest aggregate error for this window size
end for

choose forecaster and window size with lowest aggregated error and make
       final forecast using it
```

Fig. 3. Pseudocode for NWS Forecasting Methodology

is that either the forecaster that has the lowest aggregate error since the beginning of the trace will be chosen, or the forecaster that has the lowest error over an abbreviated history of fixed size will be chosen that the best forecaster. If the system has quickly changing dynamics, forecasters that work well over short histories should be more accurate since they do not include stale data.

5 An Example

To illustrate the types of forecasts that can be generated by the NWS adaptive forecasting technique, we will use the following example. Figure 4 depicts an application-level TCP/IP trace from the University of Tennessee (UTK)

to the University of California in San Diego (UCSD). The trace times a 64 kilobyte TCP/IP socket transfer and an application-level acknowledgment and from that timing and data size, it calculates a throughput measure. The socket buffers for this trace are 32 kilobytes, and the buffers used in each communication system call are 16 kilobytes. The entire trace spans the month of June, 2000 with one transfer recorded every 30 seconds.

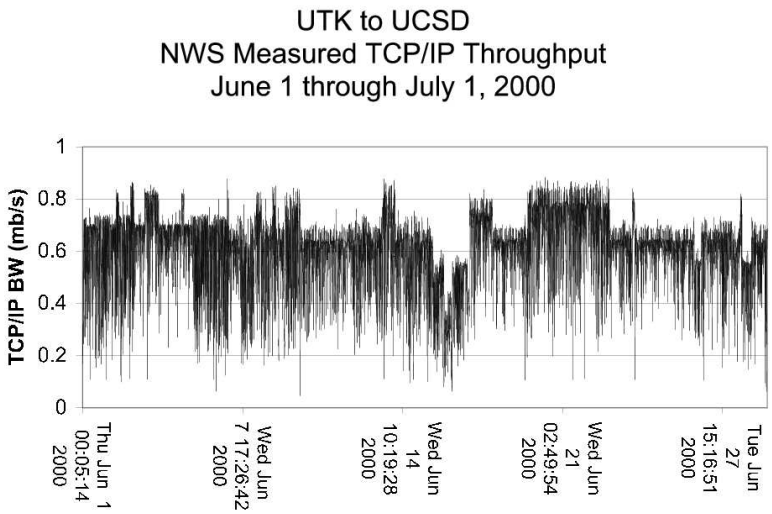

Fig. 4. Internet Throughput, 64KB messages

[3]

A companion trace of `traceroute` data showing the end-to-end gateway traversal indicates that the series is likely not a stationary one. The routes used to connect UTK with UCSD changed from time to time due to routing table misconfigurations and maintenance.

In Figure 5 we show the NWS forecasts (the light color) superimposed over the measurement series (dark color). After each measurement was gathered, it was passed to the forecasting subsystem and a forecast (using the method

[3]The actual trace contains a little over 85,000 measurements. As such, the trace data used to generate the graphical figures in this paper has been decimated. All forecasting and error calculations, however, use the complete trace. We decimate the time series output only for graphical display purposes.

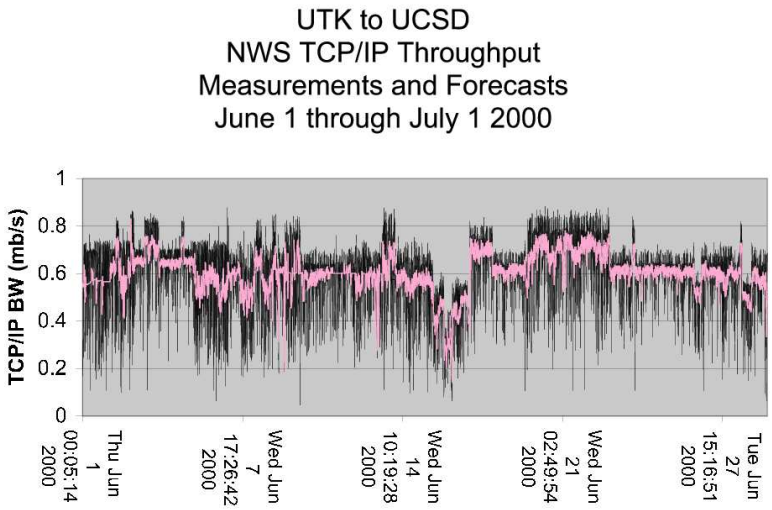

Fig. 5. NWS Forecasts of UTK to UCSD Throughput

described in Section 4) was generated to produce the forecast trace. From the figure, it is clear the the NWS forecasters determine a centralized or smoothed estimate at each step in the series. The figure also provides a qualitative depiction of the forecasting error. Each light-colored forecast point is matched vertically with the dark colored measurement data point it forecasts. The degree to which the dark features are showing (i.e. are not obscured by a light-colored features) provides an indication of the overall error.

More quantitatively, Figure 6 details the error performance of the forecasting system. The vertical axis of the graph shows the forecasters that are currently configured into the NSF Middleware Initiative (NMI) [32] release of the NWS and their individual error performance. Error (shown on the horizontal axis) is measured as the square-root of the mean square error (MSE). If each NWS forecast is considered an conditional expectation of the succeeding measurement, then the forecasting error approximates the conditional sample standard deviation. We do not claim, however, that the conditional expectation or the conditional standard deviation are either optimal or unbiased estimates for the true conditional mean and variance — only engineering approximations.

Each of the horizontal bars in the figure (except for the top two) shows the error performance of a different forecasting model. Notice that one type of

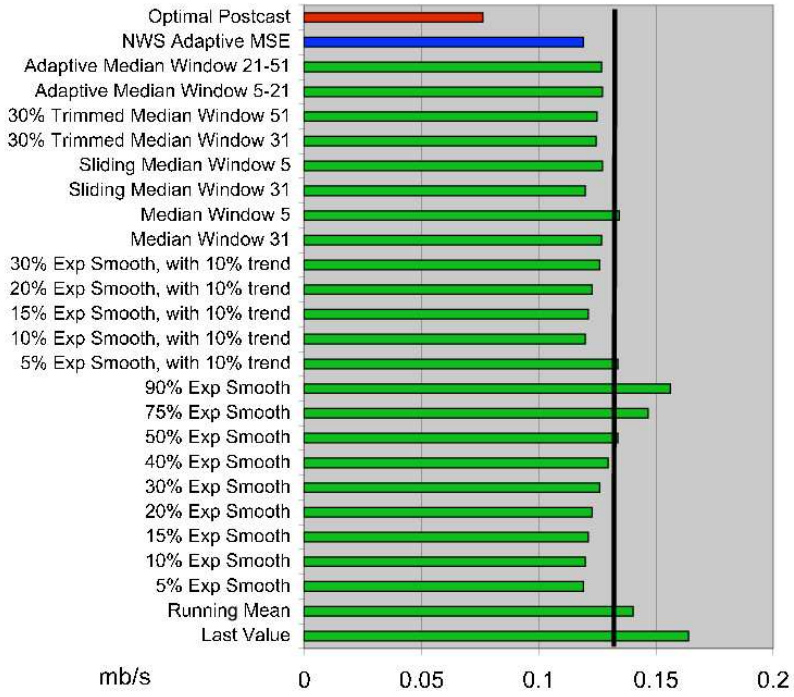

Fig. 6. NWS Forecasts of UTK to UCSD Throughput

model (e.g. exponential smoothing [24]) is used multiple times with different parameterizations (e.g. the gain factor). The entire forecasting suite is similarly populated by different parameterizations of a smaller set of models. The software has been modularized to permit new model types, and different modularizations of the included models when it is configured. Currently, the NWS uses 24 model parameterizations (shown in the figure) in the standard release. The choice of these models is based on our anecdotal experience with effective prediction techniques in the Grid settings where we or our collaborators have constructed successful schedulers.

The error bar that is second from the top in the figure shows the error performance of the adaptive NWS technique. That is, this line indicates the true error an NWS user would have seen from the forecasts generated when the trace was gathered "live." Notice that it is equivalent to the minimum er-

ror across all configured forecasters. While space constraints prevent us from demonstrating this effect more completely, in all postmortem trace analyses performed by our group since the inception of the project, this phenomenon has been observed. The NWS adaptive forecaster achieves at least equivalent (if not slightly better) error performance as the most accurate of its constituent models. We do not claim that the adaptive forecaster must achieve equivalent accuracy. It is clear that it is possible to construct a series artificially for which the adaptive technique will be less accurate. Our experience, however, is that for empirically observed measurement series taken from Grid systems, this phenomenon occurs in every case.

Also for space constraints, we have omitted the limited postcast history errors. For this trace, the best overall adaptive performance comes from considering the all previous values at any given point in the trace (despite the potential for non–stationarity). That is, the forecasters that adapt based on a shortened window of history are less accurate that the ones that considers all previous measurements.

The error bar marked "Optimal Postcast" at the top of the figure indicates the theoretically maximal forecasting performance (minimum error) that the method could have achieved if the best predictor at each step were known. That is, each time a forecast was generated, if the most accurate prediction made by any predictor in the suite were used, the aggregate error measure shown by the top error bar in the figure would have resulted. This measure represents the upper bound on accuracy since it is the most accurate that the entire suite could have been if perfect foreknowledge of predictor accuracy were possible.

The bottom two error bars are also noteworthy. The bottom most error bar (marked "Last Value") represents the accuracy obtained by simply using the last observed value as a prediction of the next performance measurement at each step. This method corresponds to the typical way in which Grid users make *ad hoc* estimates without the aid of numerical forecasting techniques. Most users simply "ping" the desired resources or read the most recent performance measurements recorded for those resources by an available monitoring tool, and compare the measurements that they observe to make their scheduling decisions. This method is, by far, the least accurate of those that are available. A second common method is to use a running average as an estimator based on the assumption that the series is converging to a single mean performance value. The running mean is more accurate than the last value as a predictor, but again, significantly less accurate that other, only slightly more sophisticated techniques.

One possible argument for using the more simple last value or running average techniques is that the computational efficiency of these methods is quite high. The last value requires no computation, and the running average can be calculated as a simple on-going update. The techniques that we have chosen to incorporate in the NWS implementation, however, come primarily from the signal processing disciplines making very high-performance versions

possible. With careful implementation, each forecast shown in Figure 5 required 161 microseconds on an unloaded 750 MHz Pentium III laptop. Thus the additional computational overhead introduced by our implementation of the adaptive methodology introduces negligible performance overhead. More concretely, considering the difference in error performance between the Last Value predictor, the adaptive NWS predictor, and the Optimal Postcast, our implementation halves the error difference between optimal and last value at a cost of 161 microseconds per forecast.

Forecasting Error

For Grid scheduling, the forecasting error can also be used to gauge the value of a particular resource. In Figure 7, we show a trace of TCP/IP throughput between adjacent workstations attached to a 100 megabit-per-second Ethernet at the San Diego Super Computer Center (SDSC). The probe size for this trace is 64 kilobytes, with one probe taken every 120 seconds, and the adaptive NWS minimum MSE forecast is superimposed over the measurement trace. The Ethernet segment, however, is also shared by other hosts at SDSC. That is, it is not dedicated to a particular cluster, but rather is a part of the shared, local-area network infrastructure. In Figure 8 we show three days worth of TCP/IP trace data between a pair of cluster nodes at UTK. The nodes are attached via switched gigabit Ethernet that is dedicated to intra-cluster communication exclusively. Both figures are plotted using the same scale. Note that the missing values in Figure 8 occur when the machine was taken out of service for maintenance.

As expected, the forecast performance of the dedicated gigabit Ethernet link is higher than that for the 100 megabit connection throughout the measurement period. The gigabit link's forecast hovers near 100 megabits per second for most of the trace, while the forecasts for the 100 megabit link are mostly just above 50 megabits. However, the $\sqrt{MSE}$ value (termed the forecast deviation in each figure) for the 100 megabit trace is 9.7 megabits-per-second. For the gigabit trace, it is 64.3 megabits-per-second. Roughly speaking, as a percentage of the forecast value, the forecast deviation is is approximately 20% of the forecast for the 100 MB Ethernet, but 60% for the gigabit link. For programs with malleable granularity that can be controlled by an on-line scheduler, a more predictable performance response despite lower absolute performance may make a resource more valuable than faster, less predictable resource. Data parallel or SPMD (Single Program Multiple Data) programs, for example, have their overall performance defined by the slowest task. In [9] we describe a dynamic scheduling technique for data parallel programs that automatically partitions the workload based on forecast performance levels. For that system, a gross over prediction of delivered performance results is extra work assigned to the potentially slow resource and, as a result, the application executes with less-than-expected performance.

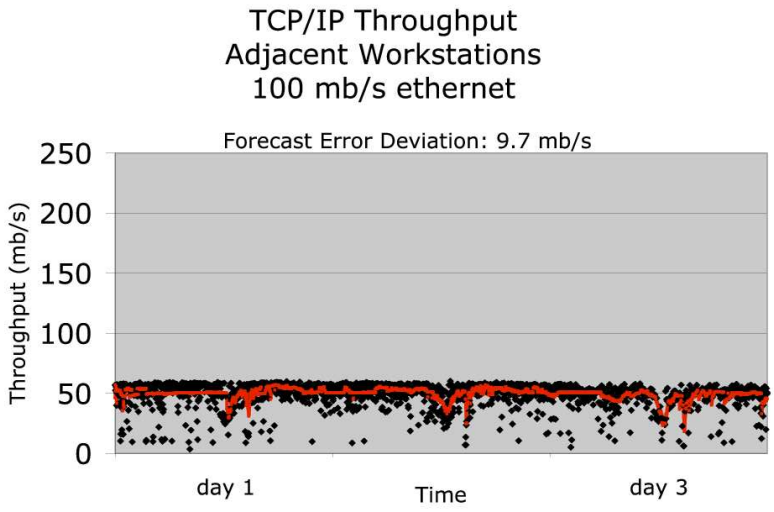

Fig. 7. NWS Measurements and Forecasts of 100 MB Ethernet at SDSC

This example also illustrates the role that forecasting can play in detecting faulty resources. For a dedicated gigabit switched network, a forecast value near 100 megabits, with an error deviation of more than 60% is indicative of a potential problem. When shown this data, the system administrators for the cluster upgraded the system software (several times, hence the drop-out in the trace) in an effort to correct a suspected configuration problem. Using the NWS forecasting, it is possible to build an alarm system that would have signaled the potential problem much earlier [29].

6 Forecasting Error and Empirical Prediction Intervals

In the previous example, the forecast error permits a ranking of resources by their predictability. For some measurement streams, the forecasting error also can be used to generate a quantifiable bound on the predictability of the measurements in the stream. By treating the MSE as the conditional sample variance, one can generate a prediction interval for the next value as $(forecast - K \cdot \sqrt{MSE}, forecast + K \cdot \sqrt{MSE})$ where K is a multiplicative factor to be determined. We have used a K-value of 3 to bound the predicted execution times of worker tasks in a master-slave distributed implementa-

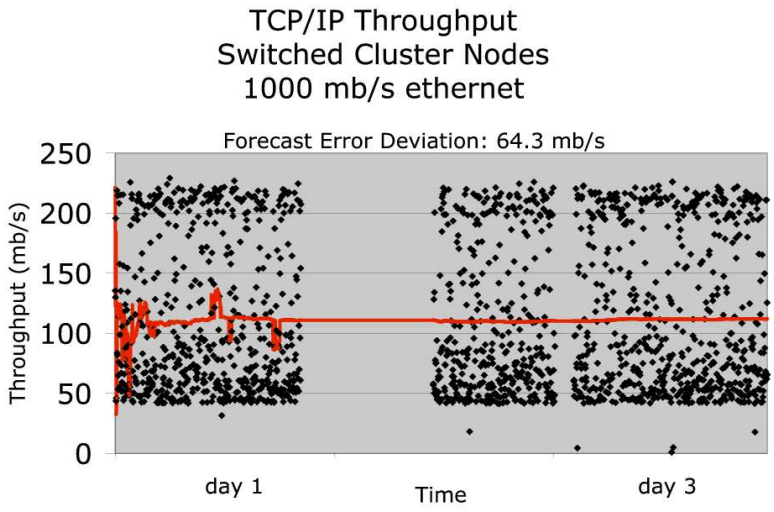

Fig. 8. NWS Measurements and Forecasts of switched gigabit Ethernet within a cluster at UTK

tion of FASTA, a commonly used genetic sequencing application [43]. For the genome sequences we examined, a K-value of 3 allowed the scheduler to determine the "dependable" task execution time across a wide range of target resources.

In order to predict the performance of an individual resource (as opposed to the convolution of data-dependent task execution time with resource performance response as in the FASTA experiment) one can often effectively employ smaller values of K. For example, we observe, that for network throughput, a factor of 2 yields a 90% or better "hit rate" for each succeeding measurement, with the rate being above 95% for most of the measurement streams we have encountered.

Figure 9 shows this form of empirical prediction interval as generated by plotting $forecast \pm (2 \cdot \sqrt{MSE})$ for the UTK-to-UCSD throughput trace shown previously in Figure 4. At each point in time, the prediction interval is formed by making a forecast for the next measurement value, and then adding and subtracting $2 \cdot \sqrt{MSE}$ for the MSE that has been observed up to that point. The capture rate for this trace is 95.6%. That is, over the entire measurement period, the confidence range predicted by $\pm 2 \cdot \sqrt{MSE}$ captures the next measured value for 95.6% of the total number of measurements. We note that one

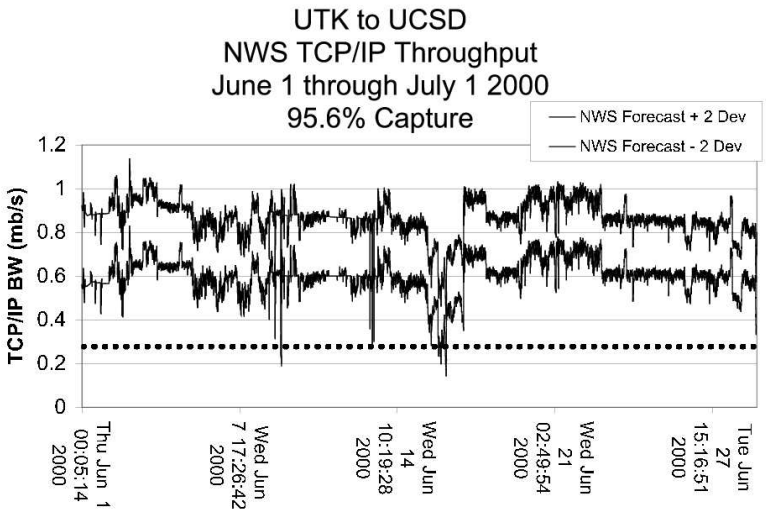

Fig. 9. Prediction Interval formed by $\pm 2\sqrt{MSE}$

can also make one-sided predictions using the same idea: For example, if a scheduler (such as the one reported on in [43]) were concerned with the *minimum* available performance, it could determine a K-value to produce lower prediction bounds that have a capture percentage approximately equal to a given value.

The dotted line in the figure represents the 5% quantile for the entire trace, with 95% of the measurements falling above this line. If the data were treated as a sample rather than as a time series, this value could be used as an empirical estimate of the minimum throughput level with 95% confidence. By treating the data as a potentially non-stationary series, and recalculating the confidence interval at each time step based on forecasting error, the NWS methodology generates a significantly tighter lower bound than a sample-based quantile method.

As an example of how pervasive this phenomenon is for TCP/IP network throughput, we show the distribution of forecast capture percentages for a complete Grid system we monitored during the month of October, 2002. The Grid Application Development Software (GrADS) [7, 23] project, as part of its research agenda, maintains a Grid testbed based on stable deployments of the Globus [19, 22] toolkit and the NWS. The purpose of the testbed is to provide support for the development of Grid programming tools, and to act

as a production Grid environment in which GrADS enabled applications can be tested and evaluated. Globus and the NWS provide the base Grid software infrastructure that GrADS software tools build upon. Approximately 50 users (programmers, graduate students, and project administration personnel) have access to the testbed at any given time, and it is maintained as a permanent resource. Thus, the GrADS testbed constitutes an example of a practical, working Grid.

During the month of October, 2002 the GrADS project developed and deployed six GrADS-enabled applications for demonstration at SC02 — a prominent high-performance computing conference that takes place annually in November. As such, the October measurement and forecast data for the testbed reflect Grid dynamics in a production computing setting.

The testbed comprises 77 host machines organized into several Linux clusters as well as various independent Unix and Linux machines. Within each cluster, the available networking is either 100 megabit Ethernet or gigabit Ethernet. Clusters at a single site are either connected via local area networking or via the campus network infrastructure (GrADS testbed sites are located at various Universities and two research laboratories). Inter-site network connectivity is provided by the Internet, although several of the sites have experimental, high-performance access to an Internet backbone. The GrADS sites are geographically distributed with machines located at Rice University, UCSD, UTK, the University of Illinois at Urbana-Champaign (UIUC), Indiana University, the Information Science Institute (ISI), and the University of California at Santa Barbara (UCSB).

The NWS provides support for organizing end-to-end network measurements hierarchically. Not all machines must conduct machine-to-machine probes of network connectivity to provide forecasts for the entire resource pool (details on this scaling technique are described in [44] and [52]). For the GrADS testbed, 1234 NWS TCP/IP probe traces are sufficient to provide a complete end-to-end performance forecast report. Finally, the NWS uses a variety of probe sizes ranging from 64 kilobytes per probe to 4 megabytes per probe, depending on the link characteristics at hand. As such, the complete GrADS testbed trace captures good cross-section of available network technologies and probe sizes under Grid computing loads.

Figure 10 shows the distribution of capture percentage over the total October trace set for $K = 2$. All network types (intra-cluster, intra-site, and inter-site) are represented. The traces have been sorted from smallest capture percentage to largest. The $x - axis$ depicts trace number and the $y - axis$ shows the capture percentage observed for each trace using $\pm(2 \cdot \sqrt{MSE})$ to form each prediction interval. The smallest capture percentage is approximately 89%. In 1084 of the 1234 traces, however, the predictions capture 95% or more of the future values. We are just beginning to study this phenomenon in detail, but anecdotally the GrADS testbed analysis reflects the common experience reported by NWS users for TCP/IP throughput in different settings.

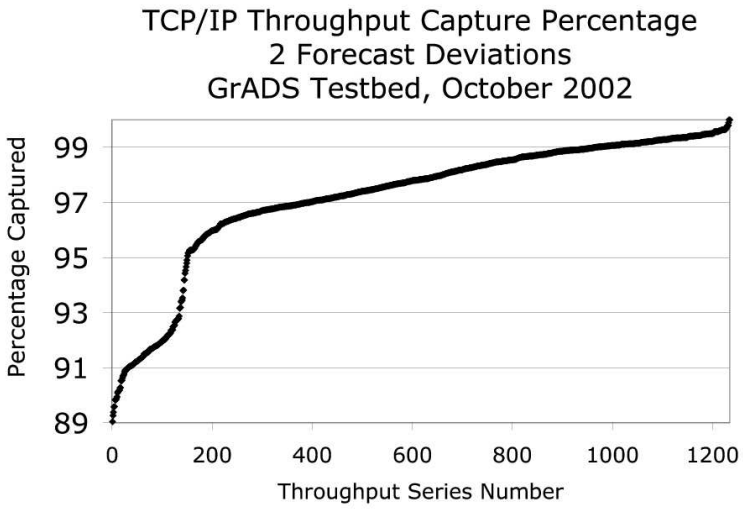

Fig. 10. Distribution of Capture Percentages for TCP/IP Throughput on the GrADS Testbed

In Figure 11 we show the cumulative distribution of CPU load measurement capture percentage, using $K = 2$, for the 77 hosts in the GrADS testbed. The NWS supports a CPU monitor that reports the percentage of CPU cycles that are available to an executing process. The default periodicity (which is what has been used to monitor the GrADS machines) is 10 seconds. Thus each of the 77 traces contains approximately 250,000 measurements of available CPU fraction at each 10 second time step. The number is approximate since data may be missing when a machine becomes unavailable as is the case in Figure 8. For 75 of the 77 traces, $forecast \pm (2 \cdot \sqrt{MSE})$ also generates a 95% (or higher) capture percentage.

It is clear that the empirical method warrants more study. Resource characteristics such as TCP/IP round-trip time are not as predictable as throughput or available CPU fraction. We suspect that available non-paged memory will prove to be similar to CPU measurements in terms of predictability, but the NWS memory sensor has only recently been developed giving us a limited experience with true load characteristics. ted experience with true load characteristics.

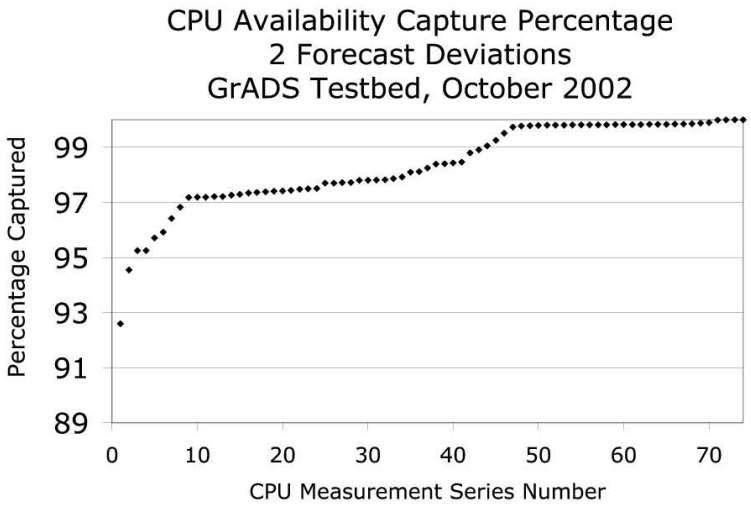

Fig. 11. Distribution of Capture Percentages for CPU Availability Measurements on the GrADS Testbed

7 Lessons Learned from Development and Deployment

Having developed and deployed the NWS in a variety of contexts, we have repeatedly observed somewhat surprising anecdotes within the user community. While we are hesitant to give these observations the status of "principles" they never-the-less recur with enough frequency to warrant some exposition, if only to provide insight into the success and failures the system has experienced. Moreover, many of our experiences run counter to the "conventional wisdom" or in some cases, contradict predicted outcomes made by acknowledged experts. In all cases, however, we present these anecdotes without attribution and acknowledge that any misrepresentation is strictly our responsibility.

7.1 Grid Performance Tools versus Grid Performance Services

Many Grid users install and use individual resource performance monitoring tools to aid in resource discovery. While system administrators clearly understand the need for Grid services such as remote sign-on and file system access, performance monitoring services (particularly for dynamically changing performance data) are often overlooked since they are used, primarily, to optimize

rather than enable application execution. At the same time, user-level performance tools, particularly for measuring network performance, are plentiful, easy to install, and simple to use. Thus many Grid installations have an administrator supported infrastructure for secure access, but leave the problem of gauging resource performance to the individual users.

There are two problems with this approach. First, most individual performance monitoring tools are designed for single user execution. Popular application-level network monitoring tools such as Iperf [26], netperf [28], and nttcp [33] all measure end-to-end network throughput by sending data from a source host to a sink hosts, and timing the transfer. To ensure that the effects of TCP slow-start [27] do not affect the measurements, these tools (by default) will transfer data continuously for tens of seconds to ensure that steady-state behavior is being observed.

If used occasionally, in isolation (e.g. for performance debugging) the network load introduced by lengthy network probes is negligible. However, if many users each run network probes individually, without coordination, a great deal of unnecessary load may be generated. For example, all hosts at the University of California, Santa Barbara (UCSB) share a common network path (once they exit the campus backbone) to the University of Wisconsin (Wisc) backbone that traverses the Abilene [1] network. While the paths through each campus may differ, all UCSB-to-Wisc transfers share the same route across Abilene and, more importantly, the performance of that route dominates the end-to-end performance. Thus multiple users at UCSB issuing throughput probes to multiple hosts at Wisc will each introduce tens of seconds worth of network load to measure the same artifact: the performance of the cross-country Abilene route. Perhaps more problematically, if enough users issue these probes simultaneously, or if multiple users probe the same host (or issue probes from the same host) the measurements that are generated measure contention between probes.

The NWS solves this problem using a hierarchy of cliques, as described in Section 3. Cliques at either campus provide intra-campus measurements, while a single campus-to-campus probe sequence measures the cross-country throughput. Moreover, the NWS proxy caching layer can automatically generate a virtual fully interconnected network by filling in the "missing" network measurements between hosts at either campus with forecasts taken for the inter-campus link. As such, the NWS measures the shared path using a single sequence of measurements but at the same time can present a virtual all-to-all measurement picture to all interested clients by correctly reporting the dominant shared performance for any pair of hosts.

A second problem with the use of tools rather than a service for generating measurements is that user tools are typically designed to require user intervention when resource failure requires the tool to abort. Returning to the network probing example described above, the TCP protocol by default does not include an inactivity time out. That is, once a TCP handshake has completed, a network partition will not cause the TCP connection to shut

down or abort [4]. The assumption made by most user tools is that the user will manually "time out" the tool and abort it from the command line. Often, due to the need for continuous and historical measurement, these tools are executed repeated within scripts causing end-point memory and process load.

The NWS TCP throughput probe, however, includes portable time out mechanisms and an adaptive time out discovery protocol [4] so that long-running, unattended execution is feasible. However, the engineering effort required to build a portable and reliable time out mechanism for TCP sockets (without kernel modification) introduces another potential point of confusion. In particular, it may be that the additional mechanisms introduce overhead that affects the quality of the measurements. Indeed, one reason often cited in justification for the use of a particular individual network monitoring tool is that the tool in question is believed, by its user, to be the most accurate among all of the available options. In addition, several users, when queried as to why they preferred a particular tool to the NWS as a service claimed that the tool in question generated more accurate measurements of end-to-end throughput. Questioned further, some speculated that the reason for the loss of accuracy was that the NWS network probe included time out mechanisms that most applications using TCP sockets do not, and the time out mechanisms introduced extra overhead.

Figure 12 shows a comparison of the throughput measured by three popular user tools – Iperf, netperf, and nttcp – and the NWS throughput measurement service. To generate this data, we ran each different method back-to-back (so that all methods would experience approximately the same ambient network conditions) every 60 seconds over an 72 hour period resulting in 400 comparable measurements for each technique. We configured all four systems to use the same end-point buffering which is the one used by default in Iperf and to transfer the same amount of data. The large circular "dot" for each method marks the median throughput observed and the bars show the range of values between the first and third quartiles. From this data, it is not possible to conclude that there is any statistical difference between the measurements generated by these methods. The three tools and the NWS service all generate clearly overlapping ranges of values. The NWS probes, however, include all of the overhead necessary to implement reliable socket time outs at the application layer.

As such, we speculate that user-reported perception of tool utility is not based on accuracy, but rather on intellectual and manual ease-of-use. All three of these network measurement tools are well-engineered, documented, simple to understand, install, and use. The NWS is a long-running Grid service designed to support many clients and resources simultaneously. While it does not require special user privileges (each user can in principle install a sepa-

[4]The optional KEEP_ALIVE feature of TCP is designed to implement an inactivity abort according to RFC-1122, but the timeout value by default can be no less than 2 hours which is often too long for Grid applications.

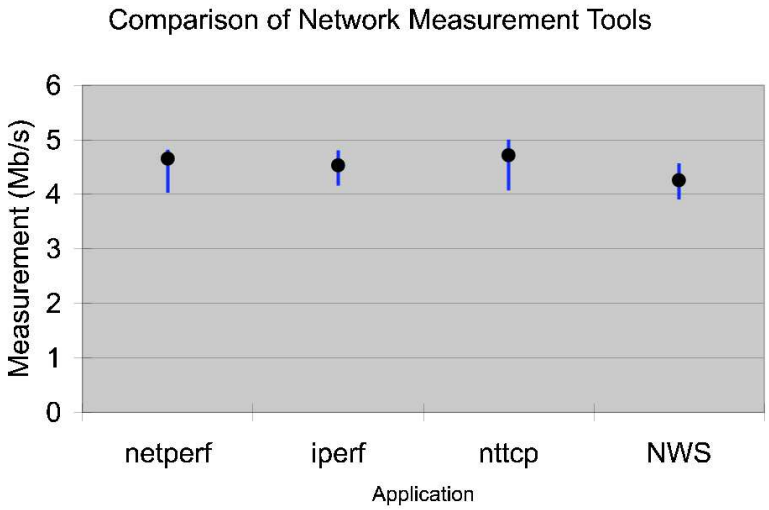

Fig. 12. Comparison of Internet Throughput Measurements between a host UCSB and one at the University of Wisconsin

rate instance of the NWS) it is necessarily more complicated than a simple "ping" tool. As a result, if the local administrator has elected not to install the system, or plans to as a low priority task, we believe users naturally gravitate towards using tools that they can easily understand, install, and maintain themselves. Subsequent familiarity then breeds a "lore" regarding tool accuracy which, when examined critically, is unverifiable. The cost of this convenience, however, is the wasted resource consumed by redundant measurements. By carefully engineering the and structuring the measurement system, a Grid service such as the NWS can yield the same levels of accuracy with greater dependability using significantly fewer resources.

7.2 Network Heterogeneity

Another observation that we have made while developing the NWS and Grid applications that use it [13, 51, 9] is that network performance is truly heterogeneous, and the way in which applications access the network should take this heterogeneity into account to achieve the best possible performance. The use of parallel sockets by applications such as GridFTP [2] and the Internet

Backplane Protocol [40] (IBP) illustrates the need to consider such heterogeneity.

For systems such as the TeraGrid [47] where a high-bandwith dedicated network links nationally distributed computing nodes, the standard congestion avoidance and control mechanisms built into commercially available TCP implementations prevent applications from achieving maximum possible end-to-end throughput. The specific reason is that TCP uses the timing of packet acknowledgments to control the speed with which it will introduce new data into the network, both at start-up, and after a packet has been dropped. For networks with high bandwidth-delay products and low drop rates (such as the 40 gigabit/second TeraGrid network), the loss of throughput can be substantial. On these systems, to avoid the need for specially engineered kernel-level TCP stacks, many applications use parallel sockets to circumvent the unnecessary congestion avoidance and control mechanisms.

However, in network settings where the bandwidth-delay product is lower, or where packet loss due to congestion is a possibility, parallel sockets can have the opposite effect. Figure 13 compares the performance of the IBP streaming download protocol [40] that uses parallel sockets, with a single socket implementation that uses NWS forecasts for proximity resolution and adaptive time out discovery [5]. The IBP Progress-driven protocol [40] uses parallel sockets and a deadline-driven scheduling algorithm to download segments from a replicated file. Different file segments are fetched in parallel within some pre-specified progress window. If the segment at the beginning of the window is late, that segment is fetched in parallel from where it is replicated before new segment transfers are initiated. One simplicity advantage of this approach is that it is completely reactive. That is, it does not require a prediction of future performance levels or failure likelihood, but rather reacts to conditions as they occur.

In contrast, the NWS protocol uses throughput forecasts to rank the replica sites in terms of their download speed. It then maintains a database of forecast response times, and forecast variance so that it can automatically determine how long it should wait for each replica to respond. Only one segment of the file is downloaded at a time. The protocol tries the replicas in order of their speed, and switches between them when a time out occurs [5, 4].

In the figure, we show the cumulative distribution of file arrival times where six replicas for each file are distributed across PlanetLab [39], the download point is located at UCSB, and each segment has an artificially induced 5% chance of failing. From the figure, it is clear that the NWS methodology outperforms the IBP methodology while maintaining the same level of robustness (both systems completely download all files) and using substantially less bandwidth. In this case, the additional network load generated by the IBP protocol through the use of parallel sockets over the Internet slows the individual file transfer times. The adaptive NWS protocol, however, uses the fastest replica when it can and relies on rapid failure discovery and remediation for robustness. Thus parallel sockets, while an excellent choice for dedicated high

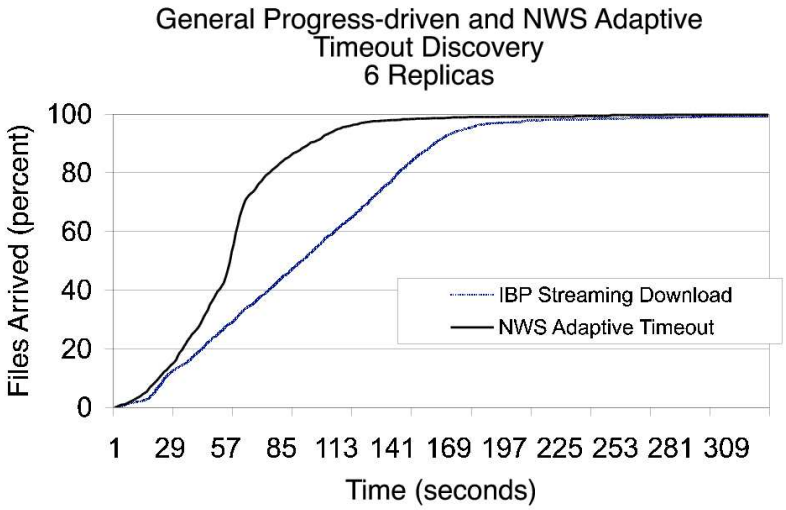

Fig. 13. Empirical Cumulative Distribution of File Download Arrival Times for IBP Download Protocol and Adaptive NWS Protocol using Six Replicas

bandwidth-delay product networks yields lower application-level performance over the Internet when compared to a socket scheduling system that uses performance forecasts to control resource usage.

8 Measuring and Predicting Other Resource Characteristics

While the empirical and adaptive time series forecasting approach has proved useful in a variety of contexts, there are quantifiable resource characteristics that are not well-modeled by a periodic statistical series. Resource availability duration (i.e. resource "lifetime"), for example, is represented as a time highly correlated time series with two modes: "available" and "unavailable" as depicted in Figure 14. Essentially, "available" must be represented as one value (a 1 in the figure) and "unavailable" as another. Further, the prediction of interest is not for the next value, but rather for the duration of time that a value will remain constant before it changes.

To make predictions of this type, the NWS requires both the ability to measure the quantity of interest and secondly a different set of forecasting

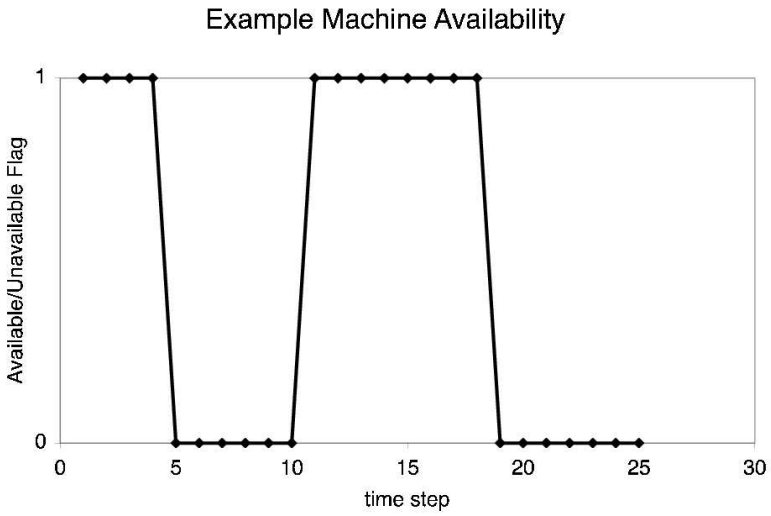

Fig. 14. Machine Availability as a Time Series

models that are not time series based. In the case of machine availability, we have developed an availability sensor that measures the time between machine restarts, and a process lifetime sensor that can be used to measure processor occupancy in cycle harvesting systems like Condor [46], Entropia [18], and BOINC [10]. These new sensors, which are part of the current system, do not rely on heart-beat messages and soft-state registration to measure availability. Doing so convolves the observed host availability distribution with the distribution of network partition frequency between the measured site and the storage location where the measurements are captured. That is, sending a heart-beat message to a collector (an NWS *memory* process in our case) as a measure of host availability records both host failures and failures in the network connecting the host and the collector in a way that cannot be easily separated later. Instead, the sensors send a running accumulation of "up time" so that the effect of missing messages due to network partition can be filtered of the measurement history.

Predicting machine availability requires forecasting techniques that are substantially more heavy-weight than the on-line time series models. The mode of operation, then, is for the NWS to archive availability measurements and calculate predictions as a background task rather than on-demand as it does for performance levels.

The type of prediction is also different from what the current system generates. Our initial target is to provide availability predictions to aid process and checkpoint scheduling. Dynamic application schedulers would like to be able to predict when a checkpoint should be taken (so as to minimize checkpoint overhead) and/or to decide if checkpointing is even necessary. For example, a machine with a 99.9% guaranteed availability of 10 minutes can run a 10 minutes job to completion 999 times out of 1000 attempts without checkpointing.

This last example also illustrates the nature of the predictions that application schedulers require. Rather than the *mean time to failure* which is a useful metric in many industrial engineering contexts, the scheduler must estimate how long a resource will be available until the probability of failure exceeds some specified threshold. That is, the scheduler is typically interested in a specific *quantile* from the cumulative failure distribution, rather than the mean. Returning to the example, if the 0.001 quantile of the cumulative machine availability distribution were known, the probability that a machine would be available *at least* as long as the specified duration would be 0.999. An application scheduler, then, requires a prediction in the form of a quantile at a specified level of certainty corresponding a failure tolerance either the application or its user is willing to accept.

Moreover, to make a reliable estimate that can be trusted at the given level of certainty, the *confidence bounds* on the estimated quantile must also be determined. Any estimate that is generated from a observed sample of measurements will include random estimation error. If a statistical bounds on that error can be calculated, worst-case bound at the specified level of confidence should represent a conservative *guarantee* of availability.

We have explored both parametric and non-parametric approaches to the problem of generating quantiles and confidence intervals on the estimated quantiles using the NWS. Because of their computational complexity and because they require efficient archival storage, we have not yet incorporated quantile estimation techniques into the NWS forecasting system. Our intention is to do so at some future release, however.

The parametric approach we have taken is to develop automatic software for implementing Maximum Likelihood Estimation (MLE) for various candidate models such as exponential, Pareto, and Weibull. Given a model and a historical trace of availability, the software estimates both the MLE parameters that best describe the data with the model, and confidence intervals for the fitted model. Figure 15 depicts a comparison of model fits for the MLE-determined exponential, Pareto, and Weibull models using availability data gathered from the student instructional machines located at UCSB. At UCSB, the power switch on the machines available to all computer science students is not protected. When using a machine from its console, students routinely "clean off" foreign processes (owned by other students) by power cycling the machine, causing a reboot. The figure compares the cumulative distribution of observed availability measurements to the three models.

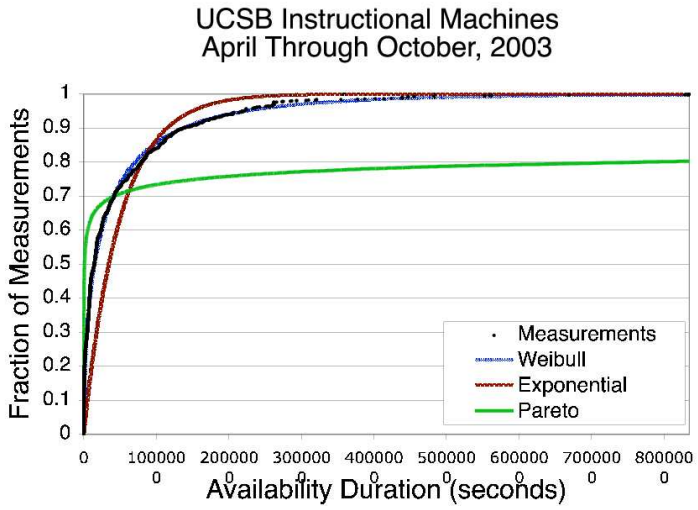

Fig. 15. Machine Availability Data and MLE Exponential, Weibull, and Pareto Models for UCSB Instructional Machines, April through October, 2003

The dark points depict individual availability durations and the smoothed lines show the three different models. The Pareto model carries significantly more weight in the tail than the data indicates. It predicts the that the 0.8 quantile will occur at approximately 8000000 seconds (approximately 92 days). That is, the Pareto model predicts that 20% of the availability durations will be longer than 8000000 seconds. From the data, however, only two of the 1765 availability durations lasted that long making the Pareto overly optimistic. In contrast, the exponential model does not predict that availability durations will last as long as they did. For example, the 0.95 quantile from the data occurs at 2189875 seconds (approximately 25 days) meaning that 5% of the measured availability durations were larger than this value. The exponential model predicts the 0.95 quantile to occur at 1495871 seconds (approximately 17 days) thereby under estimating the possibility of longer durations. The Weibull model, however, fits the data so accurately that its curve is obscured by the data itself in the figure. For the 0.95 quantile, it estimates the duration to be 2234657 seconds missing the measured quantile by 44782 seconds or a little over one-half a day. Maximum Likelihood Estimation is currently the best known automatic technique for fitting parametric models to observed data for models with a small number of parameters. Thus the

Weibull model truly describes the shape of the distribution more accurately than the Pareto or the exponential.

We have also developed software (using goodness-of-fit p-values as heuristics) that attempts to automatically make the best fit determination. We have examined other availability contexts including Condor [46] where jobs are terminated when a resource owner reclaims a resource, and an Internet host availability conducted by Long, Muir and Golding in 1995 [31]. Perhaps surprisingly, we have found that an MLE Weibull model fits the observed availability distributions rather closely. Moreover, previous work with a small number of student and faculty workstations in 2001 [25] also found Weibull models to be effective.

While the Weibull fit was clearly best in our study (see [34] for details), it did not yield the most accurate predictions of future availability durations. The software also generates confidence intervals on the MLE parameters it determines as part of the model-fitting process. From these confidence intervals, it should be possible to calculate the conservative worst-case estimate for the quantile of interest. For quantile prediction, however, it is possible to use non-parametric techniques to estimate a quantile, and confidence bounds for it, without specifying (or indeed knowing) what the underlying distribution is. One such technique uses repeated sub-sampling of the observed data and *bootstrapping* [14] to estimate the quantile. We have developed a second non-parametric method, which we term the *Brevik Method*, that is based on the Binomial distribution. Table 1 shows a comparison of the predictive accuracy achieved by using an MLE Weibull and its confidence bounds, bootstrapping, and the Brevik Method to predict future machine availability at UCSB, in the Condor pool, and in the Long, Muir, and Golding study.

Data Set	*MLE Weibull*	*Bootstrapping*	*Brevik Method*
UCSB (16 machines)	56.3%	62.5%	**87.5**%
Condor (87 machines)	95.9%	60.2%	**98.9**%
Long/Muir/Golding (83 machines)	58.0%	53.4%	**94.3**%

Table 1. Percentage of predictions made correctly using three different quantile estimation methods to estimate the 0.05 quantile with 95% confidence.

Using the first 20 measurements occurring chronologically from each machine trace, we estimated the lower 95% confidence bound on the 0.05 quantile. This number (which is different for each machine) is the minimum duration of time a scheduler could depend upon for each machine with 95% confidence if the methodology used to generate it is effective. For each data set, we identified the individual machine traces with at least 40 measurements so that the number of predictions made would be at least as large as the number of measurements used to "train" the predictor. The number of machine from each data set fitting this criterion is shown in parentheses in the left-hand column.

We then record the number of future measurements that were greater than the estimated 0.05 quantile and report them as a percentage of total number of predictions for each machine (which is greater than or equal to 20 in all cases). Thus, this experiment depicts the empirical accuracy of each estimation method using the first 20 measurements to predict the remaining measurements where there are at least 20 remaining measurements to predict. Full details from the investigation are described in [11]. From the table, though, it is clear that the Brevik Method is capable of making accurate, non-parametric estimate of future availability using relatively few measurements.

Thus, using the NWS, we have developed two new functionalities that will eventually be incorporated into the distributed software base. The first is an automatic modeling capability that can generate closed-form probability distributions that "fit" empirically observed availability measurements. We believe this functionality will be crucial to the development of realistic, possibly on-line simulations of grid, peer-to-peer, and global computing systems. Secondly, using the Brevik Method, the NWS will be able to provide accurate predictions of future availability levels using relatively few measurements.

9 Conclusions and Future Work

There are several ways in which we are currently extending our work beyond the capabilities described in the previous section. We are studying the decay in forecast accuracy (both in terms of the forecast value and the width of the empirical confidence intervals) as a function of time into the future. The current set of NWS forecasting techniques make predictions for the next time interval. As such, the periodicity with which measurements are gathered defines the time frame for which a forecast is generated. We are attempting to quantify the error associated with multi-step forecasting.

We are also investigating methodologies for automatically deriving the multiplicative factor that is needed to generate a given confidence range. The forecasters themselves are non-parametric, but the confidence interval system requires that the multiplicative factor be specified. We believe that the forecasting system must be able to adapt its parameterization automatically to be truly useful in an engineering context.

Finally, the NWS forecasting methodology does not address the problem of translating resource performance response into an estimate of application performance response. Even if resource performance forecasts were perfect, composing resource performance predictions into an application performance prediction can introduce error. To address this problem, we have been investigating ways to generate automatic correlator functions that relate resource performance forecasts to application performance [45]. The goal of this work is to combine a small number application performance measurements gathered via internal instrumentation with resource performance measurements taken simultaneously from the resources that the application is using. From these

simultaneous application-level and resource-level measurements, we derive a correlator for the application that can be used to predict future application performance from resource performance only.

The problem of modeling and predicting resource performance is central to Computational Grid research. Not only is it critical to effective program and system design, but the engineering of dynamic schedulers and fault diagnosis tools requires on-line access to prediction data as part of the Grid infrastructure. While explanatory models are beginning to emerge, fast statistical techniques applied to real-time performance measurement streams have empirically been shown to be effective. With little added computational complexity, it is possible to make predictions of future performance measurements, and to quantify the error associated with these predictions. The resulting prediction accuracy can be substantially better than simply using the last observed value, or averaging — the two most common methods of predicting future performance from historical measurement data. In addition, it is possible to derive empirical confidence intervals, based on forecast error, for some forms of resource performance response. Our experience, described using a small number of representative examples in this paper, is that these results are general for the resource types we have presented.

One of the unique features of Computational Grid computing is the central role that performance prediction must play with respect to program adaptivity and resource allocation. Despite characteristics that impede rigorous analysis (such as non-stationarity), the work we have described in this paper reflects the degree to which statistical techniques have proved successful as prediction methods in the Grid settings we have so far encountered.

References

1. Abilene. http://www.ucaid.edu/abilene/.
2. B. Allcock, J. Bester, J. Bresnahan, A. L. Chervenak, I. Foster, C. Kesselman, S. Meder, V. Nefedova, D. Quesnal, and S. Tuecke. Data management and transfer in high performance computational grid environments. *Parallel Computing Journal*, 28(5):749 – 771, 2002.
3. B. Allcock, I. Foster, V. Nefedova, A. Chervenak, E. Deelman, C. Kesselman, J. Leigh, A. Sim, and A. Shoshani. High-performance remote access to climate simulation data: A challenge problem for data grid technologies. In *Proceedings of IEEE SC'01 Conference on High-performance Computing*, 2001. `http://www.globus.org/research/papers/sc01ewa_esg_chervenak_final.pdf`.
4. M. Allen and R. Wolski. Adaptive timeout discovery using the network weather service. In *Proceedings of HPDC-11*, July 2002. `http://www.cs.ucsb.edu/~rich/publications/nws-adapt.pdf`.
5. M. Allen and R. Wolski. The livny and plank-beck problems: Studies in data movement on the computational grid. In *Proceedings of SC03*, November 2003.
6. H. Balakrishnan, M. Stemm, S. Seshan, and R. H. Katz. Analyzing stability in wide-area network performance. In *Measurement and Modeling of Computer Systems*, pages 2–12, 1997.

7. F. Berman, A. Chien, K. Cooper, J. Dongarra, I. Foster, L. J. Dennis Gannon, K. Kennedy, C. Kesselman, D. Reed, L. Torczon, , and R. Wolski. The GrADS project: Software support for high-level grid application development. *International Journal of High-performance Computing Applications*, 15(4):327–344, Winter 2001.
8. F. Berman, G. Fox, and T. Hey. *Grid Computing: Making the Global Infrastructure a Reality*. Wiley and Sons, 2003.
9. F. Berman, R. Wolski, S. Figueira, J. Schopf, and G. Shao. Application level scheduling on distributed heterogeneous networks. In *Proceedings of Supercomputing 1996*, 1996.
10. The BOINC project. `http://boinc.berkeley.edu`.
11. J. Brevik, D. Nurmi, and R. Wolski. Quantifying machine availability in networked and desktop grid systems. In *Proceedings of CCGrid04*, April 2004.
12. H. Casanova, G. Obertelli, F. Berman, and R. Wolski. The AppLeS Parameter Sweep Template: User-Level Middleware for the +Grid. In *Proceedings of IEEE SC'00 Conference on High-performance Computing*, Nov. 2000.
13. W. Chrabakh and R. Wolski. GrADSAT: A Parallel SAT Solver for the Grid. In *Proceedings of IEEE SC03*, November 2003.
14. H. Cramer. *Mathematical Methods of Statistics*. Princeton University Press, 1946.
15. K. Czajkowski, S. Fitzgerald, I. Foster, and C. Kesselman. Grid information services for distributed resource sharing. In *Proceedings 10th IEEE Symp. on High Performance Distributed Computing*, 2001.
16. C. Dovrolis, D.Moore, and P.Ramanathan. What do packet dispersion techniques measure? In *Proceedings of Infocom*, April 2001.
17. A. Downey. Using pchar to estimate internet link characteristics. In *Proceedings of ACM SIGCOMM*, September 1999.
18. The Entropia Home Page. `http://www.entropia.com`.
19. I. Foster and C. Kesselman. Globus: A metacomputing infrastructure toolkit. *International Journal of Supercomputer Applications*, 1997.
20. I. Foster and C. Kesselman. *The Grid: Blueprint for a New Computing Infrastructure*. Morgan Kaufmann Publishers, Inc., 1998.
21. I. Foster, C. Kesselman, J. Nick, and S. Tuecke. The physiology of the grid: An open grid services architecture for distributed systems integration. `http://www.globus.org/research/papers/ogsa.pdf`.
22. Globus. `http://www.globus.org`.
23. GrADS. `http://hipersoft.cs.rice.edu/grads`.
24. C. Granger and P. Newbold. *Forecasting Economic Time Series*. Academic Press, 1986.
25. T. Heath, R. Martin, and T. Nguyen. The shape of failure. In *Proceedings of the First Workshop on Evaluating and Architecting System Dependability*, July 2001.
26. The iperf tool: `http://dast.nlanr.net/Projects/Iperf`.
27. V. Jacobson. Congestion avoidance and control. In *Proceedings of SIGCOMM '88*, volume 18, August 1988.
28. R. Jones. The netperf tool: `http://www.netperf.org/netperf/NetperfPage.html`.
29. C. Krintz and R. Wolski. Nwsalarm: A tool for accurately detecting degradation in expected performance of grid resources. In *Proceedings of CCGrid01*, May 2001.

30. W. E. Leland, M. S. Taqq, W. Willinger, and D. V. Wilson. On the self-similar nature of Ethernet traffic. In D. P. Sidhu, editor, *ACM SIGCOMM*, pages 183–193, San Francisco, California, 1993.
31. D. Long, A. Muir, and R. Golding. A longitudinal survey of internet host reliability. In *14th Symposium on Reliable Distributed Systems*, pages 2–9, September 1995.
32. The nsf middleware initiative – `http://www.nsf-middleware.org`.
33. New ttcp: `http://www.leo.org/~elmar/nttcp`.
34. D. Nurmi, J. Brevik, and R. Wolski. Modeling machine availability in enterprise and wide-area distributed computing environments. In *Proceedings of European Conference on Paralell Computing (EuroPar) 2005*, August 2005.
35. The network weather service home page – `http://nws.cs.ucsb.edu`.
36. V. Paxon and S. Floyd. Why we don't know how to simulate the internet. In *Proceedings of the Winder Communication Conference also citeseer.nj.nec.com/paxon97why.html*, December 1997.
37. V. Paxson and S. Floyd. Wide area traffic: the failure of Poisson modeling. *IEEE/ACM Transactions on Networking*, 3(3):226–244, 1995.
38. A. Petitet, S. Blackford, J. Dongarra, B. Ellis, G. Fagg, K. Roche, and S. Vadhiyar. Numerical libraries and the grid. In *Proceedings of IEEE SC'01 Conference on High-performance Computing*, November 2001.
39. The planetLab home page. `http://www.planet-lab.org`.
40. J. S. Plank, S. Atchley, Y. Ding, and M. Beck. Algorithms for high performance, wide-area, distributed file downloads. Technical Report UT-CS-02-485, Department of Computer Science, University of Tennessee, October 2002. `http://www.cs.utk.edu/~plank/plank/papers/CS-02-485.html`.
41. P. Primet, R. Harakaly, and F. Bonnassieux. Experiments of network throughput measurement and forecasting using the network weather service. In *Workshop on Global and Peer-to-Peer Computing on Large Scale Distributed Systems*, May 2002.
42. M. Ripeanu, A. Iamnitchi, and I. Foster. Cactus application: Performance predictions in a grid environment. In *proceedings of European Conference on Parallel Computing (EuroPar) 2001*, August 2001.
43. N. Spring and R. Wolski. Application level scheduling: Gene sequence library comparison. In *Proceedings of ACM International Conference on Supercomputing 1998*, July 1998.
44. M. Swany and R. Wolski. Building performance topologies for computational grids. In *Proceedings of Los Alamos Computer Science Institute (LACSI) Symposium, 2002*, October 2002.
45. M. Swany and R. Wolski. Multivariate resource performance forecasting in the network weather service. In *Proceedings of IEEE SC'02 Conference on High-performance Computing*, November 2002.
46. T. Tannenbaum and M. Litzkow. The condor distributed processing system. *Dr. Dobbs Journal*, February 1995.
47. The TeraGrid Home Page. `http://www.teragrid.org`.
48. S. Vazhkudai, J. Schopf, and I. Foster. Predicting the performance of wide-area data transfers. In *Proceedings of IEEE International Parallel and Distributed Parallel Systems Conference*, April 2002.
49. R. Wolski. Dynamically forecasting network performance using the network weather service. *Cluster Computing*, 1:119–132, January 1998.

50. R. Wolski. Experiences with predicting resource performance on-line in computational grid settings. *ACM SIGMETRICS Performance Evaluation Review*, 30(4):41–49, March 2003.
51. R. Wolski, J. Brevik, C. Krintz, G. Obertelli, N. Spring, and A. Su. Writing programs that run everyware on the computational grid. *IEEE Transactions on Parallel and Distributed Systems*, 12(10):1066–1080, 2001.
52. R. Wolski, N. Spring, and J. Hayes. The network weather service: A distributed resource performance forecasting service for metacomputing. *Future Generation Computer Systems*, 15(5-6):757–768, October 1999.
53. Y. Zhang, N. Du, V. Paxson, and S. Shenker. The constancy of internet path properties. In *Proceedings of ACM SIGCOMM Internet Measurement Workshop*, November 2001.