The Architecture of the Remos System

Peter A. Dinda
Department of Computer Science
Northwestern University

Thomas Gross   Roger Karrer
Computer Science Department
ETH Zurich

Bruce Lowekamp
Department of Computer Science
College of William and Mary

Nancy Miller   Peter Steenkiste   Dean Sutherland
School of Computer Science
Carnegie Mellon University

Abstract

Remos provides resource information to distributed applications. Its design goals of scalability, flexibility, and portability are achieved through an architecture that allows components to be positioned across the network, each collecting information about its local network. To collect information from different types of networks and from hosts on those networks, Remos provides several collectors that use different technologies, such as SNMP or benchmarking. By matching the appropriate collector to each particular network environment and by providing an architecture for distributing the output of these collectors across all querying environments, Remos collects appropriately detailed information at each site and distributes this information where needed in a scalable manner. Prediction services are integrated at the user-level, allowing history-based data collected across the network to be used to generate the predictions needed by a particular user. Remos has been implemented and tested in a variety of networks and is in use in a number of different environments.

1 Introduction

The Remos system was designed to provide resource information to distributed applications. While designing Remos, we considered the needs of many different applications and the capabilities of a range of networking and computing environments. In this paper, we present the architecture of the Remos system and describe how it addresses these problems. The ability of Remos to support resource measurement in a variety of environments and for a variety of applications makes it an appropriate measurement tool for Grid environments.

The design and implementation of Remos addresses the issues of scalability, flexibility and portability needed to support applications in a variety of environments.

- Scalability: Resource monitoring in distributed systems necessarily involves many machines, a large network infrastructure, and many users. Beyond that, however, is the question of scale of a system. To support applications that have wide-area requirements, high-performance cluster requirements, or some combination of both, Remos must be able to provide an appropriate level of detail to meet the information needs of the application without swamping the application with information unnecessary to its requirements.

- Flexibility: Different users require different types of information. For example, synchronous multiprocessing, real-time video, and bulk data transfer have distinctly different bandwidth, latency, and loss requirements, and require that information across different timescales.

- Portability: The variety of operating systems, network architectures, and hardware found in different environments mandates a portable solution that is not specific to any one technology. Even standardized measurement techniques may require alternatives when those techniques are not supported.

Remos is being used in on a regular basis by several groups: the Aura Project (CMU), QuO (BBN), the HiPer-D Testbed (NSWC and S/TDC), CACTUS (University of Arizona), and the Desiderata Project (Ohio University). A number of other sites are also exploring the use of Remos.

Effort sponsored by the Advanced Research Projects Agency and Rome Laboratory, Air Force Materiel Command, USAF, under agreement number F30602-96-1-0287. The U.S. Government is authorized to reproduce and distribute reprints for Governmental purposes notwithstanding any copyright annotation thereon.
The views and conclusions contained herein are those of the authors and should not be interpreted as necessarily representing the official policies or endorsements, either expressed or implied, of the Advanced Research Projects Agency, Rome Laboratory, or the U.S. Government.

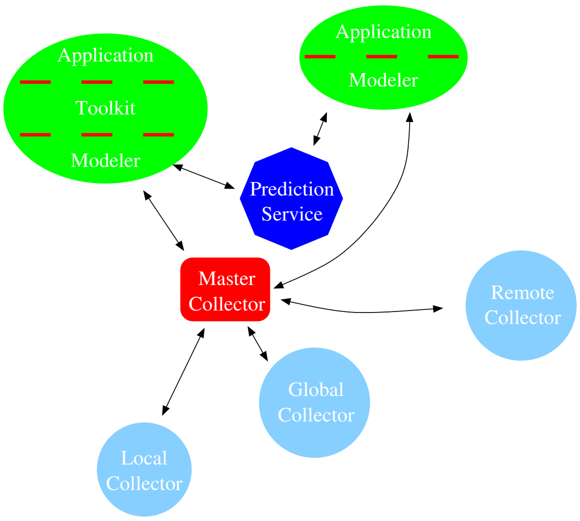

Figure 1. Overview of the components in the Remos architecture.

These projects are quite diverse, both with respect to the networks they use and the application information needs. Our evaluation along the "portability" and "flexibility" dimension is in part based on interactions with these users.

This paper describes how the design of Remos addresses these challenges. In Section 2, we describe the general architecture of Remos. Section 3 describes the techniques used by Remos to implement the architecture. Section 4 describes other work related to Remos and discusses Remos in terms of the Grid Monitoring Architecture proposed by the Grid Forum. Section 5 evaluates how the design and implementation of Remos meet the original design goals. Finally, Section 6 discusses the lessons learned in the development of Remos as well as issues that still require further work.

2 Architecture

An overview of the Remos architecture is presented in Figure 1. The Remos architecture divides the services needed between *collectors*, *modelers*, and *predictors*. The Remos API, which is exposed to applications, is implemented only in the Modeler. This design allows considerable flexibility in varying the design of the other components.

2.1 Collectors

The collectors are responsible for acquiring and consolidating the information needed by the application. Collectors can use a variety of methods of collecting information, e.g. they may incorporate or control *sensors* that perform the actual measurements, but from an architectural view they have a single function: collect information and forward it on to the Modeler. For scalability reasons, collectors can be organized in a hierarchical fashion (Figure 1). At the lowest level, collectors are responsible for collecting information about specific networks. For example, a local collector is responsible for obtaining performance information about its LAN. Global collectors are responsible for obtaining performance information about the networks connecting LANs. Local or global collectors at remote sites can be contacted to obtain information about those remote sites.

The Master Collector is responsible for gathering information from different collectors and coalescing it into a response to a modeler's query. The Master Collector maintains a database of the locations of other collectors and the portion of the network for which they are responsible. When a request comes from a modeler, the Master Collector queries the appropriate collectors and replies without revealing that the response was obtained from multiple collectors. Using this technique, it is possible to build several layers of collectors. For example, the remote collector in Figure 1 might be another Master Collector that in turn contacts a variety of local collectors when queried about its network.

One important advantage of this architecture is that it blurs the line between inter- and intra-site measurements. Because the collectors assume responsibility for contacting remote sites and for aggregating all available information into a single response, neither the Modeler nor the application must determine whether the query concerns nodes at a single site or at remote sites or consider the most appropriate measurement technique. If the WAN link is the only bottleneck along the path of the query, then the appropriate measurement will automatically be returned.

2.2 Modelers

Modelers provide the Remos API to the application and communicate with a collector to obtain information needed to respond to queries made through the API. Because the collectors exist only to obtain network resource information, the Modeler is responsible for the processing necessary to convert this information into a form of interest to the application. For example, the available bandwidth along a path may be provided by the collector as several separate measurements, but the Modeler reports only the bottleneck available bandwidth to the application. Similarly, if an application makes a topology query, the Modeler performs additional processing on the topology returned by the collector to eliminate unnecessary information and present the topology to the application in a more manageable form.

If a prediction is needed, the Modeler also acts as the intermediary between the collectors and the prediction service.

2.3 Predictors

Predictors are responsible for turning a measurement history into a prediction of future behavior. Predictors can operate in a client-server mode, turning a vector of measurements into a single vector of predictions, or in a streaming mode, transforming a stream of measurements into a stream of (vector-valued) predictions. The advantage of the client-server form is that it is stateless, while the advantage of the streaming mode is that a single model fitting operation can be amortized over multiple predictions. The trade-offs between the two modes are complex and both are useful in practice.

Figure 1 shows how client-server predictors fit into the Remos architecture. This location is the appropriate choice for several reasons. First, because Remos collects information on a component basis and later assembles it into a single response, prediction often cannot be done at the lower layers. For example, if the behavior of different resources were correlated, as might commonly be found among networking components, then predicting their behavior independently before aggregating the result would lose information. Secondly, because it is possible to specify different options for prediction, such as the amount of history to consider, the time granularity, or time in the future for which a prediction is needed, it is sometimes difficult to perform predictions at a lower layer before the application's request is known. Finally, prediction can require a substantial amount of computational power, and this architecture allows us to place the burden of performing these predictions near the particular application requesting the prediction.

For environments where predictions can be shared, streaming predictors offer the ability to amortize the cost of prediction over several consumers. Streaming predictors operate in tandem with collectors. For example, a collector may periodically measure load on a particular host, choosing its sampling rate as appropriate for the dynamics of the host. As each sample became available, it would be fed to a directly attached streaming predictor. The collector would then make these predictions available to modelers that were interested. Although this aspect is not integrated into the current Remos implementation, being able to chose between client-server and streaming predictors may be a significant feature for supporting both a variety of applications and minimizing unnecessary work where appropriate.

3 Implementation in Remos

The current implementation of Remos is diagrammed in Figure 2. This figure illustrates how the various collectors used by Remos interact when used in a grid-like environment. In the remainder of this section we discuss the Remos components in more detail.

3.1 Collectors

The motivation and design of the overall collector architecture is described in a paper by Miller and Steenkiste [19]. In this section we briefly describe the main features of the different Remos collectors.

Collectors can be classified along three primary axis:

- *How they collect information.* Collectors in Remos either use SNMP to collect information directly from routers and switches, or they use explicit benchmarking.
- *The type of network they are responsible for.* Remos has collectors for local-area and wide-area networks, and a collector for wireless LANs (802.11) is under development. The network type affects critical parameters such as what what information is interesting (e.g. latency is rarely of interest in a LAN), level of traffic aggregation (can affect prediction), and administrative privileges of the collector (can limit the use of SNMP).
- *How the collector operates.* The two primary modes are on-demand, i.e. the collects information when it receives a request, or periodic, i.e. the collector collects information at periodic intervals. We expect all collectors to aggressively cache information to reduce overhead. Remos collectors typically operate in periodic mode, since this offers better user response time.

3.1.1 SNMP Collector

The SNMP Collector is the basic collector upon which Remos relies for most of its network information. SNMP is a database protocol designed to provide network administrators with direct access and control over the status of network devices. These features can also be used to obtain network-level information about topology and performance directly from routers and switches. Because the SNMP Collector has direct access to the information the network itself stores, this collector is capable of answering the flow and topology queries that require an understanding of the details of the network's structure [19]. The SNMP Collector operates on routed networks (level 3).

An SNMP Collector is assigned to monitor a particular network, generally an IP domain corresponding to a university or department. Because SNMP agents are normally

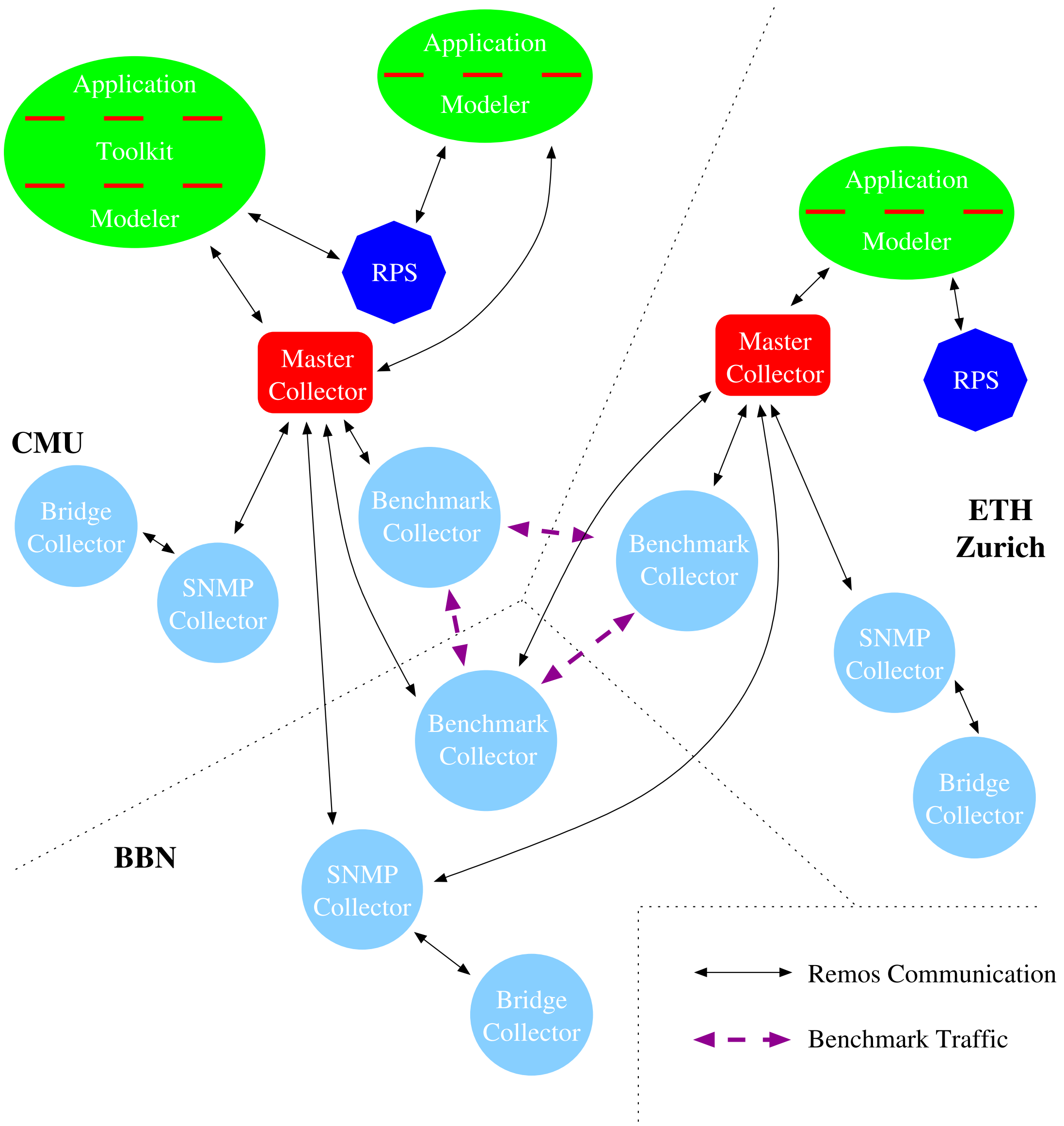

Figure 2. A detailed illustration of how the components of the Remos architecture are connected. Shown here are applications running at CMU and ETH making use of resources at CMU, ETH, and BBN. Each application is using prediction services to provide information about the future network availability. The applications at CMU are using machines at CMU and BBN, and the application at ETH is using machines at ETH and BBN. The benchmark measurements sent across the Internet are shown, but, for clarity, the connections between the SNMP and Bridge Collectors, and the network components they monitor are not shown.

only accessible from local IP addresses, this provides a natural partitioning for the location of the SNMP Collectors.

The SNMP Collector initially monitors the network on an on-demand basis. It waits for queries, then explores and begins monitoring the network components needed to respond to that query. Once it begins monitoring parts of the network, it will continue with periodic monitoring to collect history of that network for use in predictions. A logical extension for the collector would be to configure it to begin monitoring specific resources at startup, for use in a computational center, etc.

The first and most complex step the SNMP Collector must take upon receiving a query is topology discovery. Using the IP addresses of the nodes in the query and the routers they are configured to use, the collector follows the route hop-to-hop between each pair of nodes in the query. The collector caches previously discovered routes, so it must only follow new routes until it reaches a previously explored route.

Once the collector has discovered the routes used between the nodes, it queries the routers along the path for the link bandwidth between each pair of routers. It then periodically monitors the utilization of each segment by querying the octet counters for each interface on the routers. By default, the utilization is monitored every five seconds, although this is a configurable parameter.

The final responsibility of the SNMP Collector is representing the network with a virtual topology graph. When the collector discovers nodes connected to a shared Ethernet, or connected to routers it cannot access, it represents their connection with a virtual switch. In the case of shared Ethernet, this switch can be annotated with the bandwidth capacity and utilization of a shared Ethernet, representing its functionality with a standard graph format.

The SNMP Collector is implemented with Java threads, so it is capable of monitoring a number of routers and responding to many queries simultaneously.

3.1.2 Bridge Collector

The SNMP Collector by itself is only capable of monitoring level 3 routed networks. While many research networks, as well as campus networks are connected using only routers, the majority of LANs are implemented using level 2 switched Ethernet. Although the techniques used by the SNMP Collector for monitoring link capacity and utilization are identical on routers and switches, Ethernet switches to not provide explicit topology information as is provided by the IP routing tables. To collect this information, the Bridge Collector is used to determine the topology of the Ethernet LAN through queries to the forwarding database in the Bridge-MIB of each bridge or switch [18].

At startup, the Bridge Collector queries all components of a bridged Ethernet to determine its topology, then stores this information in a database. When a query is made to the SNMP Collector, the Bridge Collector provides it with the portion of the level 2 topology that is needed to fulfill the query, such as providing the path between pairs of nodes, or between a node and the edge router of the Ethernet LAN. Once it has the topology information, the SNMP Collector can collect dynamic information from the switches, and will cache the topology of that portion of the Ethernet network.

In general, bridged Ethernet networks have fairly static topologies. However, over the lifetime of the network, we expect that new nodes might be added to the network, or existing nodes may move. Extremely rarely, a switch may be moved or an additional switch added. In wireless networks, however, a mobile node may move between basestations much more frequently, which will change its location in the topology. To account for this node movement, the Bridge Collector must monitor the location of nodes on the network continuously. The location of a host can be monitored merely by checking its forwarding entry in the bridge to which it is connected. Verifying the location of bridges and switches is somewhat more difficult, however, we do not anticipate bridges or switches being moved frequently in typical networks.

3.1.3 Benchmark Collector

While SNMP offers excellent information, Remos generally cannot obtain SNMP access to network information for WANs or other networks where the Remos administrator does not have an account on a machine. In that case, we fall back on a Benchmark Collector, that does explicit testing to determine the performance characteristics of the network. A Benchmark Collector is run at each site where an SNMP Collector is. When a measurement of performance between multiple sites is needed, the Benchmark Collector exchanges data with the Benchmark Collector running at the other site of interest. By measuring the rate at which the data travels across the network, the Benchmark Collectors determine the performance of the links connecting the network and report this information to the Master Collector. Details and analysis of this implementation, in particular, is described by Miller and Steenkiste [19]. This technique is similar to the techniques used by NWS [33].

The collectors' Java implementations and their reliance on the SNMP standard and simple benchmarks have made them very portable, running on a wide variety of host architectures and supporting most network components.

3.1.4 Master Collector

Because SNMP and Benchmark Collectors monitor only a particular portion of the network, distributed applications cannot obtain all of their information from these collectors. The Master Collector is designed to solve this problem. Despite the name, a different Master Collector is used in each network where Remos applications are running.

The Modeler used by the Remos application submits a query to its Master Collector. The first task for the Master Collector to solve is identifying the IP networks and subnets needed to answer the query, along with the associated SNMP and Benchmark Collectors for those networks. The Master Collector identifies the networks containing hosts used in the query, as well as any intervening networks connecting those with the hosts. The database used is very similar to the SLP directory, and SLP may be used by the Master Collector in the near future [29].

Once the relevant networks have been identified, the Master Collector divides up the query and passes the relevant portion to the collectors responsible for the identified networks. When the responses are received from those collectors, the Master Collector combines them into one single response and returns that response to the Modeler that made the query.

3.2 Modeler

The Modeler provides the API to the user and communicates with the local Master Collector to obtain the information needed to answer the user's query. It is single-threaded and communicates with the Collector over a TCP socket, using a simple ASCII protocol. Once it receives the response from a collector, it is also responsible for inserting virtual switches to simplify the topologies returned by the collector. Because currently only topologies are exchanged between the Modeler and collector, the Modeler also performs max-min flow calculations on the Collector's topologies to determine solutions to flow queries. By connecting a different Modeler to each application, the modeler architecture provides the flexibility needed to support the information needs of different applications by allowing the information obtained from the Collector to be interpreted and predicted in different manners according to the application. The Modeler is implemented in both C and Java.

3.3 Predictor

If predictions are necessary, the Modeler uses Dinda's RPS Toolkit [9]. The relationship between RPS and Remos is somewhat complex, as each is an independent system. In this context, RPS provides prediction services and host measurement services to Remos, while Remos provides network measurement services to RPS. Remos can use RPS's client-server interface for general purpose predictions, and its streaming interface for predictions about hosts and flows for which streaming predictors have been instantiated.

In the current implementation, Remos relies on RPS collecting data itself to establish the performance history needed to make predictions. RPS does this through a host load sensor and a network flow bandwidth sensor (the latter is itself a Remos application). This is due to a limitation in the first protocol designed for use with the collectors. We are transitioning to an XML over HTTP protocol that will allow the modeler to query the collectors for a history of resource measurements. In this architecture, the collectors will be responsible for maintaining history information for each component they monitor, and will thus be able able to use RPS's client-server interface as well.

Currently, RPS's time series prediction library includes the Box-Jenkins linear time series models (AR, MA, ARMA, ARIMA), a fractionally integrated ARIMA model which is useful for modeling long-range dependence dependence such as arises from self-similar signals, a "last value" model, a windowed average model, a long term average model, and a template that creates a periodically re-fitting version of any model. More details on the implementation and performance characteristics of these models are available elsewhere [9].

While RPS's model selection is quite extensive, covering almost all approaches to linear time series prediction, it does not yet include nonlinear models such as TARs. The choice of models (system identification) is a complex topic in general [4, 5, 1, 28] and also within the context of distributed systems. We have found AR models of order 16 or better to be appropriate for prediction of host load [10], despite load's complex behavior [8]. Others have also found that simple models are sufficient for host load [32]. Once a model has been chosen, fitted to historical data, and is in use, its error must be monitored to verify that the fit continues to hold. In RPS, this continuous testing (done by the evaluator) is used to decide when the model must be refit. In contrast, the Network Weather Service uses similar feedback to decide which of a set of models to use next in a variant of the multiple expert machine learning approach [31].

4 The Grid and related work

Grid-based distributed computing has brought about the need for systems that monitor and predict both application and resource information. In addition to Remos, a number of systems have been developed that address various information needs of grid applications [31, 26, 21, 27]. One of the principle differences between Remos and these systems is that Remos was intended to provide applications with end-to-end data derived from component sensors across the network, and integrate these measurements with traditional sensor-based data and end-to-end benchmarks.

While other projects have developed techniques to derive Internet topology [11, 24, 15, 20], Remos is the first to integrate LAN topology information with performance measurements. Because the link-sharing found on LANs can have a profound influence on an application's performance, providing this information as well as site-to-site performance measurements has proven useful for predicting application performance.

Research into resource prediction has focused on determining appropriate predictive models for host behavior [10, 32, 22], and network behavior [30, 2, 12]. The RPS toolbox used by Remos incorporates many of the models studied by this research. RPS is also available as an independent tool for other research requiring predictive models.

One of the products of the Grid Forum is the Grid Monitoring Architecture [25], which is being developed by the performance working group. In this architecture each Collector is a producer. The Master Collector is a joint consumer/producer, as its responsibility is to contact the other collectors as a consumer, before aggregating the information together and providing it to another layer. Although we view the Modeler as a consumer, it could also be another joint consumer/producer, providing end-to-end performance predictions using the component data available from the collectors as a service to other applications. In the Remos architecture, the collectors also implement a limited form of directory service to locate each other. The directory service of the GMA would be natural to use for this purpose.

Overall, we find that the Remos architecture is quite compatible with the GMA, and should interoperate well with other monitoring systems once appropriate interfaces are designed. The biggest challenge presented by Remos is describing the information available through it in the directory service. Because Remos provides end-to-end data derived from component-level data, it would be difficult to describe all possible measurement pairs in the directory service. However, several solutions to this problem have been discussed by the Grid Performance Group and others, therefore we are confident that the GMA's directory service will support Remos well when fully developed.

Associated with the format of the GMA is the method used to store grid information in the first place. Significant discussion is ongoing about the advantages and disadvantages of a hierarchical approach, such as MDS-2 [6], or a relational approach [7]. Both proposals present models that are capable of associating Remos with the resources it monitors, which is the fundamental requirement Remos has for a directory service.

5 Evaluation

The flexibility and portability aspects of Remos have been discussed in other Sections, especially Section 3. Here, we discuss scalability and functionality results for the different Remos components and the system as a whole.

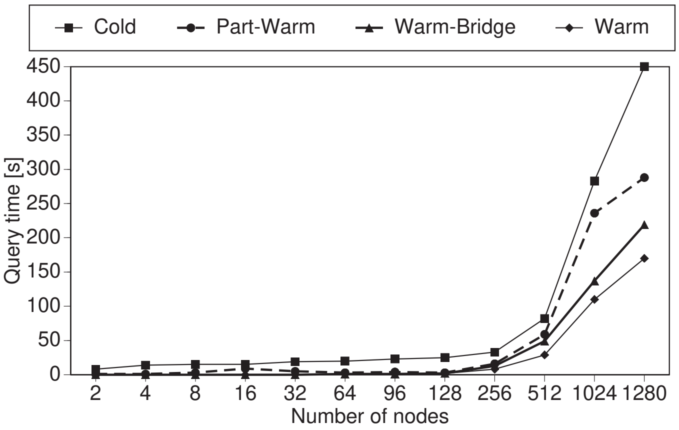

Figure 3. LAN collector response time

5.1 LAN scalability

In a first set of experiments, we look at the response time of the SNMP Collector deployed in the local area network in the School of Computer Science at CMU. The network is a very large bridged network and in the experiments, the Bridge Collector is already running and the SNMP Collector periodically collects dynamic information for all nodes in its cache with a period of 5 seconds.

Figure 3 shows how the response time increases with the number of nodes specified in the query. For all measurements, most of the elapsed time is spent in the SNMP Collector. There are three scenarios:

- *Cold cache*: the SNMP Collector has just started up so it has no information on either the static topology or the dynamic performance metrics.
- *Mixed*: the SNMP Collector has some cached information, namely the result from the previous query (typically about 1/2 or 1/3 of the data).
- *Warm cache*: the SNMP Collector has both the static and dynamic data in its cache.

We can make a number of observations. First, it clearly pays off to cache information. The warm-cache results are a factor of three or more better than the cold cache results. Second, the worst case cost of a cold cache query is $O(N^2)$. However, we implemented a number of optimization that reduce the cost, especially for large N; the measurements show the effect. Finally, the cost of warm-cache queries should be $O(N)$. We see that the cost actually grows faster, probably because of increasing memory requirements which reduce execution efficiency.

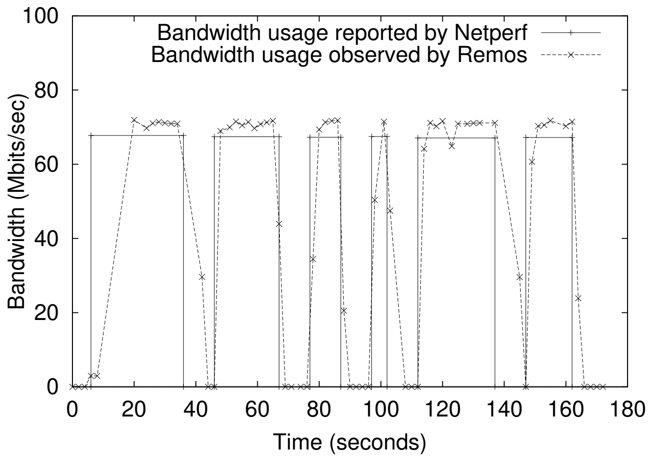

Figure 4. SNMPColl accuracy: 2 sec. interval

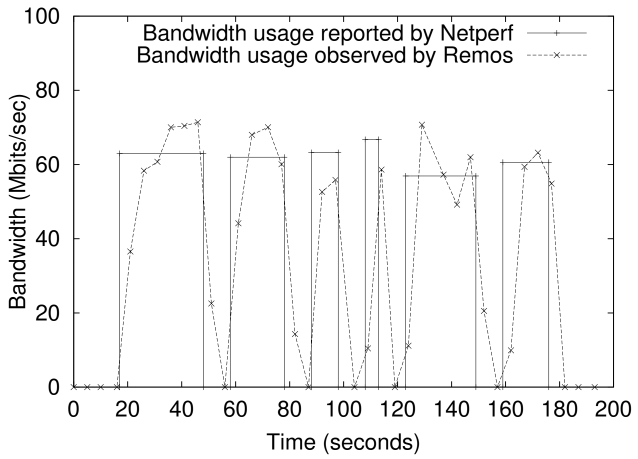

Figure 5. SNMPColl accuracy: 5 sec. interval

5.2 SNMP Collector accuracy

We ran an experiment on our private networking testbed to determine the accuracy of the SNMP Collector in tracking changes in network bandwidth. We used Netperf to generate bursts of TCP traffic of varying lengths between two endpoints on our testbed. The endpoints were separated by two 933MHz routers running FreeBSD. The SNMP Collector was set up to gather data about the links between these hosts at several different sampling intervals: 5 seconds, 2 seconds, and one second. Figure 4 compares the end-to-end bandwidth reported by Netperf with the bandwidth reported by the SNMP Collector measured in 2 second intervals. Figure 5 shows Netperf bandwidth compared to bandwidth measured by the SNMP Collector at 5 second intervals.

There is a fairly good match between the results. The

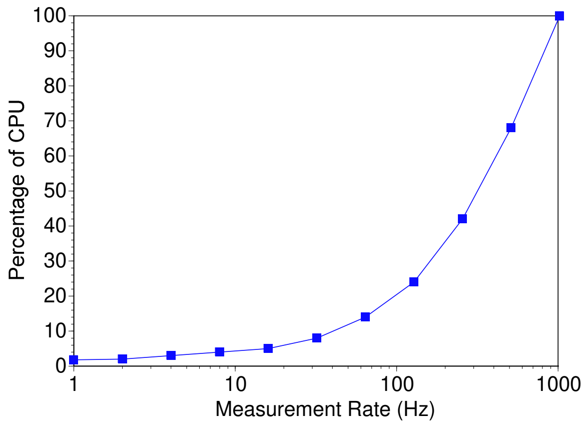

Figure 6. CPU usage of RPS-based host load prediction system as a function of measurement rate. The appropriate AR(16) predictive model is being used.

expense of tracking bandwidth more closely can create inconsistencies in the data and put added strain on network routers. In practice, we've found that a sampling interval of 5 seconds seems to be a good default for most applications.

5.3 RPS evaluation

The RPS prediction error is highly dependent on the characteristics of the data that is to be predicted and the prediction horizon. For host load, AR(16) predictors produce one-second-ahead error variances that are 70% lower than raw signal variance, and provide benefits out to at least 30 seconds. Much more detail on host load prediction is available elsewhere [10]. RPS also characterizes its own prediction error, and that characterization is usually quite accurate regardless of the data. This in large part due to the feedback in the system.

RPS prediction systems have quite low latencies and overheads, and can operate at high rates. The RPS-based host load prediction system that the Remos Modeler currently interfaces with has a latency from measurement to prediction of 1-2 ms and can operate at a rate in excess of 700 Hz on a 500 MHz 21164 Alpha machine. Figure 6 shows CPU usage of this system, using the appropriate AR(16) model, as a function of the measurement rate. At 1 KHz the CPU is saturated and the latency grows. At the normal 1 Hz rate, CPU and network usage is negligible.

In a separate experiment, we were able to run a Remos query for a single flow at about 14 Hz using the SNMP Collector, which itself typically makes SNMP queries at a $1/5$ Hz rate. At such rates, the overhead of RPS with an AR(16) or similar predictive model is in the noise.

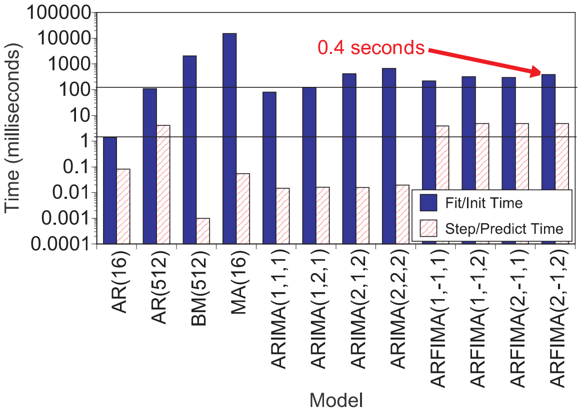

Figure 7. CPU time needed to fit/init and step/predict different RPS predictive models.

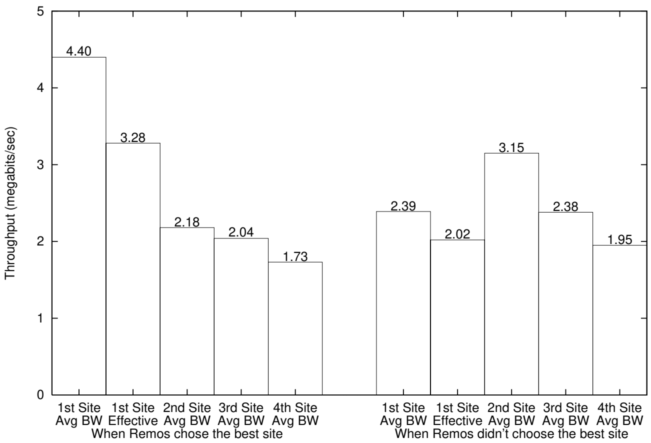

Figure 8. Average transfer rates for well-connected sites

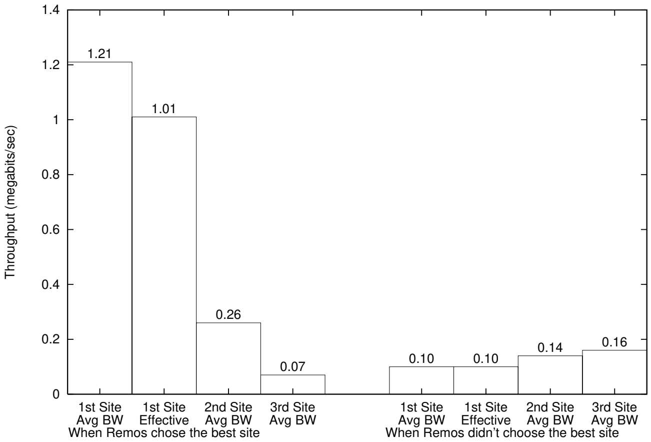

Figure 9. Average transfer rates for poorly-connected sites

However, the resource demands of an RPS system are drastically affected by the model used in prediction. RPS's models vary over four orders of magnitude in their computational costs. Figure 7 shows the costs involved with using some of the different models. These are broken down into a "fit/init" cost, which is the cost to fit the model to a sequence of samples (600), and a "step/predict" cost, which is the cost to push one new sample through the fitted model, producing one set of predictions. The variation in cost is important because the appropriate predictive models for other kinds of resources (network bandwidth, for example) are unknown at this time.

In a streaming implementation, such as in the host load prediction system, the fit/init cost can be amortized over multiple samples. However, to do this RPS must keep around per-stream state, and must pay the step/predict cost every time a new measurement becomes available. In the client-server interface that RPS also provides to Remos, the fit/init and step/predict costs are paid every time a query is made. However, the RPS request-response prediction system is stateless and computation happens only in direct response to queries. A more detailed evaluation of the overheads of RPS is available elsewhere [9].

5.4 Mirrored server experiment

One simple use of Remos is to help applications choose a remote server based on available network bandwidth. We have written a simple application that reads a 3MB file from a server after using network information obtained from Remos to choose the best server from a set of replicas [19].

We ran two sets of mirror experiments: one that used remote sites with good network bandwidth, and another experiment using sites with poor bandwidth. For the first experiment, we ran the application at Carnegie Mellon and servers at Harvard, ISI, Northwestern University (NWU), and ETH. Averaged over all 108 trials, we observed an average throughput of 2.03 Mbps from Harvard, 2.15 Mbps from ISI, 4.11 Mbps from NWU, and 1.99 Mbps from ETH. For the second experiment, we ran the application at Carnegie Mellon and the servers at the University of Coimbra, Portugal (average throughput 0.25 Mbps), the University of Valladolid, Spain (average throughput 1.02 Mbps), and the third server was run on a machine in Pittsburgh connected via a DSL link with a maximum upstream bandwidth of 0.08 Mbps. We ran 72 trials using the poorly connected sites.

In order to be able to evaluate the quality of the Remos information, we modified the application to read the file from all three servers, starting with the server that, according to Remos, has the best network connectivity. In the first experiment using well connected sites, Remos chose the remote site that ended up having the fastest transfer rate 83% of the time. Figure 8 shows the difference in throughput between the 1st place site Remos chose and the other 3 sites. The left half of the graph shows the throughput when Remos chose the best site, and the right half of the graph shows the throughput when Remos did not choose the fastest site. The second bar in each group shows effective bandwidth for the site Remos chose. This bandwidth includes the time it took to get an answer back from the Remos system.

In the second experiment, which used sites that were not well connected to CMU, Remos chose the remote site that ended up having the fastest transfer rate 82% of the time. Figure 9 shows the difference in throughput between the 1st place site Remos chose and the other 2 sites. As in Figure 8, the left half of the graph shows throughput for when Remos chose the best site, and the right half shows throughput for when it didn't. The second bar in each group once again shows the effective bandwidth for the site Remos chose.

We included the effective bandwidth measurement to show that even though it takes some time to consult Remos to choose a server, performance is still better than choosing one of the slower sites. These experiments also show that using Remos to pick a site is effective even when all of the sites have poor connectivity.

5.5 Application Experiment—Video transfer

In the previous example, Remos used the available bandwidth as a metric. This metric, however, does not always directly correspond to the metric in which the application is interested. For example, the quality of a video application that downloads and plays the video in real time may be rated by the number of correctly received frames at the client [14]. This experiment shows how the Remos metric corresponds to such an application-defined metric.

For the experiment, the video client is located at ETH.

Server Location	average bandwidth	standard deviation
ETH Zurich	63.1	5.61
EPFL Lausanne	3.03	0.17
CMU	0.50	0.28
University of Valladolid, Spain	0.37	0.28
University of Coimbra, Portugal	0.18	0.07

Table 1. Server location, the available bandwidth and the standard deviation, measured by Remos.

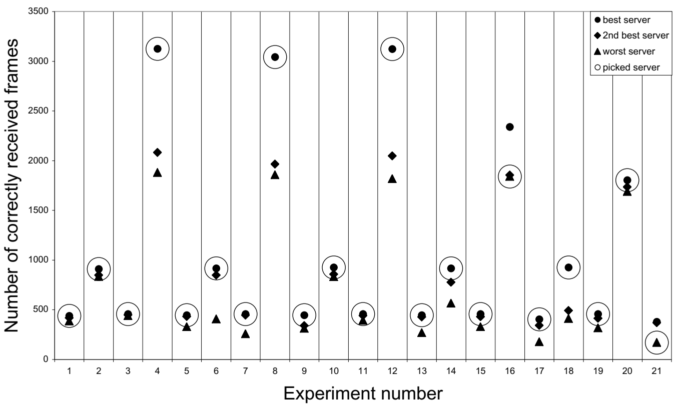

Figure 10. Server picked according to the measured bandwidth (large circle) and the number of correctly received frames in the following download.

Servers from which the videos can be downloaded are placed at different locations in Europe and the U.S (see Table 1). The video server is able to adapt the outgoing video stream to the available bandwidth by intelligently dropping frames of lower importance [14]. It thereby maximizes the numbers of frames that are transmitted correctly.

The bandwidth of the local server at ETH is an order of magnitude higher than EPFL, which in turn is an order of magnitude larger than the others.

Before downloading a video, the client issues a Remos query to measure the available bandwidth to all servers. It then downloads the movie from the server with the best connectivity. To compare the results, the client subsequently also downloads the same video from all other sites in the decreasing order of the available bandwidth. This experiment was run several times within 24 hours with different movies.

Figure 10 shows the number of correctly received frames for each experiment. The server that is selected first according to the bandwidth measurements by Remos is indicated

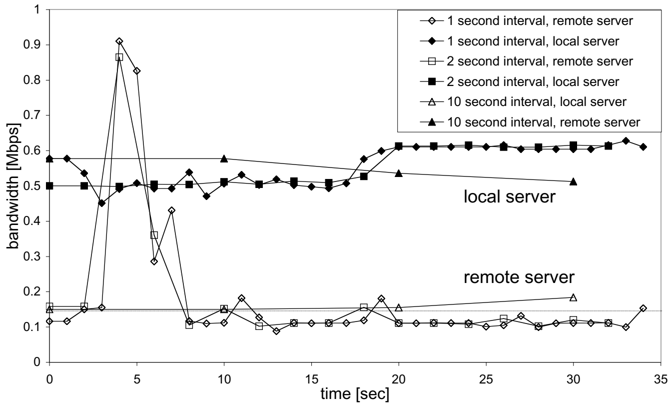

Figure 11. The bandwidth measured by the application, averaged over different time intervals, and the bandwidth reported by Remos.

by a large circle. The figure excludes the results from ETH and EPFL because the bandwidth is always higher that the bandwidth required by the application. If ETH is included, the client always picks the server at ETH. The downloaded video does not lose any frames. If ETH is excluded, the system always selects EPFL and also gets the video without dropped frames. If both ETH and EPFL are excluded, the client-perceived quality corresponds to the reported bandwidth in 90% of the cases, i.e. the client receives the most frames correctly from the server with the highest bandwidth. In the 2 cases where the best server is not picked, an inspection shows that the server only sent about half of the packets, probably due to a high load on the server.

The results show that the available bandwidth corresponds well to the application-perceived quality. However, the two wrong picks indicate that the bandwidth alone does not guarantee a good video download. Other parameters may influence the download as well and must be taken into account.

Figure 11 shows 2 experiments in detail. The same movie is downloaded from 2 different servers, a local server with a high- bandwidth connectivity and the remote server with a limited bandwidth. Each packet that arrives at the client is timestamped and the application-perceived bandwidth is calculated as the average over 3 different time intervals: 1, 2 and 10 seconds.

The download from the local server is not limited by the bandwidth. The average over small intervals shows that the bandwidth requirements vary over time. These fluctuations can be explained by the variation of the movie content. Averaging the bandwidth over a larger interval smooths the variations.

For the remote server experiment, the bandwidth measured by Remos is the horizontal line at 0.15Mbps. This line corresponds well to bandwidth measured by the application if it is averaged over a large interval. The 10 seconds interval corresponds to the time interval that Remos uses to measure the available bandwidth. Calculating the average over smaller intervals shows higher fluctuations. The reported bandwidth does not correspond well to the bandwidth of these small intervals.

This experiment demonstrates that optimal results can only be achieved when not only the metric of Remos and the application correspond, but also when the interval over which the bandwidth is reported matches the varying needs of the application. Although Remos is not currently able to fully address these points, this experiment still shows that Remos is well able to provide useful guidance to this type of application. It can help the video client to select the server. In addition, it might similarly be used to determine alternate servers and routes for a dynamic video handoff [16].

6 Reflections

In this section we try to capture what we learned about resource monitoring systems in the last four years. While these comments are of course quite subjective, we hope our thoughts will help others working in the same area.

6.1 What worked

The first step in the Remos development was the definition of the Remos API [17]. The API supports *topology queries* that provide users with network utilization information in the form of a virtual topology, and *flow queries* that return predictions of the performance that a new flow can expect. While the API supports several performance metrics, our initial implementation focused on bandwidth. Our experience suggests that these were the right design decisions. The API provides a good balance between simplicity and amount of information provided. The API works for all the networks we have encountered so far, i.e. it is network independent. Finally, bandwidth is by far the most important metric for most applications.

Underneath the fixed API, we decided to use a systems architecture that was modular and extensible. This choice also worked well. Our initial system consisted of just an SNMP Collector, and later we were able incorporate Benchmark, Bridge, and Master Collectors, without changes to the API. Because of the modular design, we were also able to use different data gathering techniques for different networks. While benchmarks are an effective way of collecting bandwidth information, it is too expensive and intrusive for many types of networks, and we need to utilize more lightweight techniques such as the SNMP Collector.

Finally, our experience with host load prediction suggests that it pays off to build on rigorous time series prediction theory. This effort not only allowed us to leverage existing experience and tools, but it also had more concrete benefits. For example, we can characterize variance, which applications need to make decisions based on the predictions. One catch is that applying time series models requires collecting periodic measurements, which is sometimes hard. We are continuing to evaluate similar techniques for network load.

6.2 What needs more work

We discovered that one of the most difficult challenges in building a resource monitoring system is making the system easily portable and robust across diverse environments. Our goal is that Remos must be able to report resource information for any networked environment, with minimal, if any, manual configuration. In practice, we discovered that bringing up Remos in a new environment can be challenging. Problems range from: network features that we had not encountered before (e.g. VLANs), and network elements that were misconfigured or have non-standard features (e.g. non-standard SNMP implementations). To some extend, these portability problems should not be a surprise: there are many network vendors and many ways to configure a network, so this problem is inherently hard. A related issue is that Remos currently assumes a fairly static environment, so network failures and host movement can confuse Remos. Improving the robustness and portability of Remos is an ongoing effort.

Remos currently relies on SNMP MIB and benchmark information. Many other sources of information could be tapped, including measurements collected by ISPs for traffic engineering purposes, application-level information [23], and network information that is collected in vendor-specific ways. Also, for certain types of networks, such as shared Ethernets, we need better techniques for performance prediction.

There are many ways in which the Remos system could be improved. We are currently working on updating the communication between Remos components. The initial implementation used a simple text format that we would like to replace with an XML format using HTTP as a communication protocol. This change would give us much more flexibility in the kinds of data we can exchange between components, and it would also allow new collectors and other pieces to be added more easily. In particular, the XML format will enable us to send an entire history of network measurements to the RPS subsystem for prediction purposes.

As network technology advances, we must modify existing collectors or add new collectors to discover information in networks with new kinds of protocols and hardware. In particular, we are working on a new collector for wireless networks, and improving our existing collectors to support mobile hosts. A related extension is to allow the system to operate though it may not know how to handle all the components in a network.

Our Benchmark Collector could be improved by adding support for other kinds of benchmarking programs, especially those that put as little load on the network as possible, and those that do not require both a source and a sink to measure network information. We would also like to provide information about metrics other than bandwidth, for example network jitter, which could benefit multimedia applications.

The current Remos system is able to provide a wealth of information at a rate that suffices for many applications. However, as processor and network speeds improve, applications may demand more frequent updates on the changing environment. An issue that has not yet been explored is how far this architecture scales in the performance domain – how high a rate of requests could be satisfied.

Finally, there are a number of issues that we have not looked at in depth, including an evaluation of techniques for caching and sharing of prediction results, dealing with non-TCP traffic (that claims to be "TCP-friendly"), and a characterization of non-network effects that influence application network performance.

6.3 When is Remos most useful?

Many applications (e.g., video streaming) only care about the performance of a single flow between two nodes that are currently exchanging data. In such cases, Remos is probably overkill, because the application can get the required information more cheaply and more accurately by monitoring its own performance [3]. However, for applications that have to select a server from a set of options, that have to select and assign a set of compute nodes with certain connectivity properties, or that have to make critical configuration decisions (e.g. to use remote or local execution, to use video plus audio, or audio only), Remos provides explicit connectivity information that would be difficult and expensive to collect otherwise [13].

We end up with a model of an adaptive application that combines two types of adaptation using different information sources. The application performs node and network selection, and high-level self-configuration based on explicit, Remos-provided resource information. This type of decision is typically made when the application starts up, or, for long running applications, periodically during execution. During execution, the application can fine-tune its performance based on direct measurements. This model is in part driven by the cost of adaptation: adaptation that does not involve changes in node usage can be cheap and fast, while changing nodes or high-level application configuration will be more expensive.

7 Conclusions

The Remos architecture is designed to provide the information needed by Grid applications across many diverse environments. Remos has been implemented and tested in a variety of different networking environments and has been used to support a variety of applications, thus demonstrating the flexibility and portability needed for emerging applications. We have used Remos to support both large numbers of machines at a single site as well as to support several sites simultaneously and find that the architecture scales well. While our architecture differs somewhat from the proposed Grid Monitoring Architecture (GMA), a comparison indicates both that Remos should interact well with GMA-based monitoring tools and that the future development and performance of tools such as Remos will be easily supported within the framework of the GMA.

The availability of the Remos API allows application developers to address new aspects of the environment. Without sacrificing portability for performance (or vice versa), it is now possible to develop applications that use information about the status of the network to determine the next adaptation steps. The availability of and experience with the Remos architecture backs up the claims made by the Remos API and provides a practical demonstration that it is possible to find a workable compromise between the conflicting objectives of functionality, performance, and portability. As networks grow in complexity, and as efforts like the Grid bring more application developers into this domain, the interest in infrastructure systems like Remos is likely to increase. Dealing with and obtaining performance information will remain an important topic; Remos provides both a set of abstractions and an architecture that have proven their value in practical settings.

References

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

[2] S. Basu, A. Mukherjee, and S. Klivansky. Time series models for internet traffic. Technical Report GIT-CC-95-27, College of Computing, Georgia Institute of Technology, February 1995.

[3] J. Bolliger and T. Gross. Bandwidth monitoring for network-aware applications. In *Proc. 10th IEEE Symp. High-Performance Distr. Comp.*, San Francisco, CA, August 2001. IEEE CS Press.

[4] G. E. P. Box, G. M. Jenkins, and G. Reinsel. *Time Series Analysis: Forecasting and Control*. Prentice Hall, 3rd edition, 1994.

[5] P. J. Brockwell and R. A. Davis. *Introduction to Time Series and Forecasting*. Springer-Verlag, 1996.

[6] K. Czajkowski, S. Fitzgerald, I. Foster, and C. Kesselman. Grid information services for distributed resource sharing. In *Proceedings of the Tenth IEEE International Symposium on High Performance Distributed Computing (HPDC 10)*, August 2001.

[7] P. Dinda and B. Plale. A unified relational approach to grid information services. GWD-GIS-012-1. http://www.cs.northwestern.edu/˜pdinda/relational-gis/, February 2001.

[8] P. A. Dinda. The statistical properties of host load. *Scientific Programming*, 7(3,4), 1999. A version of this paper is also available as CMU Technical Report CMU-CS-TR-98-175. A much earlier version appears in LCR '98 and as CMU-CS-TR-98-143.

[9] P. A. Dinda and D. R. O'Hallaron. An extensible toolkit for resource prediction in distributed systems. Technical Report CMU-CS-99-138, School of Computer Science, Carnegie Mellon University, July 1999.

[10] P. A. Dinda and D. R. O'Hallaron. Host load prediction using linear models. *Cluster Computing*, 3(4), 2000. An earlier version appeared in HPDC '99.

[11] R. Govindan and H. Tangmunarunkit. Heuristics for internet map discovery. In *IEEE INFOCOM 2000*, Tel Aviv, Israel, March 2000.

[12] N. C. Groschwitz and G. C. Polyzos. A time series model of long-term NSFNET backbone traffic. In *Proceedings of the IEEE International Conference on Communications (ICC'94)*, volume 3, pages 1400–4, May 1994.

[13] T. Gross and P. Steenkiste. A perspective on network/application coupling. In *Proc. 8th NOSSDAV Workshop (Network and Operating System Services for Digital Audio and Video), www.nossdav.org/1998/-tech*, 1998. Short paper.

[14] M. Hemy, P. Steenkiste, and T. Gross. Evaluation of adaptive filtering of MPEG system streams in IP networks. In *IEEE Intl. Conference on Multimedia and Expo 2000*, New York, 2000.

[15] S. Jamin, C. Jin, Y. Jin, D. Raz, Y. Shavitt, and L. Zhang. On the placement of internet instrumentation. In *IEEE INFOCOM 2000*, Tel Aviv, Israel, March 2000.

[16] R. Karrer and T. Gross. Dynamic handoff of multimedia streams. In *Proc. of NOSSDAV '01*, Port Jefferson, NY, 2001.

[17] B. Lowekamp, N. Miller, D. Sutherland, T. Gross, P. Steenkiste, and J. Subhlok. A resource query interface for network-aware applications. *Cluster Computing*, 2(2):139–151, 1999.

[18] B. Lowekamp, D. R. O'Hallaron, and T. Gross. Topology discovery for large ethernet networks. In *Proceedings of SIGCOMM 2001*. ACM, August 2001.

[19] N. Miller and P. Steenkiste. Collecting network status information for network-aware applications. In *IEEE INFOCOM 2000*, Tel Aviv, Israel, March 2000.

[20] K. Obraczka and G. Gheorghiu. The performance of a service for network-aware applications. In *Proceedings of the ACM SIGMETRICS SPDT'98*, October 1997. (also available as USC CS Technical Report 97-660).

[21] R. Ribler, J. Vetter, H. Simitci, and D. Reed. Autopilot: Adaptive control of distributed applications. In *Proceedings of the Seventh IEEE International Symposium on High Performance Distributed Computing (HPDC 7)*, pages 172–179, 1998.

[22] M. Samadani and E. Kalthofen. On distributed scheduling using load prediction from past information. Abstracts published in Proceedings of the 14th annual ACM Symposium on the Principles of Distributed Computing (PODC'95, pp. 261) and in the Third Workshop on Languages, Compilers and Run-time Systems for Scalable Computers (LCR'95, pp. 317–320), 1996.

[23] M. Stemm, S. Seshan, and R. Katz. Spand: Shared passive network performance discovery. In *USENIX Symposium on Internet Technologies and Systems*, Monterey, CA, June 1997.

[24] W. Theilmann and K. Rothermel. Dynamic distance maps of the internet. In *IEEE INFOCOM 2000*, Tel Aviv, Israel, March 2000.

[25] B. Tierney, R. Aydt, D. Gunter, W. Smith, V. Taylor, R. Wolski, M. Swany, and T. G. P. W. Group. A grid monitoring service architecture. DRAFT. http://www-didc.lbl.gov/GridPerf/documents.html, February 2001.

[26] B. Tierney, B. Crowley, D. Gunter, M. Holding, J. Lee, and M. Thompson. A monitoring sensor management system for grid environments. In *Proceedings of the Ninth IEEE International Symposium on High Performance Distributed Computing (HPDC 9)*, pages 97–104, 2000.

[27] B. Tierney, W. Johnston, B. Crowley, G. Hoo, C. Brooks, and D. Gunter. The netlogger methodology for high performance distributed systems performance analysis. In *Proceedings of the Seventh IEEE International Symposium on High Performance Distributed Computing (HPDC 7)*, pages 260–267, 1998.

[28] H. Tong. *Threshold Models in Non-linear Time Series Analysis*. Number 21 in Lecture Notes in Statistics. Springer-Verlag, 1983.

[29] J. Veizades, E. Guttman, C. Perkins, and S. Kaplan. Service location protocol. Internet RFC 2165, June 1997.

[30] R. Wolski. Forecasting network performance to support dynamic scheduling using the network weather service. In *Proceedings of the 6th High-Performance Distributed Computing Conference (HPDC97)*, pages 316–325, August 1997. extended version available as UCSD Technical Report TR-CS96-494.

[31] R. Wolski, N. Spring, and J. Hayes. The network weather service: A distributed resource performance forecasting service for metacomputing. *J. Future Generation Computing Systems*, 15(5-6):757–768, October 1998. Published also as UCSD Technical Report Number CS98-599.

[32] R. Wolski, N. Spring, and J. Hayes. Predicting the CPU availability of time-shared unix systems. In *Proceedings of the Eighth IEEE Symposium on High Performance Distributed Computing HPDC99*, pages 105–112. IEEE, August 1999. Earlier version available as UCSD Technical Report Number CS98-602.

[33] R. Wolski, N. Spring, and C. Peterson. Implementing a performance forecasting system for metacomputing: The network weather service. In *Supercomputing '97*, 1997.