Appears in the 7th International Symposium on High-Performance Distributed Computing

A Resource Query Interface for Network-Aware Applications

Bruce Lowekamp Nancy Miller Dean Sutherland
Thomas Gross* Peter Steenkiste Jaspal Subhlok

{lowekamp,nam,dfsuther,trg,prs,jass}@cs.cmu.edu

School of Computer Science, Carnegie Mellon University, Pittsburgh PA 15213

Abstract

Development of portable network-aware applications demands an interface to the network that allows an application to obtain information about its execution environment. This paper motivates and describes the design of Remos, an API that allows network-aware applications to obtain relevant information. The major challenges in defining a uniform interface are network heterogeneity, diversity in traffic requirements, variability of the information, and resource sharing in the network. Remos addresses these issues with two abstraction levels, explicit management of resource sharing, and statistical measurements. The flows *abstraction captures the communication between nodes, and the* topologies *abstraction provides a logical view of network connectivity. Remos measurements are made at network level, and therefore information to manage sharing of resources is available. Remos is designed to deliver* best effort *information to applications, and it explicitly adds statistical reliability and variability measures to the core information. The paper also presents preliminary results and experience with a prototype Remos implementation for a high speed IP-based network testbed.*

*Thomas Gross is also with ETH Zurich, Switzerland

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.

1 Introduction

Networked systems provide an attractive platform for a wide range of applications, yet effective use of the resources is still a major challenge. A networked system consists of compute nodes (hosts), network nodes (routers and switches), and communication links. Network conditions change continuously due to the sharing of resources, and when resource demands exceed resource availability, application performance suffers. Congestion on network nodes and links can reduce the effective bandwidth and increase the response time observed by applications. On compute nodes, competing jobs and higher-priority activities reduce availability. An attractive way of dealing with such changes is to make applications *system-aware*, i.e., the application periodically adapts to the system in an application-specific manner.

A variety of applications can benefit from system-awareness, and the manner in which they adapt to their execution environment varies widely. Communication intensive scientific simulations can select the optimal set of nodes for execution based on runtime information about the system state. Wide-area distributed applications, such as videoconferencing and digital libraries, can dynamically select the best servers and routes to best satisfy service requests. Finally, applications can be designed to tradeoff precision in order to meet timing requirements in response to a reduction in resource availability.

The Remos system developed at Carnegie Mellon University provides applications with a query-based interface to their execution environment including the network state. It is designed for experiments with the coupling of network-aware applications and network architectures. The Remos system has its roots in two separate projects: an investigation of resource management in application-aware networks (Darwin) and an effort to support system-aware applications through libraries, frameworks, and tools (Remulac).

Network-aware applications must be able to obtain information about resource availability, in particular, the network's capabilities and status. Unfortunately, network architectures differ significantly in their ability to provide such information to a host or to an executing application To avoid dependences on the idiosyncrasies of network architectures and communication systems, application development for networks requires a system-independent interface between applications and networks. A uniform interface allows the development of portable applications that can adapt on a range of network architectures. Furthermore, a system-independent interface is crucial for allowing applications to adapt on heterogeneous networks, where components are realized with different network technologies.

This paper describes Remos, which represents our solution to high-level and portable interaction between a network and network-aware applications. Remos is a standard interface for applications to obtain the relevant network information needed for effective adaptation. We present motivation for Remos, describe the design and initial implementation, and present preliminary results. We conclude with comparison to related work and our view of the directions of future Remos related research.

2 Design challenges

In this section we briefly summarize the problems that a uniform, portable network query interface like Remos must address:

- *Dynamic behavior*: Network conditions can change quickly, so the interface must characterize this dynamic aspect. Different applications may want this information in different formats, e.g., some applications are more interested in burst bandwidth while others primarily care about long term average bandwidth. Furthermore, applications are most interested in future network behavior, not historical information, and hence it is important to provide expected future behavior, particularly when reasonable basis for prediction (e.g., existing reservations) exist.

- *Sharing (internal and external)*: Application level *connections* between computation nodes share physical links with other connections. This dynamic sharing of resources with entities external to an application is the major reason for the variable network performance experienced by applications. Parallel and distributed computations often simultaneously transfer data across multiple point-to-point connections, so there is also internal sharing as these connections compete with each other for resources. An interface that provides performance data must therefore consider the network's sharing policy when estimating bandwidth availability.

- *Information diversity*: Applications may be interested in a wide variety of information, ranging from static network topology, dynamic bandwidth estimations on a variety of times scales, and latency information. This information may have to be retrieved through diverse mechanisms ranging from querying information databases using standard protocols to direct measurements on the network interface.

- *Network heterogeneity*: Network architectures differ significantly in their ability to collect and report network information such as communication performance, link capacities and utilization, and network topology. The nature of information that is made available by a network architecture impacts the design, overheads, and accuracy of information associated with a query interface.

- *Level of abstraction*: One of the thorny issues in designing an interface between applications and networks is to decide what aspects of the network should be exposed to applications. One solution is to expose as much information as possible. However, exporting low level or system specific information conflicts with other goals, particularly the portability of the API across heterogeneous networks. It also raises both ease of use and scalability concerns (the Internet is too huge for exposing all information). An alternative is to provide the information at a much higher level, with focus on performance characteristics of interest to the application. A high level of abstraction can make the interface easier to use and avoid information overload, but it can also result in vagueness and inaccuracies.

- *Accuracy and fidelity:* The information provided by an interface between networks and applications may not be completely accurate, e.g., the estimates of the future bandwidth may not be able to take all traffic into account. Nor may the information be of high fidelity, e.g., if the average bandwidth over a large interval is computed based on a small number of measurements. The issue then is how much accuracy and fidelity should be demanded (from the network) or promised (to the application).

3 Remos API

Remos is a query-based interface that provides "best-effort" information to a client. The applications specifies the kind of information it needs and Remos supplies the best available information. To limit the scope of a query, the application may select network parameters and parts of a larger network that are of interest. We sketch the main features of the Remos API and explain how they address the challenges presented above. A complete API is published in [4]. We organize our discussion around the three major design issues: the level of abstraction, dynamic behavior and sharing, and accuracy.

3.1 Level of abstraction

To accommodate the diverse application needs, the Remos API provides two levels of abstraction: a higher level based on flow queries and a lower level based on network topology.

Remos supports flow-based queries. A *flow* is an application-level connection between a pair of computation nodes. Queries about bandwidth and latency of sets of flows form the core of the Remos interface. Using flows instead of physical links provides a level of abstraction that makes the interface portable and independent of system details. Flow-based queries present the API implementation with the challenge of translating network specific information into solutions to specific queries. However, they allow the application developer to write adaptive network applications that are independent of heterogeneity inherent in a network computing environment. Past experience indicates that the flow abstraction should be easy to use by application developers. We describe the basic form of flow queries later in this section.

Remos also supports queries about the network *topology*. The reason we expose a network-level view of connectivity is that certain types of questions are more easily or more efficiently answered based on topology information. For example, finding the pair of nodes with the highest bandwidth connectivity may be expensive if only flow-based queries are allowed. The topology information provided by Remos consists of a graph with compute nodes, network nodes, and links, each annotated with their physical characteristics, such as latency and available bandwidth. Topology information is in general harder to use, since the complexity of translating network-level data into application-level information is partially left to the user.

Topology queries return a *logical* interconnection topology in the form of a graph. This graph represents the network behavior as seen by the application; the graph does not necessarily reflect the physical topology. Using a logical topology gives Remos the option of hiding network features that do not affect the application. E.g., if the routing rules imply that a physical link is not used, or can be used only up to a fraction of its capacity, then that information is reflected in the graph. The graph can also represent half or full-duplex bidirectional links. Similarly, if two sets of hosts are connected by a complex network (e.g. the Internet), Remos can represent this network by a single link with appropriate statistical characteristics.

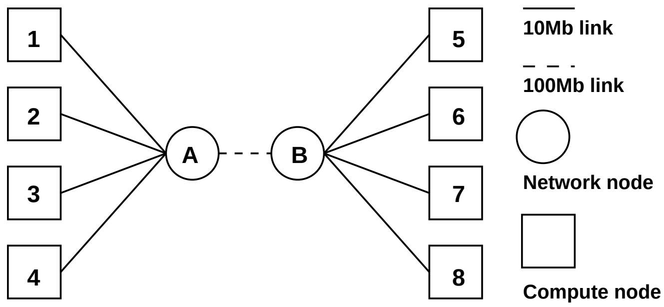


Figure 1. Remos graph representing the structure of a simple network

Figure 1 shows a simple example of a logical topology graph. Circles represent network nodes (e.g., routers) and squares indicate compute nodes (endpoints). Note that even though we show only the bandwidth capacities of the links, the topology also can also include capacities of nodes. E.g., if A or B have an internal bandwidth of less then 100 Mbps, then this restriction should be represented in the topology. Figure 1 suggests a simple topology consisting of two routers, eight endpoints, and nine physical links. However, since Figure 1 is a *logical* topology, it can represent a much broader set of (physical) networks. E.g., the link between A and B could represent a complex network. However, the internal details of that network are not relevant to the communication performance between the eight endpoints, so Remos turns this network into a single link with the bandwidth reflecting what the network can offer to this application.

The basic network topology query has the following form:

```
remos_get_graph(nodes, graph, timeframe)
```

The query fills a `graph` that is relevant for connecting the `nodes` set. The graph will contain all the nodes specified in the `nodes` parameter, as well as nodes that are part of the network that connects the specified nodes and the relevant communication links. The information annotated to the graph is relevant for the specified `timeframe`, which represents a window of time in the past or the future.

3.2 Dynamic resource sharing

Applications can generate flows that cover a broad spectrum, ranging from constrained low-bandwidth audio to bursty high-bandwidth data flows. Remos collapses this broad spectrum into three types of flows. *Fixed flows* have a specific bandwidth requirement. *Variable flows* have related requirements and demand the maximum available bandwidth that can be provided to all such flows in a given ratio. (e.g., all flows in a typical all-to-all communication operation have the same requirements) Finally, *independent flows* simply want maximum available bandwidth. These flow types also reflect priorities when sufficient resources are not available to satisfy all the flows. *Fixed flows* are considered first, followed by *variable flows*, then *independent flows*.

Applications can specify all flow requirements in their request simultaneously, and Remos takes internal resource sharing into account when responding to the queries. E.g., if multiple application flows share the $A - B$ link in Figure 1, then Remos will take the shared link into account, and the sum of the reported available bandwidths for all the flows will not exceed the bandwidth of the $A - B$ link.

A general flow query has the following form:

```
remos_flow_info(fixed_flows,
    variable_flows, independent_flows,
                              timeframe)
```

Remos will first try to satisfy the `fixed_flows` and allocate bandwidth for these flows. Next, bandwidth is assigned to the `variable_flows` simultaneously in the ratio specified in the query. Finally, `independent_flows` are satisfied from the capacity that is available after previous assignments. The corresponding data structures are filled with assignments to the flow requests based on the `timeframe` parameter.

Determining how the throughput of a flow is affected by other messages in transit is very complicated and network specific, and it is unrealistic to expect Remos to characterize these interactions accurately. In general, Remos assumes that, all else being equal, the bottleneck link bandwidth is shared equally by all flows (that are not bottlenecked elsewhere). If other information is available, Remos can use different sharing policies when estimating flow bandwidths. The basic sharing policy assumed by Remos corresponds to the max-min fair share policy [11], which is the basis of ATM flow control for ABR traffic [13, 2], and is also used in other environments [9].

Note that clients of the topology interface are responsible for accounting for sharing effects, both across application flows, and between application flows and other competing flows. This is one of the reasons to prefer the query interface over the topology interface when both can provide equivalent information.

3.3 Accuracy

Applications ideally want information about the level of service they can expect to receive in the future, but most users today must use past performance as a predictor of the future. Different applications are also interested in activities on different timescales. A synchronous parallel application expects to transfer bursts of data in short periods of time, while a long running data intensive application may be interested in throughput over an extended period of time. For this reason, relevant queries in the Remos interface accept a timeframe parameter that allows the user to request data collected and averaged for a specific time window. The user can also request the expected availability of resources in a window in the future, and models for estimation of future resource availability are being investigated.

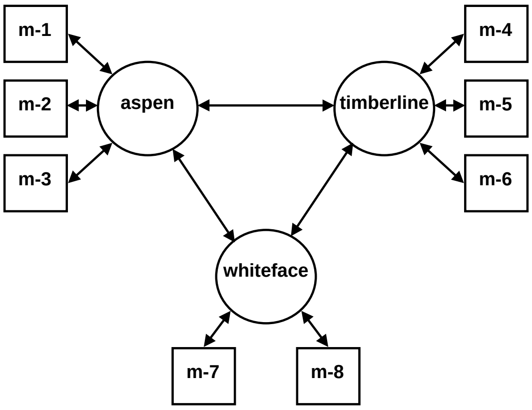
Network information such as available bandwidth varies dynamically due to sharing, and often cannot be measured accurately. As a result, characterizing these metrics by a single number can be misleading. E.g., knowing that bandwidth availability has been very stable represents a different scenario from it being an average of rapidly changing instantaneous bandwidths. To address these aspects, the Remos interface adds statistical variability and estimation accuracy parameters to all dynamic quantitative information. Since the actual distributions for the measured quantities are generally not known, we present the variability of network parameters using quartiles [12].

4 Implementation

The initial implementation of the Remos interface is for a dedicated IP-based testbed at Carnegie Mellon illustrated in Figure 2. The testbed uses PCs running NetBSD as flexible routers, 100Mbps point-to-point Ethernet segments as links, and DEC Alpha systems as endpoints.

The Remos implementation has two components, a Collector and Modeler; they are responsible for network-oriented and application-oriented functionality, respectively. The Collector consists of a process that retrieves information from routers using SNMP [3]. The information obtained covers both static topology and dynamic bandwidth information. For latency, the Collector currently assumes a fixed per-hop delay. (A reasonable approximation as long as we use a LAN testbed). A large environment may require multiple Collectors. The Modeler is a library that can be linked with the application. It satisfies application requests based on the information provided by the Collector. Its primary tasks are generating a logical topology, associating appropriate static and dynamic information with each of the network components, and satisfying flow requests based on the logical topology. The modeler exports Remos information through a Java and a C interface.

We would like to point out that the Remos API is independent of its implementation, and the suitable implementation techniques will be different for networks of different types and sizes. The above choices were made for a particular network environment that was available to us. However, this implementation will work on similar networks that use SNMP-capable routers.



Links: 100Mbps point-to-point ethernet
Endpoints: DEC Alpha Systems (*manchester-* labeled m-**)
Routers: Pentium Pro PCs running NetBSD *(aspen,timberline, whiteface)*

Figure 2. IP-based testbed for Remos implementation and experiments

5 Usage and results

The Remos interface can be used in a variety of ways to develop network-aware applications, ranging from dynamic selection of network resources for execution, to customization of the program execution in response to changes in system resources. In this paper, we present a preliminary report on the usage of Remos for selection of the best set of nodes for the execution of parallel and distributed programs.

The node selection algorithm will be referred to as *clustering*. It selects n nodes that are closest to a given set of one or more nodes. Clustering has been implemented as a user level library routine that proceeds as follows. First, it calls a Remos routine to obtain the logical topology of the relevant graph, as follows:

```
remos_get_graph(nodes, graph, timeframe)
```

For our experiments, `nodes` is the set of nodes on our IP-based testbed, and the topology of the testbed is returned in `graph`. The `timeframe` parameter is obtained from the application and specifies whether only static capacity information should be appended to the graph, or dynamic information relating to a window in history should be used.

Once the topology is obtained, the clustering routine repeatedly measures the aggregate *distance* between the current set of nodes in the cluster and all other nodes, and adds the node which is the *closest* neighbor. The details are beyond the scope of this paper, and we simply state that the *distance* is based on the bandwidth and latency estimates returned by Remos, and the algorithm uses a greedy heuristic for adding nodes. The clustering routine is a client of Remos information and we will refer to the procedure as *Remos based clustering*.

We now present results on the usage of clustering on two programs: *fast Fourier transforms (FFT)* and *Airshed pollution modelling*. The FFT program performs a two dimensional FFT, which is parallelized such that it consists of a set of independent 1 dimensional row FFTs, followed by a transpose, and a set of independent 1 dimensional column FFTs. Airshed contains a rich set of computation and communication operations, as it simulates diverse chemical and physical phenomena [18]. Both programs were developed using the Fx compiler system [8, 19] and easily modified to use the clustering information. All experiments were performed on the IP-based testbed illustrated in Figure 2.

We first examine the value of node selection when there are no competing applications on the testbed. We should point out that our testbed presents a difficult test for the clustering routine using Remos. Since the testbed is connected with high performance links and switches throughout, and any node can be reached from any other node with at most 3 hops, it is difficult to judge what sets of nodes are better connected than others.

The FFT program and the Airshed program were executed on sets of nodes selected by the Remos based clustering routine, and the results were compared with execution on other representative sets of nodes. In all examples, the programs gave node *m-4* as a start node and used clustering to select remaining nodes. The results are presented in Table 1. We observe that the execution time on the nodes selected by clustering was generally (but not always) lower than for other node sets, but only by relatively small amounts. As noted earlier, this is a hard case for Remos based clustering, but our toolchain is doing a fair job of node selection. However, we note that node selection has limited importance on a small network with fast routers and links and no external traffic.

To study the importance of node selection in the presence of competing applications, we performed another set of experiments. We added a synthetic program that generates communication traffic between nodes *m-6* and *m-8* on our testbed. The program used was `netperf` and it continuously sent as much data as possible between the pair of nodes for the duration of the experiments.

Once again the Remos based clustering routine was used for node selection in this environment. The programs were then executed on the selected sets of nodes. For comparison, the programs were re-executed on a set of nodes that the clustering routine could have potentially selected if it only used the static physical capabilities of the testbed communication nodes and links. The results are tabulated in Table 2.

We observe that the execution times are larger by 80-200 percent when dynamic traffic information is not used for node selection. The reason is that selection of nodes using Remos measured dynamic traffic makes it possible to avoid links with heavy traffic that are potential communication bottlenecks for a parallel program. The table also shows the performance of the programs in the absence of the external traffic (last column). We observe that with our dynamic node selection, the performance degrades only marginally in the presence of traffic, while a naive selection of nodes (we chose a *good* set of nodes without considering traffic) can lead to a dramatic degradation in the performance. The conclusion is that node selection is very important for performance in dynamic environments with competing traffic, and our greedy clustering routine using Remos is effective in selecting good sets of nodes in such realistic situations.

6 Related Work

A number of resource management systems allow applications to make queries about the availability of computation resources, some examples being Condor [14] and LSF (Load Sharing Facility). In contrast, Remos is motivated by network driven applications and focuses primarily on communication resources.

Resource management systems for large scale internet-wide computing is an important area of current research, and some well known efforts are Globus [6] and Legion [7]. These systems provide support for a wide range of functions such as resource location and reservation, authentication, and remote process creation mechanisms. The Remos interface focuses on providing good abstractions and support for application level access to network status information, and allow for a closer coupling of applications and networks. The Remos system can be used to extend the functionality of the above systems.

Recent systems that focus on measurements of communication resources across internet wide networks include Network Weather Service (NWS) [20] and *topology-d* [15]. NWS makes resource measurements to predict future resource availability, while *topology-d* computes the logical topology of a set of internet nodes. Both these systems actively send messages to make communication measurements between pairs of computation nodes. An important characteristic that distinguishes Remos is that applications interact with a portable interface to the network that includes flow query and logical topology abstractions. The network measurements are used to answer queries but not reported directly to applications. Remos implementations make measurements at network level when possible, which minimizes the measurement overhead and yields key information for managing sharing of resources.

A number of groups have looked at the benefits of explicit feedback to simplify and speed up adaptation [10, 5]. However, the interfaces developed by these efforts have been designed specifically for the scenarios being studied, while Remos is a general interface.

A number of sites are collecting Internet traffic statistics, e.g., [1]. This information is not in a form that is usable for applications, and it is typically also at a coarser grain than what applications are interested in using. Another class of related research is the collection and use of application specific performance data, e.g., a Web browser that collects information the response times of different sites [17].

An important aspect of the Remos interface is that dynamic quantities are reported with statistical parameters and not just as single values. Related work in estimating stochastic values such as [16] is important for developing good Remos implementations.

7 Concluding remarks

Remos is an attempt to develop a uniform interface between networks and network-aware applications that covers a range of network architectures. It allows applications to discover, at runtime, the properties of their execution environments. Such interfaces can also be of value to tools that attempt to place computations onto network nodes.

Remos' goal of a single portable interface between applications and different types of networks raises many new questions that will be addressed in detail in future research. The primary problem is the implementation of a single API on diverse network environments, spanning local area to

Problem Description		Remos Selected Nodes		Other Representative Node Sets					
Name of Program	No. of Nodes	Node set	Exec. Time (secs)	Node set	Exec. Time (secs)	Percent increase	Node set	Exec. Time (secs)	Percent increase
FFT (512)	2	m-4,5	.462	m-1,4	.468	1.3	m-4,8	.481	4.1
FFT (512)	4	m-4,5,6,7	.266	m-1,2,4,5	.287	3.7	m-1,4,6,7	.268	.03
FFT (1K)	2	m-4,5	2.63	m-1,4	2.66	1.1	m-4,8	2.68	1.9
FFT (1K)	4	m-4,5,6,7	1.51	m-1,2,4,5	1.62	7.3	m-1,4,6,7	1.61	6.6
Airshed	3	m-4,5,6	908	m-4,6,8	907	-.1	m-1,4,7	917	1.1
Airshed	5	m-4,5,6,7,8	650	m-1,2,3,4,5	647	-.4	m-1,2,4,5,7	657	1.1

Table 1. Comparison of performance of FFT and Airshed parallel programs on computation nodes selected using Remos, and two other sets of nodes that are representative of alternate choices without Remos knowledge

Problem Description		Execution Time with External Traffic					Exec. time without External Traffic (secs)
		Remos selected Nodes (Dynamic Measurements)		Nodes selected with only Static Measurements			
Name of Program	No of Nodes	Node Set	Time (secs)	Node Set	Time (secs)	Percent increase	
FFT (512)	2	m-4,5	.475	m-4,6	1.40	194	.462
FFT (512)	4	m-1,2,4,5	.322	m-4,5,6,7	.893	177	.266
FFT (1K)	2	m-4,5	2.68	m-4,6	7.38	175	2.63
FFT (1K)	4	m-1,2,4,5	2.07	m-4,5,6,7	3.71	79	1.51
Airshed	3	m-1,4,5	905	m-4,5,6	2113	133	908
Airshed	5	m-1,2,3,4,5	674	m-4,5,6,7,8	1726	156	650

Table 2. Performance implications of node selection using Remos in the presence of external traffic. Measurements use a synthetic program that generates significant traffic between nodes m-6 and m-8 on our IP-based testbed

internet-wide networks and various types of present and future network implementation technologies. Our implementation is based on SNMP measurements on the router nodes in the network. To extend the implementation to more complex networks, we will have to address bridges and other devices that necessitate different measurement methodologies. We have emphasized the use of passive measurements that do not generate new traffic, and demonstrated that this approach can yield satisfactory results. However, for internet-wide networks, active measurements are often the only option, and the best way to use them is currently an active area of research. The Remos API allows queries for future availability of resources, and includes fields for statistical variability and accuracy. However, techniques to compute these fields in different network environments continues to be a topic of research.

The current Remos API does not provide adequate support for multicast or reservations. These features can be added when they are more widely available to applications. The current Remos API focuses on performance aspects. At a later time, other dimensions, such as cost (of transmitting data via a specific link) and security (of physical links in the network) can be included.

The Remos API is a realistic interface to address the concerns discussed in this paper; a prototype implementation demonstrates that the interface can be realized and has value. A uniform interface for network-aware applications significantly reduces the difficulty of coupling an application to a network. As network architectures continue to evolve, we anticipate that interfaces like Remos will play an important role in raising the acceptability and practicality of network-aware applications.

References

[1] http://www.nlanr.net/INFO.

[2] ATM User-Network Interface Specification. Version 4.0, 1996. ATM Forum document.

[3] CASE, J., MCCLOGHRIE, K., ROSE, M., AND WALDBUSSER, S. Protocol Operations for Version 2 of the Simple Network Management Protocol (SNMPv2), January 1999. RFC 1905.

[4] DEWITT, T., GROSS, T., LOWEKAMP, B., MILLER, N., STEENKISTE, P., SUBHLOK, J., AND SUTHERLAND, D. Remos: A resource monitoring system for network-aware applications. Tech. Rep. CMU-CS-97-194, Carnegie Mellon University, Dec 1997.

[5] ECKHARDT, D., AND STEENKISTE, P. A Wireless MAC with Service Guarantees. In preparation, 1998.

[6] FOSTER, I., AND KESSELMAN, K. Globus: A metacomputing infrastructure toolkit. *Journal of Supercomputer Applications*. To appear.

[7] GRIMSHAW, A., WULF, W., AND LEGION TEAM. The Legion vision of a worldwide virtual computer. *Communications of the ACM 40*, 1 (January 1997).

[8] GROSS, T., O'HALLARON, D., AND SUBHLOK, J. Task parallelism in a High Performance Fortran framework. *IEEE Parallel & Distributed Technology 2*, 3 (Fall 1994), 16–26.

[9] HAHNE, E. L. Round-robin scheduling for max-min fairness in data networks. *IEEE Journal on Selected Areas in Communication 9*, 7 (September 1991).

[10] INOUYE, J., CEN, S., PU, C., AND WALPOLE, J. System support for mobile multimedia applications. In *Proceedings of the 7th International Workshop on Network and Operating System Support for Digital Audio and Video* (St. Louis, May 1997), pp. 143–154.

[11] JAFFE, J. M. Bottleneck flow control. *IEEE Transactions on Communications 29*, 7 (July 1981), 954–962.

[12] JAIN, R. *The Art of Computer Systems Performance Analysis*. John Wiley & Sons, Inc., 1991.

[13] JAIN, R. Congestion control and traffic management in ATM networks: Recent advances and a survey. *Computer Networks and ISDN Systems* (February 1995).

[14] LITZKOW, M., LIVNY, M., AND MUTKA, M. Condor — A hunter of idle workstations. In *Proceedings of the Eighth Conference on Distributed Computing Systems* (San Jose, California, June 1988).

[15] OBRACZKA, K., AND GHEORGHIU, G. The performance of a service for network-aware applications. Tech. Rep. TR 97-660, Computer Science Department, University of Southern California, Oct 1997.

[16] SCHOPF, J., AND BERMAN, F. Performance prediction in production environments. In *12th International Parallel Processing Symposium* (Orlando, FL, April 1998), pp. 647–653.

[17] STEMM, M., SESHAN, S., AND KATZ, R. Spand: Shared passive network performance discovery. In *USENIX Symposium on Internet Technologies and Systems* (Monterey, CA, June 1997).

[18] SUBHLOK, J., STEENKISTE, P., STICHNOTH, J., AND LIEU, P. Airshed pollution modeling: A case study in application development in an HPF environment. In *12th International Parallel Processing Symposium* (Orlando, FL, April 1998), pp. 701–710.

[19] SUBHLOK, J., AND YANG, B. A new model for integrated nested task and data parallel programming. In *Proceedings of the Sixth ACM SIGPLAN Symposium on Principles and Practice of Parallel Programming* (June 1997), ACM.

[20] WOLSKI, R., SPRING, N., AND PETERSON, C. Implementing a performance forecasting system for metacomputing: The network weather service. Tech. Rep. TR-CS97-540, University of California, San Diego, May 1997.