Chapter 1

PERFORMANCE OF NETWORK-BASED PROBLEM-SOLVING ENVIRONMENTS

Rajini I. Balay,
Mladen A. Vouk and
Harry Perros *
Center for Advanced Computing and Communications
and
Department of Computer Science
North Carolina State University.

Abstract The Network-Based Problem Solving Environment (NBPSE) paradigm is emerging as an important approach to complex problem solving, management and decision making. The quality of service and the performance of an NBPSE depend heavily on the communication infrastructure and middleware used to construct it. In this paper we discuss communication related performance issues through examples drawn from an operational NBPSE, a commercial middleware product, and a custom-made middleware designed for development of NBPSE prototypes. For example, in an IP-based high performance network-oriented environment, functionally extensive commercial communication middleware may operate as much as 2 or more times slower than specialized lightweight middleware. Furthermore, the way the problem solution is distributed and accessed, as well as the user activity level, can significantly impact field performance and the scalability of an NBPSE.

Keywords: Network-based Problem Solving Environments, middleware, software bus, performance evaluation, simulation model, latency, throughput, CORBA.

*This work was supported in part by MCNC/EPA grant C94-7050-027/029, NSF award ACS-9696131, IBM SUR grants and CACC award 5-3007

1. INTRODUCTION

A Problem Solving Environment (PSE) is a computer-based system that provides all the computational facilities necessary to solve a target class of problems efficiently. Modern network-based PSEs (NBPSE) are collections of distributed, cooperating interfaces, libraries, databases, programs, tools, clients, and intelligent agents which facilitate user interaction and cooperative execution of the components charged with the solution tasks (Gallopoulos et al., 1994; Rice and Boisvert, 1996; Singh and Vouk, 1999).

The need for effective and efficient communication among the PSE components (or objects) is obvious. This has resulted in a proliferation of communication building blocks, or **middleware**, for distributed scientific and other computing and problem solving. For effective NBPSE operation, a complete version of such middleware needs to support actively at least the following: 1) unicast and multicast messaging, 2) synchronous and asynchronous message processing, 3) efficient transfer of high-volume data, 4) ability to import information from and export information to processes that are not part of the NBPSE, 5) ability to invoke functions of a subsystem from any other process, and 6) last, but not least, end-to-end quality of service (QoS) (Balay, 1998). In many cases communication is based on the TCP/UDP/IP stack. Two examples of IP-based middleware are PVM and MPI. A large group of systems are also based on the CORBA-compliant commercial object brokers (Singh and Vouk, 1999). While middleware can greatly simplify development of NBPSEs, it usually introduces extra overhead. This overhead can be accentuated in environments where the resources (such as workstations and supercomputers) are interconnected by high-speed links (e.g., switched 100, 155 or 1000 Mbps links) because any middleware communication deficiencies may become a major performance bottleneck.

In general, NBPSE communication performance is a direct function of a) its communication and data management middleware, b) its architecture and topology, c) its application-level algorithms and solutions, and d) its usage patterns (including the number and distribution of users). In this paper we illustrate some issues related to items a), b) and d) using a combination of simulation and measurement-based examples from a real operational NBPSE called EDSS (based on special-purpose middleware called LSB), and from a CORBA based middleware suite called SOMobjects. In Section 2 we describe our empirical environment. In Section 3, we discuss our experiments, simulations and results. Conclusions are in Section 4.

2. EMPIRICAL ENVIRONMENT

The **Environmental Decision Support System** (EDSS) is an operational NBPSE developed by the MCNC Environmental Programs Group in cooperation with the US EPA and NC State University (Ambrosiano et al., 1995). EDSS was developed to help research scientists and environmental planners and managers model and evaluate environmental quality issues and make decisions at different levels of granularity. It was used as a prototype for the EPA's Models-3 (Dennis et al., 1996) system. EDSS runs on Unix and PC platforms and it is in daily use at a number of sites in the U.S.A. and world-wide.

EDSS supports air quality modeling through three principal subsystems, a data and model engineering and configuration management subsystem, a simulation planning and execution management subsystem, and an analysis and visualization subsystem. The simulation planning subsystem, called the *Study Planner*, provides a graphical user interface (GUI) through which users can graphically link executable programs into a graph-based description of the solution workflow, called a study. A study can then be executed either locally or in a distributed fashion with the help of a centralized *Scheduler* and a host of remote *Monitors*. Visualization support is provided by the Package for Analysis and Visualization of Environmental data (*PAVE*) (Thorpe et al., 1996). PAVE accesses the datasets remotely and allows users to "slice and dice" the data files at a distance through formulae and selection of data based on the regions and times of interest. In addition, EDSS has a number of administration and helper modules (e.g., Console, File Browser, Help).

The communication middleware used in EDSS is the **Lightweight Software Bus** (LSB) developed by NC State University (Balay et al., 1996; Balay et al., 1997). It is a message broker consisting of a set of client/server communication and message management routines written in C (BusMaster), and a client middleware layer. The client layer is a set of communication functions that reside between the client application and the TCP sockets, and which allow PSE components to communicate with each other and with the BusMaster without dealing with the socket layer communication details. In the case of EDSS, LSB "glue" provides coarse-grained network-based integration of the PSE components. An LSB client can be located on the machine that hosts the BusMaster, or it can be on any remote machine interconnected via TCP/IP.

BusMaster routes messages and answers queries, or BusFunction calls, from a client regarding system status, existence of another client, or about the message types registered in the system. BusFunction calls are *synchronous*, while message transfer requests between clients are *asyn-*

chronous. LSB clients support: GUIs, messages, direct communication channels, and socket interfaces.

The CORBA based experiments were conducted using Version 3.0 (IBM, 1996) of the IBM SOMobjects which is CORBA Version 1.1 compliant (OMG, 1995). SOMobjects uses a daemon process, called somdd, to manage server processes on a particular machine. SOMobjects implements both the Static Invocation Interface (SII) and the Dynamic Invocation Interface (DII), as defined in CORBA.

3. PERFORMANCE ISSUES

3.1 IMPACT OF MIDDLEWARE

In TCP/IP-based applications, the middleware resides in a layer between the TCP/IP protocol stack and the application code. Depending on the functionality, middleware can be classified either as lightweight or as full-function (or general-purpose). In the case of lightweight middleware, the functionality is limited to a subset of functions needed for the support of the immediate application. In the case of full-function middleware, all operations and communication modes supported by the middleware are available to an application. Unless the middleware architecture allows subsetting of its functionality to reduce the run-time overhead, applications based on a general-purpose middleware will tend to carry more communication overhead.

We assess middleware overhead in terms of a) message *latency* and b) data *throughput*. We use latency of a message with an empty body to determine the message header manipulation overhead. Throughput is the amount of data (payload) that can be transferred between communicating processes, through the middleware and the "network", per unit time. Ideally, middleware should provide low latency and throughput penalty. Unfortunately, this is not always the case.

Latency. For example, lightweight middleware can incur delays an order of magnitude smaller than a full-function commercial of the shelf (COTS) product. To illustrate this we compared a null-body asynchronous (oneway) message, and a null-body request-reply (twoway) message sent through LSB, with SOMobjects delays for oneway and synchronous (twoway) method invocations of the parameterless operations for the two invocation policies defined in CORBA (OMG, 1995), i.e., SII and DII. The latency of a *oneway* message in LSB was about 0.20 ms whereas it was 2.04 ms for SOM SII and 2.06 ms for SOM DII. In the case of a *twoway* messages, the latency in LSB was 1.83 ms, and it was 4.32 ms and 5.26 ms for SOM SII and DII respectively

(both operating between RS6000 AIX platforms over 155 Mbps ATM links). Similar experiments were conducted by Gokhale and Schmidt (Gokhale and Schmidt, 1997) using two other CORBA-compliant products, Orbix and ORBeline. Though their experiments were conducted using a somewhat different hardware configuration (they used Sun workstations), their results are illustrative of the range of overhead that can be incurred. They reported latencies of 0.5 ms for Orbix SII, 0.4 ms for Orbix DII, 0.1 ms for ORBeline SII and 0.1 ms for ORBeline DII, in the *oneway* message case. For *twoway* messages, the latencies were 2.7 ms for Orbix SII, 10.5 ms for Orbix DII, 2.0 ms for ORBeline SII and 2.0 ms for ORBeline DII [1].

Throughput. User-level throughput is the number of bytes of data a user application can transmit or receive in unit time using a reliable protocol. Full-function COTS middleware can reduce throughput to about 50% of what it could be without it. For example, let throughput over plain Berkeley sockets estimate the combined overhead due to the TCP/IP layer and the underlying "networking" infrastructure. The difference between the user-level throughput and the throughput via plain Berkeley sockets then estimates the "resistance" of the middleware. If the latter is large, the middleware (or the way it is being used) may not be suitable for an NBPSE. Typically we transfered 64 Mbytes in messages ranging in size from 1 Kbytes to 256 Kbytes. The maximal asynchronous throughput via plain Berkeley sockets was measured using TTCP (USNA, 1984). The experiment was mimicked in LSB and SOMobjects. The details are described in (Balay, 1998). For messages over about 1 Kbytes, LSB throughput was about 77% of the TTCP end-user throughput (typically about 110+ Mbps). In the case of non-brokered (direct) communication channels, the LSB throughput was consistently about 90% of the TTCP throughput. The difference was primarily due to connection establishment delays and the differences in the implementation of the direct channel mode and TTCP. In comparison, SOMobjects throughput was about half to one quarter that of LSB.

Of course, performance gains seen for the lightweight middleware, such as LSB, also must be compared with the more extensive functionality and development flexibility offered by a more complex full-function ORB.

3.2 ARCHITECTURE AND USAGE

An NBPSE is usually composed of modules distributed in various ways across a number of machines. Some of these modules can be computational repositories, or servers, that respond to requests from other

modules in the system. If they are the bottleneck, the whole NBPSE performance may suffer. For example, if a client (and in turn the user) that requests a service from another module is blocked until a reply is received, it is important that the duration of that blocking be short. In addition to client service time, the delay will be impacted by the NBPSE communication topology, and by the usage patterns of the individual platforms and modules. The **operational profile** (OP) of an NBPSE can be employed to describe its usage dynamics. OP analysis includes the number of users, the type, the sequence and the frequency of operations in the field, etc. (Musa, 1993; Musa, 1998). OPs can be mapped to traffic generated within an NBPSE network, and to the flows between the outside world and the NBPSE. A detailed analysis and simulation of NBPSE OP and workload is recommended prior to any finalization of its architecture or topology. We illustrate the influence of user traffic levels, inter-module communication complexity, the number of NBPSE elements, and the role of servers through a simulation.

Simulation. The first simulations were of 10 LSB clients with functionalities that were the same as those of the actual EDSS modules. The clients were located on two machines host1 and host2, with 7 and 3 clients on each machine respectively. The BusMaster process was on host1. We varied the load on the LSB clients by modifying the external arrival rate to four EDSS modules with GUIs: Console, Browser, Study Planner and PAVE. Each GUI client selected its messages using its operational profile, and the "targets" of its messages using a uniform random distribution. The time between successive user requests at these modules, defined as the mean inter-arrival time (IAT), was varied from 300 ms to 800 ms. It is worth noting that a typical interactive NBPSE session requires round-trip user-level response times of the order of 250 ms or less per key or mouse-stroke (Vouk and Singh, 1997). Delay accrues serially in the application itself, in the middleware, and in the underlying networking infrastructure. We investigated the impact of the variability in the user input through burstiness of user requests. Burstiness was defined by the squared coefficient of variation of IAT (C^2) in the range 1 to 20.

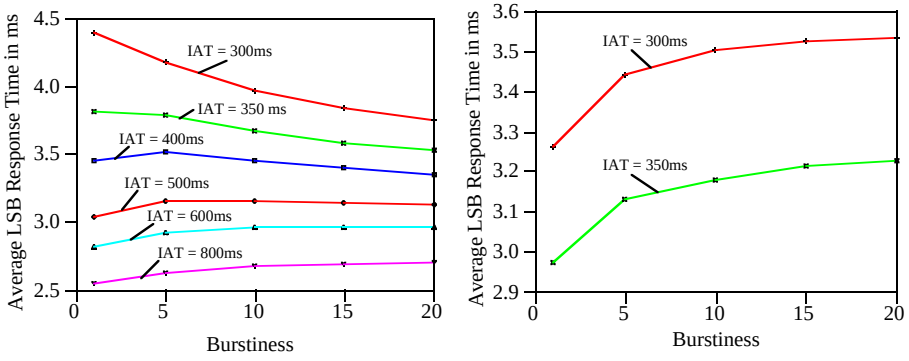
Measures. We use three measures for performance: (1) the *BusMaster, or LSB broker, response time*; (2) the *client-to-client round trip delay*; and (3) the *direct connection establishment delay.*

The middleware *response time* is the time that the BusMaster in LSB takes to respond to a synchronous BusFunction call from a client. It measures how the LSB middleware responds to a client request in differ-

ent situations. The *client-to-client round-trip delay* is the time elapsed between sending a request to another module in EDSS, and receiving a reply from it. This delay includes the processing delay at the LSB middleware as well as that at the other PSE module. The third measure, the *direct connection establishment delay* is the time taken by the LSB middleware to communicate with the destination client to establish a direct channel, i.e., a mechanism that bypasses message brokering. This delay also includes the middleware and client processing delay.

User Traffic. Variability in the user traffic loads can have larger impact than the actual intensity of the traffic. For a given IAT, the larger the C^2, the more pressure is put on the system. For example, when user activity is high (IAT=300 ms), an increase in burstiness from 1 to 20 increased the mean round-trip delay by a factor of 10 (from about 100 to about 1000 ms), and the mean direct connection establishment delay by a factor of almost 2 (from about 45 to abut 90 ms). Interestingly, the culprit for the 1000 ms average delays recorded for IAT of 300 ms and burstiness of 20 was not the middleware but the clients themselves. The delays observed at the BusMaster during the simulation were in the range of 2 to 5 ms. The rest of the overhead cames from the inability of the GUI clients to process all the requests. CPU load on those hosts did not exceed 90%, so the cause was not saturation in the computing resource. However, the waiting that is part of synchronous messaging did cause a considerable backlog at some clients. The delays improved significantly when IAT was over 500 ms. At IAT=800 ms, all mean round-trip delays (regardless of burstiness investigated) reduced to about 90 ms, and the direct connection establishment delays reduced to about 30 ms (Balay, 1998).

Client Communication Complexity. Individual client communication complexity can have considerable impact as well. For example, for every user input or message that EDSS PAVE receives, on the average, it sends N_i=6 external communication requests to other EDSS components. Each request can be either a BusFunction call, a message transfer request or a request for a direct connection. The probability that a message is synchronous is about 0.79, the probability that a message is asynchronous is negligible, and the probability that a request is for a direct connection is about 0.21. In this example, N_i and the probabilities of the different message types define the operational profile and the complexity of the communication behaviour of the PAVE module. Simulation results are interesting and somewhat counter-intuitive. They are shown in Figure 1.1.



(a) PAVE client with a large number of synchronous and direct connection requests

(b) Modified PAVE client with reduced number of synchronous and direct connection requests

Figure 1.1 Effect of Complex Interactions and Client Behaviour

With reference to part (a) of the figure, one might expect that, as the intensity of the user input increases, the broker response time would also increase. This was not always the case. For example, for a mean IAT of 400 ms, the BusMaster response time increased until the burstiness reached 5. After that, it decreased with further increase in the burstiness. The reason is that, as the user inputs became more irregular, the PAVE module was blocked for longer periods of time due to its high N_i value and the high probability that the request is a blocking direct connection. As a result, the rate at which PAVE issued requests to the BusMaster, and to the host CPU queue, was reduced. This indirectly caused the waiting time in the CPU queue to be reduced on the machine on which the BusMaster and PAVE resided. This in turn reduced the mean response time of the BusMaster. Fewer requests to the BusMaster, e.g. N_i=1, result in a faster response time - part (b) of the Figure 1.1.

Obviously, the communication complexity can have a strong effect. It is advisable to choose and evaluate NBPSE middleware based on an understanding of how individual components behave. However, we note that, in our experiments, middleware impact on performance was about an order of magnitude smaller than the effect of individual PSE elements (applications). We also note that in this case, neither host configurations nor the underlying networks were a significant bottleneck. This may change as the platforms become faster and more powerful.

Number of Networked NBPSE Elements. The number of modules/elements in the system affects its performance. Modules may enter

and leave an NBPSE dynamically. For example, a new module may be added to the system every time a new user enters the system. This is likely to generate a lot of new traffic. The other extreme is where new functionality is added to the system, but this new functionality is activated on demand and does not generate much extra traffic on its own. We examined the two extremes via an EDSS simulation where the number of modules was varied from 10 to 70 in steps of 10.

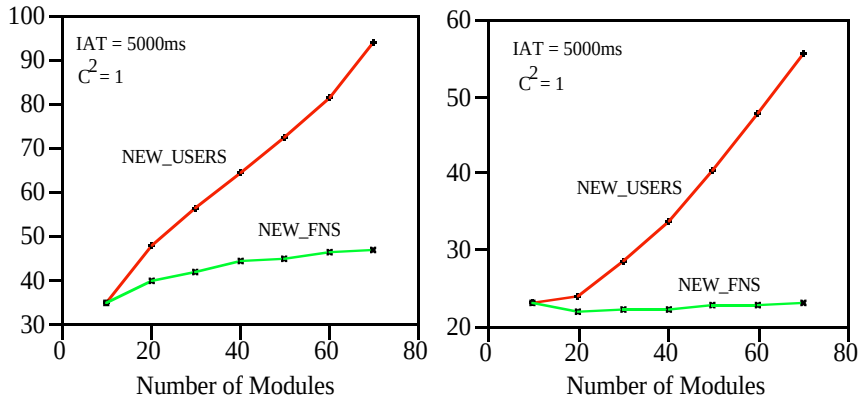


Figure 1.2 Performance as the number of modules in system varies

The results are illustrated in Figure 1.2. It shows the round-trip and direct connection establishment delays vs. number networked modules. As new modules with GUIs are added to the system (NEW_USERS), client-to-client delays increase because of the extra traffic they add to the system. However, when the added modules are not traffic generating (NEW_FNS), the delay increases are very mild, if any. Of course, this is expected, but this highlights the fact that for an NBPSE where modules/users join or leave NBPSE dynamically, their effect on the system should be anticipated.

Servers. Some clients in an NBPSE provide services that may be in frequent demand. We call these *servers*. To illustrate what happens to a system in which there is a high traffic server, we designated a module in the EDSS system as a *hot-client*; a module whose services are used more than that of others. Then we directed 80% of the messages in the system to this client. At the same time, we allowed direct connections among different clients (e.g., ftp traffic) on a uniform basis. We then

computed performance measures for two different types of hot-clients: (1) the client located on host1 (SVR_HOST1), i.e., a server on a heavily used machine, and (2) the client located on host2 (SVR_HOST2), a lightly used machine.

For the same simulation conditions, the delays encountered in the SVR_HOST2 case were less than in SVR_HOST1 case. This is because the utilization of the CPU node representing host1 was 10% lower than it was for the SVR_HOST1 case. The delays observed for a burstiness factor of 1 and a varying mean IAT are shown in Figure 1.3. Also shown

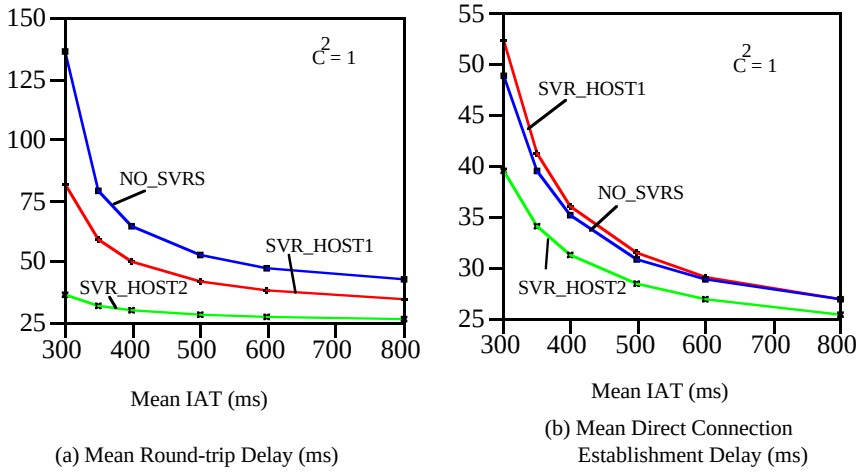


Figure 1.3 Effect of changing activity of servers

in the figure are the delay times observed for the case where the messages were distributed uniformly among all the clients (NO_SVRS). The average round-trip delay for both hot-client cases was lower than that for the NO_SVRS case because, the average per message service time on the hot-clients was lower. However, the BusMaster respone time showed an increase in the SVR_HOST1 from the NO_SVRS case, due to the increase in the CPU utilization of host1. On the other hand, the direct connection establishment delays for SVR_HOST1 and NO_SVRS were not significantly different since the destinations for a direct connection requests were chosen uniformly. The moral is that in a peer-oriented equal participation mode, something that might be more of a reality in future NBPSEs, any station may become a server for a while and the individual user desktop workstation will probably need to be as powerful as any element of the system lest it became the bottleneck.

4. SUMMARY

Middleware used for building an NBPSE can induce a significant amount of communication overhead. Specifically, more flexible and more comprehensive commercial middleware can introduce communication overhead that can be much larger than the overhead incurred when specialized middleware is built to support an NBPSE. However, other factors such as individual client performance, NBPSE topology and usage patterns, can affect the performance of an NBPSE even more. In order to design and build systems which satisfy user expectations, it is necessary to explore, model and predict performance of an NBPSE based on its functionality, middleware, network characteristics and usage profiles during its design phase. At this point in time it would appear that the available middleware performs an order of magnitude better than the applications and their solution topologies themselves. In fact a major performance bottleneck is likely to be the PSE software and its usage patterns. This may change as the new generation of more powerful desktop workstations become more readily available and the hosts become more evenly matched with the underlying networking infrastructure. But, unless there are rapid advances in the NBPSE user level quality of service support, applications themselves in combination with user habits may still be the dominant factor.

Notes

1. The latencies have been approximated from the graphs presented in (Gokhale and Schmidt, 1997) for a parameterless method invocation over a server with 1 object, using the round-robin algorithm.

References

Ambrosiano, J., Balay, R., Coats, C., Eyth, A., Fine, S., Hils, D., Smith, T., Thorpe, S., Turner, T., and Vouk, M. (1995). The Environmental Decision Support System: Air Quality Modeling and Beyond. In *Proceedings of the U.S. EPA Next Generation Environmental Modeling Computational Methods (NGEMCOM) Workshop, Bay City, Michigan.*

Balay, R. I. (1998). *A Lightweight Middleware Architecture and Evaluation of Middleware Performance.* PhD thesis, North Carolina State University.

Balay, R. I., Perros, H., and Vouk, M. A. (1997). The Lightweight Software Bus : A Message Broker for Prototyping Problem Solving Environments. In *Proceedings of the Thirteenth International Conference On Computer Communication*, pages 379–388. ICCC.

Balay, R. I., Vouk, M. A., and Perros, H. (1996). A Lightweight Software Bus for Prototyping Problem Solving Environments. In *Proceedings of the 11th International Conference on Systems Engineering*, pages 626–630.

Dennis, R. L., Byun, D. W., Novak, J. H., Galluppi, K., Coats, C., and Vouk, M. (1996). The Next Generation of Integrated Air Quality Modeling: EPA's Models-3. *Atmospheric Environment*, 3(12):1925–1938.

Gallopoulos, E., Houstis, E., and Rice, J. R. (Summer 1994). Computer as Thinker/Doer: Problem-Solving Environments for Computational Science. In *Problem-Solving Environments*, pages pp. 11–23. IEEE Computational Science & Engineering.

Gokhale, A. and Schmidt, D. C. (1997). Evaluating CORBA Latency and Scalability Over High-Speed ATM Networks. In *17th International Conference on Distributed Systems.*

IBM (1996). *SOMobjects Developer's Toolkit.* IBM, version 3.0 edition.

Musa, J. (1998). *Software Reliability Engineering.* McGraw-Hill, New York.

Musa, J. D. (1993). Operational Profiles in Software-Reliability Engineering. *IEEE Software*, pages 14–32.

OMG (1995). *The Common Object Request Broker: Architecture and Specification.* Object Management Group, Revision 2.0 edition.

Rice, J. R. and Boisvert, R. F. (1996). From Scientific Software Libraries to Problem-Solving Environments. *IEEE Computational Science & Engineering*, pages 44–53.

Singh, M. P. and Vouk, M. A. (1999). Network Computing. In Webster, J. G., editor, *Encyclopedia of Electrical and Electronics Engineering*, volume 14, pages 114–132. Joh Wiley & Sons, New York.

Thorpe, S., Ambrosiano, J., Coats, C., Eyth, A., Fine, S., Hils, D., Smith, T., Trayanov, A., Balay, R., and Vouk, M. (1996). The Package for Analysis and Visualization of Environmental Data. In *Proceedings of the Computing in Environmental Resource Management*, RTP, NC. Air and Waste Management Association.

USNA (1984). TTCP: a test of TCP and UDP Performance. Modified code for testing performance over an ATM network is available at http://renoir.csc.ncsu.edu/CSC402/ttcp.

Vouk, M. A. and Singh, M. P. (1997). Quality of Service and Scientific Workflows. In Boisvert, R., editor, *The Quality of Numerical Software: Assessment and Enhancements*, pages 77–89. Chapman and Hall.